A. Écriture d'un entier naturel dans une base b (b est un entier supérieur ou égal à 2) :

Activité 1 :

1. Calcule le nombre entier $3\times4^3+1\times4^2+2\times4^1+0\times4^0$ (**Rappel :** $x^0=1$ pour tout nombre x non nul).
Ce nombre est écrit comme la somme ***de puissances de 4*** multipliées par des entiers ***compris entre 0 et 3***.
On dira que ce nombre s'écrit $(3120)_4$ ***en base 4***.

2. A quel nombre entier est égal : ***a.*** $(3311)_4$. ***b.*** $(200)_3$. ***c.*** $(543210)_6$. ***d.*** $(708)_9$. ***e.*** $(8439)_{10}$.

3. a. Calcule les puissances de 2 ($2^0=$ … ; $2^1=$ … ; $2^2=$ … ; ..... ; $2^7=$ …).
b. Déduis en l'écriture du nombre entier 203 en base 2.
c. De la même manière écris 203 en base 5, puis en base 8.

4. a. Quelle est l'écriture de $2\times3^2+2$ en base 3 ?
b. Quelle est l'écriture de $4\times3^2+2\times3+5$ en base 3 ?

Ce qu'il faut savoir : Si N est un entier naturel tel que $N=a_k\times b^k+a_{k-1}\times b^{k-1}+\ldots\ldots+a_1\times b^1+a_0\times b^0$, b étant un nombre entier naturel supérieur ou égal à 2 et tous les nombres a_i étant ***des entiers compris entre 0 et b-1*** , alors on dit que $(a_k a_{k-1}\ldots a_1 a_0)_b$ est ***l'écriture en base b*** du nombre entier naturel N .

On notera $N=(a_k a_{k-1}\ldots a_1 a_0)_b$.

Remarque : On utilisera la notation $(\ldots\ldots\ldots\ldots\ldots)_b$ pour préciser dans quelle base est l'écriture.
On conviendra que si l'on n'écrit pas cette notation, l'écriture est en base « naturelle », c'est à dire la base 10.

B. Cas particulier de la base 2 :

Activité 2 :

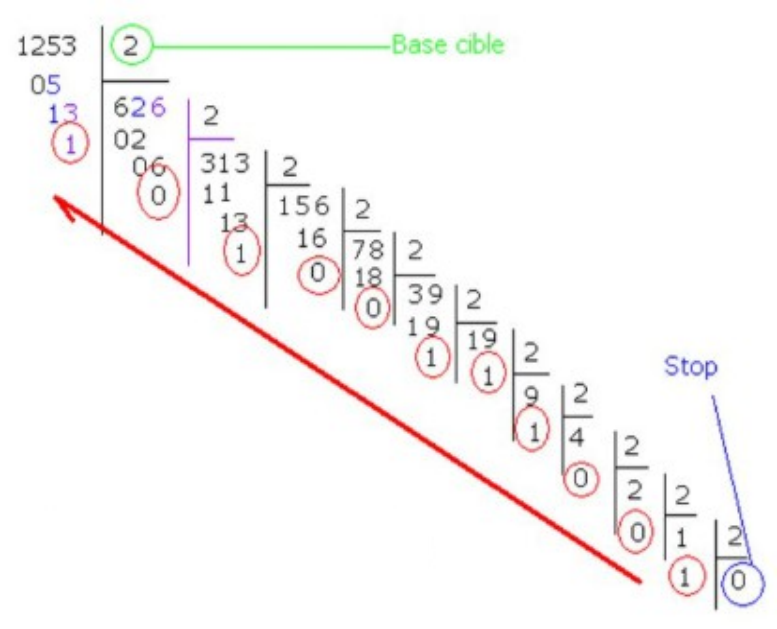
On veut écrire le nombre entier naturel 73 en base 2.
Pour cela on va chercher à l'écrire sous la forme $b_n\times2^n+\ldots+b_1\times2^1+b_0\times2^0$ car ainsi on obtient en base 2 le nombre $(b_n\ldots b_1 b_0)_2$.

1. Écris la division euclidienne de 73 par 2 et l'égalité euclidienne : $dividende=quotient\times diviseur+reste$.
Écris ensuite la division euclidienne du quotient obtenu par 2 et l'égalité euclidienne correspondante que tu inséreras dans l'égalité euclidienne précédente. Pour finir poursuis ainsi jusqu'à obtenir un quotient nul.

2. Déduis-en l'écriture de 73 sous la forme $b_n\times2^n+\ldots+b_1\times2^1+b_0\times2^0$, puis son écriture en base 2.

Ce qu'il faut savoir : Pour convertir un entier naturel (qui est en base 10) en base b avec $b \geqslant 2$, on divise cet entier naturel par b jusqu'à obtenir un quotient égal à 0.
Le résultat cherché est ***la juxtaposition des restes du dernier au premier***.

Exemple :



$1\,253 = \left(10011100101\right)_2$

Activité 3 : En utilisant la méthode ci-dessus, effectue les conversions ci-dessous.
1. Écris 47 , puis 53 et 245 en base 2.
2. Écris 67, puis 231 et 2 578 en base 7.

C. Cas particulier de la base 16 :

Activité 4 :
1. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 2 ?
2. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 5 ?
3. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 10 ?
4. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 12 ? Quel est le problème ?

Ce qu'il faut savoir : Pour convertir un nombre entier naturel N en base 16, il faut l'écrire sous la forme $N = a_k \times 16^k + a_{k-1} \times 16^{k-1} + \ldots\ldots + a_1 \times 16^1 + a_0 \times 16^0$, les a_i étant ***des entiers compris entre 0 et 15***.
On pose alors $c_i = a_i$ si $0 \leqslant a_i \leqslant 9$ et $c_i =$ A, B, C, D, E ou F si $a_i =$ 10, 11, 12, 13, 14 ou 15.
Et on dit que $\left(c_k c_{k-1} \ldots c_1 c_0\right)_{16}$ est ***l'écriture en base 16*** ou ***en hexadécimal*** du nombre entier naturel N .

On notera $\mathbf{N = \left(c_k c_{k-1} \ldots c_1 c_0\right)_{16}}$.

Exemple : $\left(A5\mathrm{E}\right)_{16} = 10 \times 16^2 + 5 \times 16^1 + 14 \times 16^0 = 2\,654$.

Activité 5 :
1. A quel nombre entier est égal : ***a.*** $\left(7\mathrm{DD}\right)_{16}$. ***b.*** $\left(2\mathrm{A}\right)_{16}$. ***c.*** $\left(4F2\mathrm{C}\right)_{16}$.
2. Écris chaque nombre entier en base 16 : ***a.*** 62. ***b.*** 1455. ***c.*** 8675.

D. Exercices d'application :

Exercice 1 :

1. A quel entier est égal : ***a.*** $(101010101)_2$. ***b.*** $(111000)_2$. ***c.*** $(00110011)_2$. ***d.*** $(101000001)_2$.

2. Convertis en binaire : ***a.*** 458. ***b.*** 133. ***c.*** 47. ***d.*** 1 024. ***e.*** 65.

Exercice 2 :

1. A quel entier est égal : ***a.*** $(A320)_{16}$. ***b.*** $(FABE51)_{16}$.

2. Convertis en hexadécimal : ***a.*** 2 020. ***b.*** 1 234. ***c.*** 56 026. ***d.*** 64 218.

Exercice 3 : Convertis en binaire les nombres écrits en hexadécimal suivants (On commencera par les convertir en base 10). ***a.*** $(101010)_{16}$. ***b.*** $(59A75)_{16}$

Exercice 4 :

1. On veut convertir en hexadécimal $(1001101)_2$. Pour cela, il suffit de compléter le tableau ci-dessous.

Écriture binaire « par paquets de 4 »	**0**100	1101
Écriture décimal « des paquets de 4 »		
Écriture hexadécimale		

On ajoute des « 0 » pour avoir un paquet de 4 chiffres.

Complète ce tableau et convertis $(1001101)_2$ en hexadécimal.

2. En utilisant la même méthode, convertis en hexadécimal les nombres ci-dessous.

a. $(10000000011001)_2$. ***b.*** $(10001000010001)_2$. ***c.*** $(100110000111)_2$. ***d.*** $(101110101100)_2$.

Exercice 5 :

1. Pose et effectue chacune des opérations ci-dessous en base 2. Pour vérifier, convertis chaque terme des opérations en base 10, refais les opérations en base 10 et compare le résultat avec celui obtenu en base 2.
Important : On utilisera le fait que en base 2 on a $(1)_2+(1)_2=(10)_2$ pour bien gérer les retenues.

a. $(100011)_2+(110100)_2$. ***b.*** $(100101)_2+(110000)_2$. ***c.*** $(100011)_2+(111)_2$. ***d.*** $(10000000)_2-(100)_2$.

2. Pose et effectue : ***a.*** $(98)_{16}+(B9)_{16}$. ***b.*** $(D23)_{16}+(46\text{A})_{16}$. ***c.*** $(150F6)_{16}-(7E3\text{A})_{16}$.

Exercice 6 : Considérons le programme écris ci-dessous en Python.

```
n = int(input("Entrez un nombre entier"))
b = ""
while n!= 0:
    q = int(n/2)
    b = str(n-q*2)+b
    n = q
print(b)
```

1. Recopie et complète ce tableau jusqu'à ce que le programme se termine lorsqu'on a choisi ***n=71*** :

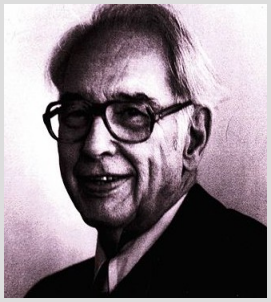
q		35	etc…
n-q*2		1	etc…
b	" "	" 1 "	etc…
n	71	35	etc…
n!=0	VRAI	VRAI	etc…

2. Explique le rôle de ce programme.

3. Tape ce programme sur Python et teste le sur certains résultats de cette fiche d'exercices.

A. Le principe de la numérisation de l'information :

Ce qu'il faut savoir :



John Atanasoff

• A la fin des années 1930, ***John Atanasoff*** (physicien et mathématicien américain d'origine bulgare) a mis au point avec un de ses étudiants, ***Clifford Berry***, l'ABC (Atanasoff Berry Computer), un ordinateur de 300 kg, capable d'effectuer une opération toutes les 15 secondes. L'ABC est considéré comme ***le premier ordinateur***. Pour créer cet appareil, Atanasoff a imaginé un système électronique composé d'interrupteurs ouverts ou fermés qui permettent d'obtenir des 0 et des 1 et de réaliser des opérations avec ces données : c'est ***le système binaire***.

• Pour représenter l'information dans un ordinateur, ***seuls deux symboles sont utiles***. En effet, un ordinateur fonctionne avec des circuits électroniques et ne peut distinguer que deux état :

- 1er état : le circuit électrique est ***ouvert***, ce qui correspond au ***chiffre binaire 0***.
- 2è état : le circuit électrique est ***fermé***, ce qui correspond au ***chiffre binaire 1***.

• En informatique, ***un bit*** (contraction de ***B***inary dig***IT***), est un chiffre binaire (0 ou 1). C'est la base du système binaire. ***Un octet*** est un paquet de 8 bits, par exemple (11010110).

Remarque : Il ne faut pas confondre ***bit*** et ***byte***. En effet, en Anglais byte signifie un octet, soit 8 bits !

Activité 1 : Nous avons vu dans la séquence précédente que nous pouvons représenter tous les nombres entiers naturels en binaire.
Par exemple, avec 1 bit, nous pouvons représenter deux nombres entiers naturels : $(0)_2=0$ et $(1)_2=1$.

1. Écris les nombres entiers naturels que tu peux représenter en binaire avec 2 bits. Combien y-en-a-t-il ?
2. Écris les nombres entiers naturels que tu peux représenter en binaire avec 3 bits. Combien y-en-a-t-il ?
3. Quels nombres entiers naturels peut-on représenter en binaire avec 6 bits ? Et avec 1 octet (8 bits) ?

Ce qu'il faut savoir :

• La numérisation de toutes les informations (sons, images, textes, nombres), ***repose sur le codage des nombres entiers naturels en binaire***.
Avec n bits, on peut coder $\mathbf{2^n}$ ***nombres entiers naturels : tous les entiers naturels de 0 à*** $\mathbf{2^n-1}$ **.**

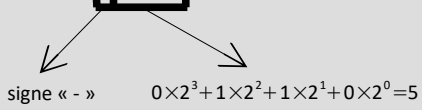
• Pour les nombres entiers et les textes qui sont ***des informations discrètes*** (il y a un nombre fini de valeurs différentes de lettres et un nombre fini d'entiers si on se fixe des bornes), il suffit d'associer chaque information à un nombre entier naturel codé en binaire.

• Pour les nombres réels, les sons et les images qui sont ***des informations continues*** (il y a un nombre infini de valeurs différentes), il faut les ***discrétiser***. C'est à dire que nous allons approximer ce nombre infini d'informations par un nombre fini d'informations que nous allons une fois encore associer à des nombres entiers naturels codés en binaire.

B. Premier codage des entiers relatifs, le binaire signé :

Ce qu'il faut savoir : Dans ***le binaire signé***, ***le bit de poids fort*** (=le premier bit le plus à gauche) est utilisé pour codé le signe (0 pour « + » et 1 pour « - ») et les bits suivants sont utilisés pour coder la valeur absolue du nombre entier relatif.

Exemple : Avec 5 bits, on a $(10110)_2$ code le nombre entier relatif -5.



Activité 2 : Dans cette activité, nous allons travailler sur 6 bits.

1. Quels sont les entiers relatifs codés par les mots binaires ci-dessous ?
a. $(111111)_2$. ***b.*** $(011111)_2$. ***c.*** $(101010)_2$. ***d.*** $(010101)_2$.

2. a. Quels sont les entiers relatifs codés par $(100000)_2$ et par $(000000)_2$?
b. Que peut-on conclure ?

3. a. Code, en binaire signé sur 6 bits, les entiers relatifs 11 et -12.
b. Pose et effectue la somme (en faisant attention aux retenues) des codes en binaire signé de 11 et -12.
c. Quel entier relatif est codé par le résultat du ***b.*** ? Que peux-tu en déduire ?

C. Second codage des entiers relatifs, le complément à 2 :

Ce qu'il faut savoir : Le binaire signé comportant des inconvénients mis en évidence dans l'activité 2, on préférera utiliser ***la méthode de codage du complément à 2***.

Avec cette méthode, si on code un entier relatif ***sur n bits***, on pourra coder $\mathbf{2^n}$ ***nombres entiers relatifs différents***, à savoir tous les nombres entiers relatifs ***compris entre*** $\mathbf{-2^{n-1}}$ ***et*** $\mathbf{2^{n-1}-1}$.

Si x est un entier relatif compris entre -2^{n-1} et $2^{n-1}-1$:
• Si *x est positif* ($0\leqslant x\leqslant 2^{n-1}-1$) : On code x , qui est un entier naturel, classiquement.

• Si *x est négatif* ($-2^{n-1}\leqslant x\leqslant -1$) : On code x par le code de $x+2^n$ qui est lui aussi un entier naturel.

Activité 3 :

1. Quels sont les nombres entiers relatifs que l'on peut coder avec la méthode du complément à 2 sur 4 bits, sur 1 octet, 4 octets ou 8 octets ?

2. a. Détermine le code avec la méthode du complément à 2, sur 1 octet, du nombre 4.

b. Dans le code obtenu, remplace les 0 par des 1 et les 1 par des 0, puis ajoute $(1)_2=(00000001)_2$ au résultat obtenu (en prenant bien garde aux retenues en base 2).

c. Détermine ensuite le code avec la méthode du complément à 2, sur 1 octet, de -4 . Que remarques-tu ?

3. Cette remarque précédente est-elle encore vérifiée pour 127 et -127 ? (réalise les calculs utiles à répondre)

4. a. Sur 4 bits, quels sont les nombres entiers relatifs qui peuvent être codés avec la méthode du complément à 2 ? Code tous ces nombres entiers relatifs avec la méthode du complément à 2.

b. Que peux-tu remarquer pour le bit de poids fort des nombres négatifs ? Et des nombres positifs ?

5. a. Code le nombre relatif -1 sur 6 bits, puis sur 1 octet.

b. Que remarques-tu ?

Ce qu'il faut savoir :

• Si on connaît le code avec la méthode du complément à 2 d'un entier relatif, pour obtenir le code de son opposé, on réécrit le code du nombre de départ en remplaçant les 1 par des 0 et les 0 par des 1, puis on ajoute $(1)_2$ au code obtenu(en prenant garde aux retenues en base 2).

Exemple : $(110100)_2$ est le code de -12 sur 6 bits.
Je remplace les 1 par des 0 et les 0 par des 1 : $(001011)_2$.
Puis j'ajoute $(1)_2=(000001)_2$: $(001011)_2+(000001)_2=(001100)_2$.
$(001100)_2$ représente bien l'entier naturel 12 (compris entre -2^5 et 2^5-1).

• Avec la méthode du complément à 2, le code d'un entier relatif positif commence toujours par 0 et celui d'un entier relatif négatif commence toujours par 1.

• Avec la méthode du complément à 2, le code de l'entier -1 n'est constitué que de 1.
Exemple : le code de -1 est $(1111)_2$ sur 4 bits, ou $(111111)_2$ sur 6 bits ou encore $(11111111)_2$ sur un octet.

Activité 4 : Quels nombres entiers relatifs sont représentés, par la méthode du complément à 2 par
1. $(00000000)_2$ ***2.*** $(10000000)_2$ ***3.*** $(01111111)_2$ ***4.*** $(10000001)_2$.

aide : Commence par déterminer si cela représente un nombre positif ou négatif.
Puis convertis ces codes binaires en des entiers naturels. Pour finir conclus.

Activité 5 :

1. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2**i
    print(a)
print(time()-t)
```

a. Teste et explique ce que fait ce programme.

b. Quel est le dernier résultat affiché ?

2. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2222**i
    print(a)
print(time()-t)
```

a. Teste (sois patient!) et explique ce que fait ce programme.

b. Quel est le dernier résultat affiché ?

3. Comment peux-tu expliquer la différence de résultats pour les questions ***c.*** des questions ***1.*** et ***2.*** ?

Ce qu'il faut savoir :

• Avec certains langages, le nombre d'octets avec lequel on travaille pour représenter les entiers relatifs peut être précisé (***1, 2, 4 ou 8 octets***, c'est à dire ***8, 16, 32 ou 64 bits***).

• Avec Python, il est inutile de se soucier de ce nombre, car c'est le logiciel qui décide seul combien d'octets sont nécessaires pour coder les entiers relatifs. Avec Python, le type des nombres entiers se nomme ***int*** (pour integer = entier en Français).

• Cependant, si un entier dépasse une certaine taille, le logiciel Python le découpe en plusieurs parties pour traiter les opérations. Ceci a pour conséquence de ***ralentir de façon importante les calculs*** et de ***nécessiter davantage d'espace mémoire***.

D. Exercices d'application :

Exercice 1 : Donne, avec la méthode du complément à 2, code les entiers ci-dessous sur 1 octet.
a. 100. ***b.*** 75. ***c.*** -50. ***d***. -128. ***e.*** -1. ***f.*** 128. ***g.*** -89.

Exercice 2 : Dans chaque cas, donne le code des entiers a et b sur 1 octet avec la méthode du complément à 2. Pose et effectue la somme des 2 codes obtenus, puis vérifie que c'est bien le code de l'entier $a+b$.
1. $a=35$ et $b=65$. ***2.*** $a=-12$ et $b=45$. ***3.*** $a=-84$ et $b=29$.

Exercice 3 : Détermine les nombres entiers relatifs qui sont représentés, avec la méthode du complément à 2, par les codes suivants. ***a.*** $(11001011)_2$. ***b.*** $(11010100)_2$. ***c.*** $(10000010)_2$. ***d.*** $(10101010)_2$.

Exercice 4 : VRAI ou FAUX ? (Justifie tes réponses)
1. Les premiers ordinateurs sont apparus au début du XIXè siècle.

2. 1 000 s'écrit aussi $(ABC)_{16}$ en hexadécimal.

3. $(1001)_2 \times (111)_2 = (111111)_2$.

4. L'entier relatif -2 est codé sur 5 bits $(10010)_2$ avec la méthode du complément à 2.

5. Tout ce qui est numérisé dans un ordinateur est identique aux informations réelles.

Exercice 5 : Écrire un programme sur Python qui demande à l'utilisateur de saisir un entier relatif r codable sur 1 octet avec la méthode du complément à 2, qui détermine le code et le renvoie.
Aide : Revoir l'exercice 6 de la séquence précédente (=Écriture d'un entier naturel en base b⩾2).

Séquence : *Représentation binaire approximative d'un réels : les flottants.*	**Numérique et Sciences Informatiques**	***Thème :*** *Représentation des données (types de base)*

I – Représentation de la partie à droite de la virgule d'un nombre à virgule

En notation décimale, les chiffres à gauche de la virgule représentent des entiers, des dizaines, des centaines, des milliers, … etc, et ceux à droite de la virgule représentent des dixièmes, centièmes, millièmes, etc.

Ainsi : $3,8125_{(10)} = 3\times10^{0} + 8\times10^{-1} + 1\times10^{-2} + 2\times10^{-3} + 5\times10^{-4}$

De la même façon, pour un nombre à virgule en base 2, on utilise les puissances négatives de 2.
Observe cet exemple :

$$\begin{aligned} 11,1101_{(2)} &= 1\times2^{1} + 1\times2^{0} + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} \\ &= 2 + 1 + \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} \\ &= 2 + 1 + 0,5 + 0,25 + 0 + 0,0625 \\ &= 3,8125_{(10)} \end{aligned}$$

De la même manière, convertis le nombre suivant en base décimale : $(101,010101)_{2,}$

II – Conversion en binaire de la partie à droite de la virgule

1. Observez le schéma ci-contre :

Que permet-il de faire ?

$0,375\times10 = \boxed{3}\ ,75$

$0,75\times10 = \boxed{7}\ ,5$

$0,5\times10 = \boxed{5}\ ,0$

2. Faites la même chose avec 0,42 et 0,8769. Comment sait-ton que l'on a terminé ?

3. Pour obtenir en binaire la partie à droite de la virgule, on procède de la même façon, mais en multipliant par 2 à chaque étape car on travaille alors en base 2.

a. En adaptant le procédé ci-dessus, déterminer la partie à droite de la virgule en base 2 de 0,375.

b. Vérifiez que le résultat obtenu en base 2, lorsqu'il est converti en base 10 redonne bien 0,375.

c. Faire la même chose avec 0,625 puis vérifier le résultat obtenu.

d. Faire de même avec 0,1. Que remarquez-vous ?

$0,375\times2 = \boxed{0}\ ,75$

$0,75\times2 = \boxed{1}\ ,5$

$0,5\times2 = \boxed{1}\ ,0$

4. a. On souhaite réaliser un algorithme en langage naturel qui effectue cette conversion.
En observant les conversions faites précédemment, écrire en langage naturel les instructions qui sont répétées.

b. Peut-on prévoir à l'avance combien de fois elles vont être répétées ?
Si non, comment peut-on savoir à quel moment il faudra arrêter de les répéter ?

c. Rédiger un algorithme en langage naturel pour effectuer la conversion.

5. Voilà un programme Python qui met en œuvre cet algorithme.
Les bits sont stockés dans une chaîne de caractères.

```
from decimal import *

n = Decimal(input("Entrer la partie décimale : "))
en_binaire = ''

i = 0
while (n != 0) and (i < 64):
    n = n*2
    if n >= 1:
        en_binaire = en_binaire + '1'
        n = n - 1
    else:
        en_binaire = en_binaire + '0'
    i = i+1

print(en_binaire)
```

a. Testez le programme avec les valeurs du début de l'activité.
b. Modifiez le programme pour qu'il affiche les calculs intermédiaires comme ci-dessous :

$$0,375\times 2 = 0\ ,\ 75$$
$$0,75\ \times 2 = 1\ ,\ 5\ .$$
$$0,5\ \times 2 = 1\ ,\ 0$$

6. Reprenez le programme Python précédent et remplacez **Decimal** par **float.**
Reprenez les tests précédents. Que remarquez-vous ?

---

5. Voilà un programme Python qui met en œuvre cet algorithme.
Les bits sont stockés dans une chaîne de caractères.

```
from decimal import *

n = Decimal(input("Entrer la partie décimale : "))
en_binaire = ''

i = 0
while (n != 0) and (i < 64):
    n = n*2
    if n >= 1:
        en_binaire = en_binaire + '1'
        n = n - 1
    else:
        en_binaire = en_binaire + '0'
    i = i+1

print(en_binaire)
```

a. Testez le programme avec les valeurs du début de l'activité.
b. Modifiez le programme pour qu'il affiche les calculs intermédiaires comme ci-dessous :

$$0,375\times 2 = 0\ ,\ 75$$
$$0,75\ \times 2 = 1\ ,\ 5 \qquad \text{donc } 0,375=0,011.$$
$$0,5\ \times 2 = 1\ ,\ 0$$

6. Reprenez le programme Python précédent et remplacez **Decimal** par **float.**
Reprenez les tests précédents. Que remarquez-vous ?

NOM et Prénom :

L'accident de Dhahran

Dans le tableau ci-dessous, placez le développement binaire de 0,1 en prenant soin de noter sa partie entière dans le bit de poids fort et indiquez soigneusement la place de la virgule.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	...
																																		...

La batterie Patriot dispose d'un registre de 24 bits. Indiquez dans le tableau ci-dessus quelle est la partie du nombre qui est effectivement stockée en mémoire.

Après avoir analysé le document du groupe d'experts du GAO, écrivez dans le tableau ci-dessous l'erreur commise dans le stockage de l'unité de temps.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	...

Exprimez cette erreur en base 10 :

Calculez l'erreur cumulée au bout de 1h de fonctionnement :

Dans le tableau donné en annexe par le GAO (Appendix II), contrôlez les résultats affichés dans les lignes correspondant à 8h et à 100h de fonctionnement.

Sachant qu'un Scud a une vitesse d'environ 3750,26 mph, quelle est la distance qu'il parcourt durant le laps de temps correspondant à l'erreur commise pour une centaine d'heures de fonctionnement (1 mile ≈ 1,6 km) :

A. Codage des nombres réels :

Ce qu'il faut savoir : Le codage ***sur 64 bits*** (8 octets) en binaire des nombres réels pour les représenter en machine est défini par ***la norme IEEE 754***.

Pour représenter un nombre à virgule en binaire, on va utiliser une représentation similaire à la notation scientifique en base 10, mais ici en base 2 : il s'agit de ***la représentation en virgule flottante***.

Cette représentation est $Sm\times 2^n$ où ***S*** est ***le signe*** (+ ou -), ***m*** est ***la mantisse*** ($1\leqslant m<2$)
et ***n l'exposant*** qui est un nombre entier relatif.

On codera en binaire ce nombre ainsi représenté sur 64 bits ainsi :

64è bit	63è bit	61è bit	….....	53è bit	52è bit	51è bit	50è bit	49è bit	….....	3è bit	2è bit	1er bit
signe	**exposant décalé**				**m – 1 = partie décimale de la matisse**							

1 bit *(0 code + et 1 code -)*

11 bits *codent un nombre entier naturel* ***d*** = ***exposant décalé*** = *n+1 023* *(0 ≤ d ≤ 2 047).*

Si d=0 ou 2 047, cela est réservé à des situations exceptionnelles telles que ***l'infini*** *ou* ***« not a number »****.*

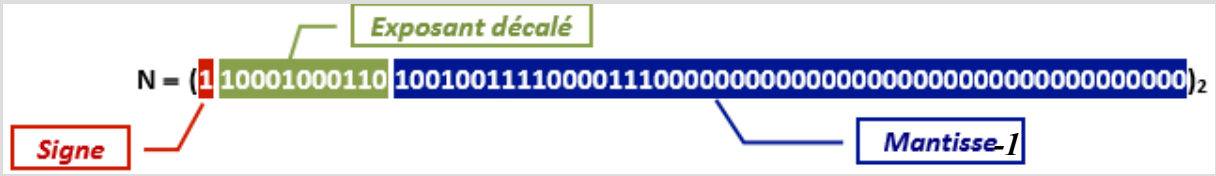
Si 1≤ d ≤ 2 046, ***on soustrait 1 023*** *pour obtenir l'exposant n qui est un entier relatif.*

52 bits *codent m-1 avec 1≤ m<2.*

La partie entière de m est toujours égale à 1, *il est inutile de coder ce chiffre et de perdre 1 bit.*

m-1, la partie décimale de m est une somme de puissances négatives de 2.

Exemple :



Signe = 1 : le nombre ***est négatif***.

Exposant décalé d = $(10001000110)_2=2^{10}+2^6+2^2+2^1=1024+64+4+2=$ 1 094.
Donc ***l'exposant n*** = 1 094-1 023=***71***.

Mantisse = $1,10010011110000111000\ldots)_2=1+2^{-1}+2^{-4}+2^{-7}+2^{-8}+2^{-9}+2^{-10}+2^{-15}+2^{-16}+2^{-17}\approx 1,5772$.

La partie entière de la mantisse est toujours 1.

DONC ce code représente le réel : $-1,5772\times 2^{71}\approx -3,724\times 10^{21}$.

Cas particulier : On convient que le réel ***0 est codé par (0 00000000000 0000...00)$_2$***.

Activité 1 :

1. Trouve le nombre réel codé au format IEEE 754 par : 0 00100000011 11010011100101011000000000000000000000000000000000000.

2. Inversement, on veut maintenant coder le nombre réel -118,375 au format IEEE 754.

a. Divise 118,375 par 2 autant de fois qu'il le faut afin d'obtenir un résultat supérieur ou égal à 1 et strictement inférieur à 2. Déduis-en l'écriture de -118,375 sous la forme $Sm\times 2^n$ où S est le signe, m la matisse et n l'exposant.

b. A quel bit correspond le signe S ?

c. Calcule l'exposant décalé

d. A quel code binaire correspond cet exposant décalé ?

e. Calcule la partie décimale de la mantisse.

f. Multiplie la partie décimale de la mantisse par 2. Si le résultat est strictement inférieur à 1, le premier bit du code de la partie décimale de la mantisse est 0. En revanche, si le résultat est supérieur à 1, le premier bit est 1 et tu dois soustraire 1 au résultat.
Puis recommence ceci : multiplie le dernier résultat par 2. Si tu obtiens un résultat strictement inférieur à 1, ajoute le bit 0 à droite du code, sinon ajoute 1 à droite du code et soustrais 1 au résultat. Etc...
Tu devras t'arrêter lorsque tu obtiendras exactement 1. A ce moment, tu ajouteras le bit 1 à droite du code, puis tu compléteras par des 0 à droite du code pour qu'il y ait bien 52 bits.
Donne ainsi le code de la partie décimale de la mantisse sur 52 bits.

g. Déduis-en le code au format IEEE 754 de -118,375.

3. En reproduisant la méthode de la question 2., détermine le code au format IEEE 754 de ces nombres réels.

a. 75,281 25. ***b.*** -0,687 5.

Activité 2 :

1. a. Comment code-t-on en binaire le nombre entier -7 (sur 8 bits) ? Et le nombre réel -7,0 (sur 64 bits) ?

b. Pourquoi le même nombre n'est-il pas codé de la même façon ?
Quelle précaution faudra-t-il prendre lorsqu'on écrit un programme (notamment avec Python) ?

2. a. Applique la méthode de l'activité 1 pour tenter de convertir le nombre réel 0,2 au format IEEE 754.
Que se passe-t-il ?

b. Quand les 52 bits de la mantisse ne suffisent pas, si le 53è bit est 1, on arrondit le bit de la mantisse qui est le plus à droite par excès. Mais si le 53è bit est 0, on arrondit le bit de la mantisse qui est le plus à droite par défaut. Quel sera finalement le code de 0,2 au format IEEE 754 ?

c. Déduis-en le code au format IEEE 754 du réel 0,1. Puis applique la méthode de l'activité 1 pour trouver celui du réel 0,3.

d. Ouvre Python et tape dans la console : ***>>>(0.1+0.2)-0.3*** puis ***« Entrée »***.
Pourquoi le résultat est-il surprenant ? Comment l'expliquer ?

Ce qu'il faut savoir :

• Il est impératif de bien distinguer le type ***« integer »*** du type ***« float »*** dans un programme. En particulier l'entier 7 et le flottant 7.0 ne se codent pas de la même manière en binaire dans une machine.

• Sur machine les calculs avec les nombres entiers sont exacts, mais les calculs avec les nombres flottants peuvent présenter d'infimes erreurs d'arrondi.

B. Exercices d'application :

Exercice 1 : Détermine à quels nombres réels correspondent ces codes au format IEEE 754.

a. $(1\ 01111111110\ 1000...0)_2$. ***b.*** $(0\ 10000001000\ 0001100101011100...0)_2$. ***c.*** $(0\ 00000000000\ 000...0)_2$.

Exercice 2 : Convertis les nombres suivants au format IEEE 754.
Pour chaque cas, précise si les codes représentent exactement ou approximativement les nombres réels.

a. 3,625. ***b.*** -4,5. ***c.*** -50. ***d.*** $-\frac{11}{15}=-0,7333...$ ***e.*** 1.

A. Écriture d'un entier naturel dans une base b (b est un entier supérieur ou égal à 2) :

Activité 1 :

1. Calcule le nombre entier $3\times4^3+1\times4^2+2\times4^1+0\times4^0$ = $\mathbf{3\times64+1\times16+0\times1=172+16=188}$

(**Rappel :** $x^0=1$ pour tout nombre x non nul).

Ce nombre est écrit comme la somme ***de puissances de 4*** multipliées par des entiers ***compris entre 0 et 3***.

On dira que ce nombre s'écrit $(3120)_4$ ***en base 4***. $\mathbf{188=(3120)_4}$

2. A quel nombre entier est égal : ***a.*** $(3311)_4$. ***b.*** $(200)_3$. ***c.*** $(543210)_6$. ***d.*** $(708)_9$. ***e.*** $(8439)_{10}$.

3125 ***18*** ***10 885 032 150*** ***575*** ***8 439***

3. a. Calcule les puissances de 2 ($2^0=$ … ; $2^1=$ … ; $2^2=$ … ; ..... ; $2^7=$ …).

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

b. Déduis en l'écriture du nombre entier 203 en base 2.

$\mathbf{203=128+64+8+2+1=1\times2^7+1\times2^6+0\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0=(11001011)_2}$

c. De la même manière écris 203 en base 5, puis en base 8.

$\mathbf{203=(1303)_5=(313)_8}$

4. a. Quelle est l'écriture de $2\times3^2+2$ en base 3 ? $\mathbf{2\times3^2+2=2\times3^2+0\times3^1+2\times3^0=(202)_3}$

b. Quelle est l'écriture de $4\times3^2+2\times3+5$ en base 3 ?

$\mathbf{4\times3^2+2\times3+5=1\times3^3+1\times3^2+2\times3^1+5\times3^0=(1125)_3}$

Ce qu'il faut savoir : Si N est un entier naturel tel que $N=a_k\times b^k+a_{k-1}\times b^{k-1}+ \ldots\ldots +a_1\times b^1+a_0\times b^0$, b étant un nombre entier naturel supérieur ou égal à 2 et tous les nombres a_i étant ***des entiers compris entre 0 et b-1*** , alors on dit que $(a_k a_{k-1}\ldots a_1 a_0)_b$ est ***l'écriture en base b*** du nombre entier naturel N .

On notera $N=(a_k a_{k-1}\ldots a_1 a_0)_b$.

Remarque : On utilisera la notation $(\ldots\ldots\ldots\ldots)_b$ pour préciser dans quelle base est l'écriture.
On conviendra que si l'on n'écrit pas cette notation, l'écriture est en base « naturelle », c'est à dire la base 10.

B. Cas particulier de la base 2 :

Activité 2 :

On veut écrire le nombre entier naturel 73 en base 2.

Pour cela on va chercher à l'écrire sous la forme $b_n\times 2^n+\ldots+b_1\times 2^1+b_0\times 2^0$ car ainsi on obtient en base 2 le nombre $\left(b_n\ldots b_1 b_0\right)_2$.

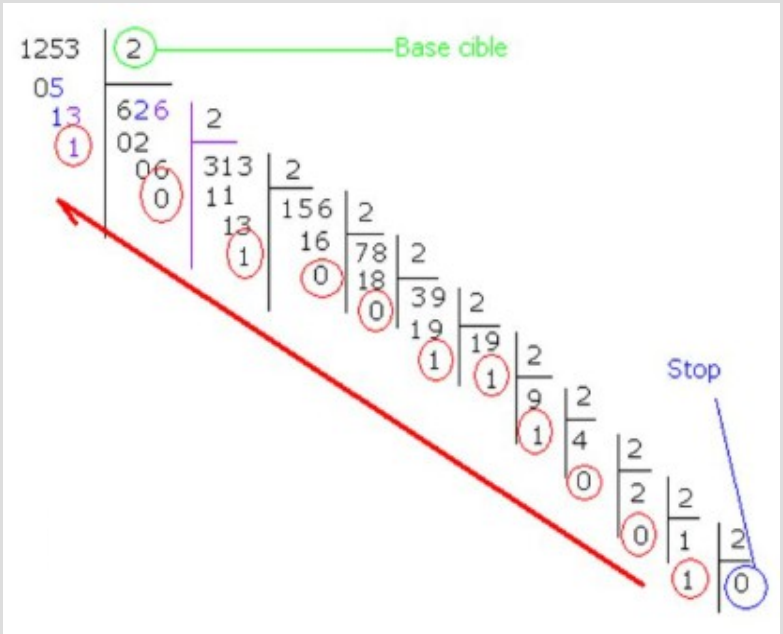
1. Écris la division euclidienne de 73 par 2 et l'égalité euclidienne : $dividende=quotient\times diviseur+reste$.
Écris ensuite la division euclidienne du quotient obtenu par 2 et l'égalité euclidienne correspondante que tu inséreras dans l'égalité euclidienne précédente. Pour finir poursuis ainsi jusqu'à obtenir un quotient nul.

$\mathbf{73=2\times 36+1}$.
Or $\mathbf{36=2\times 18+0}$ **, donc** $\mathbf{73=2\times(2\times 18+0)+1=2^2\times 18+1}$.
Or $\mathbf{18=2\times 9+0}$ **, donc** $\mathbf{73=2^2\times(2\times 9+0)+1=2^3\times 9+1}$.
Or $\mathbf{9=2\times 4+1}$ **, donc** $\mathbf{73=2^3\times(2\times 4+1)+1=2^4\times 4+2^3\times 1+1}$.
Or $\mathbf{4=2\times 2+0}$ **, donc** $\mathbf{73=2^4\times(2\times 2+0)+2^3\times 1+1}$
$\mathbf{=1\times 2^6+0\times 2^5+0\times 2^4+1\times 2^3+0\times 2^2+0\times 2^1+1\times 2^0}$.

2. Déduis-en l'écriture de 73 sous la forme $b_n\times 2^n+\ldots+b_1\times 2^1+b_0\times 2^0$, puis son écriture en base 2.
$\mathbf{73=\left(1001001\right)_2}$.

Ce qu'il faut savoir : Pour convertir un entier naturel (qui est en base 10) en base b avec $b\geqslant 2$, on divise cet entier naturel par b jusqu'à obtenir un quotient égal à 0.
Le résultat cherché est ***la juxtaposition des restes du dernier au premier***.

Exemple :



1 253 = $\left(10011100101\right)_2$

Activité 3 : En utilisant la méthode ci-dessus, effectue les conversions ci-dessous.
1. Écris 47 , puis 53 et 245 en base 2. $\mathbf{47=\left(101111\right)_2}$, $\mathbf{53=\left(110101\right)_2}$ **et** $\mathbf{245=\left(11110101\right)_2}$.
2. Écris 67, puis 231 et 2 578 en base 7. $\mathbf{67=\left(124\right)_7}$, $\mathbf{231=\left(450\right)_7}$ **et** $\mathbf{2\ 578=\left(10342\right)_7}$.

C. Cas particulier de la base 16 :

Activité 4 :

1. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 2 ? ***2 chiffres = 0 et 1.***

2. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 5 ? ***5 chiffres = 0, 1, 2, 3 et 4.***

3. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 10 ? ***10 chiffres = 0, 1, ..., 8 et 9.***

4. Combien de chiffres différents utilise-t-on pour écrire un nombre en base 12 ? Quel est le problème ?
12 chiffres. Problème : il n'y a que 10 chiffres !

Ce qu'il faut savoir : Pour convertir un nombre entier naturel N en base 16, il faut l'écrire sous la forme $N = a_k \times 16^k + a_{k-1} \times 16^{k-1} + \ldots\ldots + a_1 \times 16^1 + a_0 \times 16^0$, les a_i étant ***des entiers compris entre 0 et 15***.
On pose alors $c_i = a_i$ si $0 \leqslant a_i \leqslant 9$ et $c_i =$ A, B, C, D ou E si $a_i =$ 10, 11, 12, 13, 14 ou 15.
Et on dit que $\left(c_k c_{k-1} \ldots c_1 c_0\right)_{16}$ est ***l'écriture en base 16*** ou ***en hexadécimal*** du nombre entier naturel N .

On notera $N = \left(c_k c_{k-1} \ldots c_1 c_0\right)_{16}$ **.**

Exemple : $\left(A5\mathrm{E}\right)_{16} = 10 \times 16^2 + 5 \times 16^1 + 14 \times 16^0 = 2\,654.$

Activité 5 :

1. A quel nombre entier est égal : ***a.*** $\left(7\mathrm{DD}\right)_{16}$. ***b.*** $\left(2\mathrm{A}\right)_{16}$. ***c.*** $\left(4F2\mathrm{C}\right)_{16}$.
2 013 ***42*** ***20 268***

2. Écris chaque nombre entier en base 16 : ***a.*** 62. ***b.*** 1455. ***c.*** 8675.
$\left(\mathbf{3E}\right)_{16}$ $\left(\mathbf{5AF}\right)_{16}$ $\left(\mathbf{21}E3\right)_{16}$

D. Exercices d'application :

Exercice 1 :

1. A quel entier est égal : ***a.*** $\left(101010101\right)_2$. ***b.*** $\left(111000\right)_2$. ***c.*** $\left(00110011\right)_2$. ***d.*** $\left(101000001\right)_2$.
341 ***56*** ***51*** ***321***

2. Convertis en binaire : ***a.*** 458. ***b.*** 133. ***c.*** 47. ***d.*** 1 024. ***e.*** 65.
$\left(\mathbf{111001010}\right)_2$ $\left(\mathbf{10000101}\right)_2$ $\left(\mathbf{101111}\right)_2$ $\left(\mathbf{10000000000}\right)_2$ $\left(\mathbf{1000001}\right)_2$

Exercice 2 :

1. A quel entier est égal : ***a.*** $\left(A320\right)_{16}$. ***b.*** $\left(FABE51\right)_{16}$.
41 760 ***16 432 721***

2. Convertis en hexadécimal : ***a.*** 2 020. ***b.*** 1 234. ***c.*** 56 026. ***d.*** 64 218.
$\left(\mathbf{7}E4\right)_{16}$ $\left(\mathbf{4}D2\right)_{16}$ $\left(\mathbf{DADA}\right)_{16}$ $\left(\mathbf{FADA}\right)_{16}$

Exercice 3 : Convertis en binaire les nombres écrits en hexadécimal suivants (On commencera par les convertir en base 10). ***a.*** $\left(101010\right)_{16}$. ***b.*** $\left(59A75\right)_{16}$
1 052 688 = $\left(\mathbf{100000001000000010000}\right)_2$ ***367 221 =*** $\left(\mathbf{1011001101001110101}\right)_2$

Exercice 4 :

1. On veut convertir en hexadécimal $(1001101)_2$. Pour cela, il suffit de compléter le tableau ci-dessous.

Écriture binaire « par paquets de 4 »	**0**100	1101
Écriture décimal « des paquets de 4 »	4	13
Écriture hexadécimale	4	D

On ajoute des « 0 » pour avoir un paquet de 4 chiffres.

Complète ce tableau et convertis $(1001101)_2$ en hexadécimal. $(\mathbf{1001101})_2=(\mathbf{4D})_{16}$.

2. En utilisant la même méthode, convertis en hexadécimal les nombres ci-dessous.

a. $(10000000011001)_2$. $(\mathbf{2019})_{16}$ ***b.*** $(10001000010001)_2$. $(\mathbf{2211})_{16}$ ***c.*** $(100110000111)_2$. $(\mathbf{987})_{16}$ ***d.*** $(101110101100)_2$. $(\mathbf{BAC})_{16}$

Exercice 5 :

1. Pose et effectue chacune des opérations ci-dessous en base 2. Pour vérifier, convertis chaque terme des opérations en base 10, refais les opérations en base 10 et compare le résultat avec celui obtenu en base 2.
Important : On utilisera le fait que en base 2 on a $(1)_2+(1)_2=(10)_2$ pour bien gérer les retenues.

a. $(100011)_2+(110100)_2$. $(\mathbf{1010111})_2$ ***b.*** $(100101)_2+(110000)_2$. $(\mathbf{1010101})_2$ ***c.*** $(100011)_2+(111)_2$. $(\mathbf{101010})_2$ ***d.*** $(10000000)_2-(100)_2$. $(\mathbf{1111100})_2$

2. Pose et effectue : ***a.*** $(98)_{16}+(B9)_{16}$. $(\mathbf{151})_{16}$ ***b.*** $(D23)_{16}+(46\mathrm{A})_{16}$. $(\mathbf{118D})_{16}$ ***c.*** $(150F6)_{16}-(7E3\mathrm{A})_{16}$. $(\mathbf{D2BC})_{16}$

Exercice 6 : Considérons le programme écris ci-dessous en Python.

```
n = int(input("Entrez un nombre entier"))
b = ""
while n!= 0:
    q = int(n/2)
    b = str(n-q*2)+b
    n = q
print(b)
```

1. Recopie et complète ce tableau jusqu'à ce que le programme se termine lorsqu'on a choisi ***n=71*** :

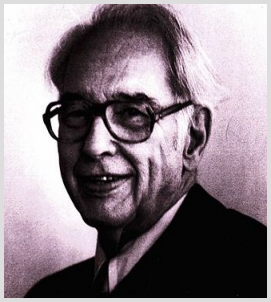
q		35	17	8	4	2	1	0
n-q*2		1	1	1	0	0	0	1
b	""	" 1 "	"11"	"111"	"0111"	"00111"	"000111"	"1000111"
n	71	35	17	8	4	2	1	0
n!=0	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	VRAI	FAUX

2. Explique le rôle de ce programme.
Ce programme convertit un entier naturel en base 2.

3. Tape ce programme sur Python et teste le sur certains résultats de cette fiche d'exercices.

A. Le principe de la numérisation de l'information :

Ce qu'il faut savoir :



John Atanasoff

• A la fin des années 1930, ***John Atanasoff*** (physicien et mathématicien américain d'origine bulgare) a mis au point avec un de ses étudiants, ***Clifford Berry***, l'ABC (Atanasoff Berry Computer), un ordinateur de 300 kg, capable d'effectuer une opération toutes les 15 secondes. L'ABC est considéré comme ***le premier ordinateur***. Pour créer cet appareil, Atanasoff a imaginé un système électronique composé d'interrupteurs ouverts ou fermés qui permettent d'obtenir des 0 et des 1 et de réaliser des opérations avec ces données : c'est ***le système binaire***.

• Pour représenter l'information dans un ordinateur, ***seuls deux symboles sont utiles***. En effet, un ordinateur fonctionne avec des circuits électroniques et ne peut distinguer que deux état :
- 1er état : le circuit électrique est ***ouvert***, ce qui correspond au ***chiffre binaire 0***.
- 2è état : le circuit électrique est ***fermé***, ce qui correspond au ***chiffre binaire 1***.

• En informatique, ***un bit*** (contraction de ***B***inary dig***IT***), est un chiffre binaire (0 ou 1). C'est la base du système binaire. ***Un octet*** est un paquet de 8 bits, par exemple (11010110).

Remarque : Il ne faut pas confondre ***bit*** et ***byte***. En effet, en Anglais byte signifie un octet, soit 8 bits !

Activité 1 : Nous avons vu dans la séquence précédente que nous pouvons représenter tous les nombres entiers naturels en binaire.
Par exemple, avec 1 bit, nous pouvons représenter deux nombres entiers naturels : $(0)_2=0$ et $(1)_2=1$.

1. Écris les nombres entiers naturels que tu peux représenter en binaire avec 2 bits. Combien y-en-a-t-il ?
Avec 2 bits on a $(00)_2$ ***= 0 ,*** $(01)_2$ ***= 1 ,*** $(10)_2$ ***= 2 et*** $(11)_2$ ***= 3. Donc les*** $4=2^2$ ***premiers entiers naturels.***

2. Écris les nombres entiers naturels que tu peux représenter en binaire avec 3 bits. Combien y-en-a-t-il ?
Avec 3 bits on a $(000)_2$ ***= 0 ,*** $(001)_2$ ***= 1 ,*** $(011)_2$ ***= 3 ,*** $(100)_2$ ***= 4 ,*** $(101)_2$ ***= 5 ,*** $(110)_2$ ***= 6 et*** $(111)_2$ ***= 7.***
Donc les $8=2^3$ ***premiers entiers naturels.***

3. Quels nombres entiers naturels peut-on représenter en binaire avec 6 bits ? Et avec 1 octet (8 bits) ?
Avec 6 ou 8 bits on peut coder les $2^6=64$ ***ou*** $2^8=256$ ***premiers entiers naturels.***

Ce qu'il faut savoir :

• La numérisation de toutes les informations (sons, images, textes, nombres), ***repose sur le codage des nombres entiers naturels en binaire***.
Avec n bits, on peut coder 2^n ***nombres entiers naturels : tous les entiers naturels de 0 à*** 2^n-1 .

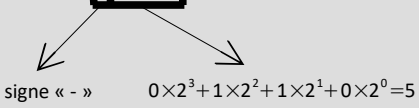
• Pour les nombres entiers et les textes qui sont ***des informations discrètes*** (il y a un nombre fini de valeurs différentes de lettres et un nombre fini d'entiers si on se fixe des bornes), il suffit d'associer chaque information à un nombre entier naturel codé en binaire.

• Pour les nombres réels, les sons et les images qui sont ***des informations continues*** (il y a un nombre infini de valeurs différentes), il faut les ***discrétiser***. C'est à dire que nous allons approximer ce nombre infini d'informations par un nombre fini d'informations que nous allons une fois encore associer à des nombres entiers naturels codés en binaire.

B. Premier codage des entiers relatifs, le binaire signé :

Ce qu'il faut savoir : Dans ***le binaire signé***, ***le bit de poids fort*** (=le premier bit le plus à gauche) est utilisé pour codé le signe (0 pour « + » et 1 pour « - ») et les bits suivants sont utilisés pour coder la valeur absolue du nombre entier relatif.

Exemple : Avec 4 bits, on a $(10110)_2$ code le nombre entier relatif -5.



Activité 2 : Dans cette activité, nous allons travailler sur 6 bits.

1. Quels sont les entiers relatifs codés par les mots binaires ci-dessous ?

a. $(111111)_2$. ***b.*** $(011111)_2$. ***c.*** $(101010)_2$. ***d.*** $(010101)_2$.

-31 ***31*** ***-9*** ***21***

2. a. Quels sont les entiers relatifs codés par $(100000)_2$ et par $(000000)_2$? ***Les deux codes représentent 0.***

b. Que peut-on conclure ? ***C'est problématique.***

3. a. Code, en binaire signé sur 6 bits, les entiers relatifs 11 et -12. $(\mathbf{001011})_2$ ***et*** $(\mathbf{101100})_2$ **.**

b. Pose et effectue la somme (en faisant attention aux retenues) des codes en binaire signé de 11 et -12.

$$\begin{array}{r} 0\,0\,1\,0\,1\,1 \\ +\ 1\,0\,1\,1\,0\,0 \\ \hline 1\,1\,0\,1\,1\,1 \end{array}$$

$(\mathbf{001011})_2$ **+** $(\mathbf{101100})_2$ **=** $(\mathbf{110111})_2$ **.**

c. Quel entier relatif est codé par le résultat du ***b.*** ? Que peux-tu en déduire ?

$(\mathbf{110111})_2$ ***représente le nombre -23. Comme*** $\mathbf{11}+(-\mathbf{12})=-\mathbf{1}$ ***et pas -23, c'est problématique.***

C. Second codage des entiers relatifs, le complément à 2 :

Ce qu'il faut savoir : Le binaire signé comportant des inconvénients mis en évidence dans l'activité 2, on préférera utiliser ***la méthode de codage du complément à 2***.

Avec cette méthode, si on code un entier relatif ***sur n bits***, on pourra coder $\mathbf{2}^n$ ***nombres entiers relatifs différents***, à savoir tous les nombres entiers relatifs ***compris entre*** $-\mathbf{2}^{n-1}$ ***et*** $\mathbf{2}^{n-1}-\mathbf{1}$.

Si x est un entier relatif compris entre -2^{n-1} et $2^{n-1}-1$:

• Si *x est positif* ($0\leqslant x\leqslant 2^{n-1}-1$) : On code x , qui est un entier naturel, classiquement.

• Si *x est négatif* ($-2^{n-1}\leqslant x\leqslant -1$) : On code x par le code de $x+2^n$ qui est lui aussi un entier naturel.

Activité 3 :

1. Quels sont les nombres entiers relatifs que l'on peut coder avec la méthode du complément à 2 sur 4 bits, sur 1 octet, 4 octets ou 8 octets ?

Sur 1 octet = 8 bits, on peut coder $2^8=256$ ***nombres : les nombres de*** $-2^7=-128$ ***à*** $2^7-1=127$ **.**
Sur 4 octets = 32 bits, on peut coder $2^{32}=$ ***4 294 967 296 nombres :***
les nombres de $-2^{31}=$ ***-2 147 483 648 à*** $2^{31}-1=$ ***2 147 483 647.***
Sur 8 octets = 64 bits, on peut coder $2^{64}=$ ***18 446 744 073 709 551 616 nombres :***
les nombres de $-2^{63}=$ ***-9 223 372 036 854 775 808 à*** $2^{31}-1=$ ***9 223 372 036 854 775 807.***

2. a. Détermine le code avec la méthode du complément à 2, sur 1 octet, du nombre 4. $(\mathbf{00000100})_2$

b. Dans le code obtenu, remplace les 0 par des 1 et les 1 par des 0, puis ajoute $(1)_2=(00000001)_2$ au résultat obtenu (en prenant bien garde aux retenues en base 2).
Dans $(\mathbf{00000100})_2$ **,** ***on remplace les 0 par des 1 et les 1 par des 0 :*** $(\mathbf{11111011})_2$ **.**
On ajoute $(\mathbf{00000001})_2$ ***et on obtient*** $(\mathbf{11111100})_2$ **.**

c. Détermine ensuite le code avec la méthode du complément à 2, sur 1 octet, de -4 . Que remarques-tu ?
Avec la méthode du complément à 2, sur 1 octet=8bits, -4 se code par le nombre entier naturel $-4+2^8=252$ **.** ***On a donc*** $(\mathbf{11111100})_0$ **.**
On remarque que le procédé de la question b. permet de retrouver le code de -4, l'opposé de 4.

3. Cette remarque précédente est-elle encore vérifiée pour 127 et -127 ? (réalise les calculs utiles à répondre)
On admettra que cette remarque est toujours vérifiée !
Avec la méthode du complément à 2, sur 1 octet=8bits, 127 se code $(\mathbf{01111111})_2$ **.**
Dans $(\mathbf{01111111})_2$ **,** ***on remplace les 0 par des 1 et les 1 par des 0 :*** $(\mathbf{0101})_2$ **.**
On ajoute $(\mathbf{00000001})_2$ ***et on obtient*** $(\mathbf{10000001})_2$ **.**
Avec la méthode du complément à 2, sur 1 octet=8bits, -127 se code par le nombre entier naturel $-127+2^8=129$ **.** ***On a donc*** $(\mathbf{10000001})_2$ **.**
Une fois encore que le procédé de la question b. permet de retrouver le code de -127, l'opposé de 127.

4. a. Sur 4 bits, quels sont les nombres entiers relatifs qui peuvent être codés avec la méthode du complément à 2 ? Code tous ces nombres entiers relatifs avec la méthode du complément à 2.
Sur 4 bits, on peut coder $2^4=16$ ***nombres : les nombres de*** $-2^3=-8$ ***à*** $2^3-1=7$ **.**
-8 : $(\mathbf{1000})_2$ **,** ***-7 :*** $(\mathbf{1001})_2$ **,** ***-6 :*** $(\mathbf{1010})_2$ **,** ***-5 :*** $(\mathbf{1011})_2$ **,** ***-4 :*** $(\mathbf{1100})_2$ **,** ***-3 :*** $(\mathbf{1101})_2$ **,** ***-2 :*** $(\mathbf{1110})_2$ **,**
-1 : $(\mathbf{1111})_2$ **,** ***0 :*** $(\mathbf{0000})_2$ **,** ***1 :*** $(\mathbf{0001})_2$ **,** ***2 :*** $(\mathbf{0010})_2$ **,** ***3 :*** $(\mathbf{0011})_2$ **,** ***4 :*** $(\mathbf{0100})_2$ **,** ***5 :*** $(\mathbf{0101})_2$ **,**
6 : $(\mathbf{0110})_2$ ***et 7 :*** $(\mathbf{0111})_2$ **.**

b. Que peux-tu remarquer pour le bit de poids fort des nombres négatifs ? Et des nombres positifs ?
On admettra que cette remarque est toujours vérifiée !
Je remarque que le bit de poids fort des nombres négatifs est toujours 1 et que celui des nombres positifs est toujours 0.

5. a. Code le nombre relatif -1 sur 6 bits, puis sur 1 octet.
Sur 6 bits, on code -1 par l'entier naturel $-1+2^6=63$ **,** ***soit*** $(\mathbf{111111})_2$ **.**
Sur 1 octet = 8 bits, on code -1 par l'entier naturel $-1+2^8=255$ **,** ***soit*** $(\mathbf{11111111})_2$ **.**

b. Que remarques-tu ? ***On admettra que cette remarque est toujours vérifiée !***
Je remarque que le code du nombre -1 n'est toujours constitué que de 1 quelque soit le nombre de bits utilisés.

Ce qu'il faut savoir :

• Si on connaît le code avec la méthode du complément à 2 d'un entier relatif, pour obtenir le code de son opposé, on réécrit le code du nombre de départ en remplaçant les 1 par des 0 et les 0 par des 1, puis on ajoute $(1)_2$ au code obtenu(en prenant garde aux retenues en base 2).

Exemple : $(110100)_2$ est le code de -12 sur 6 bits.

Je remplace les 1 par des 0 et les 0 par des 1 : $(001011)_2$.

Puis j'ajoute $(1)_2=(000001)_2$: $(001011)_2+(000001)_2=(001100)_2$.

$(001100)_2$ représente bien l'entier naturel 12 (compris entre -2^5 et 2^5-1).

• Avec la méthode du complément à 2, le code d'un entier relatif positif commence toujours par 0 et celui d'un entier relatif négatif commence toujours par 1.

• Avec la méthode du complément à 2, le code de l'entier -1 n'est constitué que de 1.

Exemple : le code de -1 est $(1111)_2$ sur 4 bits, ou $(111111)_2$ sur 6 bits ou encore $(11111111)_2$ sur un octet.

Activité 4 : Quels nombres entiers relatifs sont représentés, par la méthode du complément à 2 par

1. $(00000000)_2$ ***0***

2. $(10000000)_2$ $x+2^8=2^7$ $x=2^7-2^8=-128$

3. $(01111111)_2$ $2^6+\ldots+2^0$ $=127$

4. $(10000001)_2$. $x+2^8=2^7+2^0$ $x=2^7+2^0-2^8=-127$

aide : Commence par déterminer si cela représente un nombre positif ou négatif.
Puis convertis ces codes binaires en des entiers naturels. Pour finir conclus.

Activité 5 :

1. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2**i
    print(a)
print(time()-t)
```

a. Teste et explique ce que fait ce programme.

Ce programme calcule et affiche 2^i pour i variant de 0 à 4 999 (5 000 calculs). Puis il calcule et affiche le temps nécessaire à Python pour réaliser ces calculs.

b. Quel est le dernier résultat affiché ?

Le dernier résultat affiché est le temps mis pour que Python réalise les calculs (ici moins de 0,5 s).

2. Considérons le programme ci-dessous sur Python.

```
from time import time
t=time()
for i in range(5000) :
    a=2222**i
    print(a)
print(time()-t)
```

a. Teste (sois patient!) et explique ce que fait ce programme.

Ce programme calcule et affiche 2222^i pour i variant de 0 à 4 999 (5 000 calculs). Puis il calcule et affiche le temps nécessaire à Python pour réaliser ces calculs.

b. Quel est le dernier résultat affiché ?

Le dernier résultat affiché est le temps mis pour que Python réalise les calculs (ici près de 40 s soit 80 fois plus lent).

3. Comment peux-tu expliquer la différence de résultats pour les questions ***c.*** des questions ***1.*** et ***2.*** ?

Python a besoin de beaucoup plus de temps pour calculer avec des nombre entiers très grands.

Ce qu'il faut savoir :

• Avec certains langages, le nombre d'octets avec lequel on travaille pour représenter les entiers relatifs peut être précisé (***1, 2, 4 ou 8 octets***, c'est à dire ***8, 16, 32 ou 64 bits***).

• Avec Python, il est inutile de se soucier de ce nombre, car c'est le logiciel qui décide seul combien d'octets sont nécessaires pour coder les entiers relatifs. Avec Python, le type des nombres entiers se nomme ***int*** (pour integer = entier en Français).

• Cependant, si un entier dépasse une certaine taille, le logiciel Python le découpe en plusieurs parties pour traiter les opérations. Ceci a pour conséquence de ***ralentir de façon importante les calculs*** et de ***nécessiter davantage d'espace mémoire***.

D. Exercices d'application :

Exercice 1 : Donne, avec la méthode du complément à 2, code les entiers ci-dessous sur 1 octet.

a. 100. $(\mathbf{01100100})_2$ ***b.*** 75. $(\mathbf{01001011})_2$ ***c.*** -50. $(\mathbf{11001110})_2$ ***d***. -128. $(\mathbf{10000000})_2$ ***e.*** -1. $(\mathbf{11111111})_2$ ***f.*** 128. ***Impossible*** ***g.*** -89. $(\mathbf{10100111})_2$

Exercice 2 : Dans chaque cas, donne le code des entiers a et b sur 1 octet avec la méthode du complément à 2. Pose et effectue la somme des 2 codes obtenus, puis vérifie que c'est bien le code de l'entier $a+b$.

1. $a=35$ et $b=65$.
$(00100011)_2+(01000001)_2=(01100100)_2$.
Or $(01100100)_2$ *représente* $2^6+2^5+2^2=100$.
Et on a bien $35+65=100$.

2. $a=-12$ et $b=45$.
$(11110100)_2+(00101101)_2=(100100001)_2$.
Mais comme sur 1 octet on ne garde que 8 bits, le bit de poids fort n'est pas gardé et le résultat est $(00100001)_2$ *qui représente* $2^5+2^0=33$.
Et on a bien $-12+45=33$.

3. $a=-84$ et $b=29$.
$(10101100)_2+(00011101)_2=(11001001)_2$
Or $(11001001)_2$ *représente l'entier relatif x, avec* $x+2^8=2^7+2^6+2^3+2^0$, *soit* $x=-55$.
Et on a bien $-84+29=-55$.

Exercice 3 : Détermine les nombres entiers relatifs qui sont représentés, avec la méthode du complément à 2, par les codes suivants. ***a.*** $(11001011)_2$. ***b.*** $(11010100)_2$. ***c.*** $(10000010)_2$. ***d.*** $(10101010)_2$.

a. $x+2^8=2^7+2^6+2^3+2^1+2^0$, *donc x=-53.*

b. $x+2^8=2^7+2^6+2^4+2^2$, *donc x=-44.*

c. $x+2^8=2^7+2^1$, *donc* $x=-126$.

d. $x+2^8=2^7+2^5+2^3+2^1$, *donc* $x=-86$.

Exercice 4 : VRAI ou FAUX ? (Justifie tes réponses)

1. Les premiers ordinateurs sont apparus au début du XIXè siècle.
Le premier ordinateur (le ABC) est apparu à la fin des années 1930 (au début du XXè siècle). Donc c'est FAUX.

2. 1 000 s'écrit aussi $(ABC)_{16}$ en hexadécimal.
$(\mathbf{ABC})_{16}$ ***Représente le nombre*** $10\times16^2+11\times16^1+12\times16^0$ ***= 2 748. Donc c'est FAUX.***

3. $(1001)_2\times(111)_2=(111111)_2$.

```
  1001
X  111
------
  1001
 10010
+100100
------
111111
```

Ainsi (1001)$_2$ X (111)$_2$ = (111111)$_2$. Donc c'est VRAI.

4. L'entier relatif -2 est codé sur 5 bits $(10010)_2$ avec la méthode du complément à 2.
-2 est codé par l'entier naturel $-2+2^5=-2+32=30$ ***, ce qui donne*** $(11110)_2$ ***, donc c'est FAUX.***

5. Tout ce qui est numérisé dans un ordinateur est identique aux informations réelles.
Les informations discrètes (Nombres entiers et les lettres) qui sont numérisées sont identiques aux informations réelles. Cependant, pour les informations continue (sons, images, nombres réels), les infirmations numérisées sont des approximations des informations réelles. Donc c'est FAUX.

Exercice 5 : Écrire un programme sur Python qui demande à l'utilisateur de saisir un entier relatif r codable sur 1 octet avec la méthode du complément à 2, qui détermine le code et le renvoie.
Aide : Revoir l'exercice 6 de la séquence précédente (=Écriture d'un entier naturel en base b⩾2).

```
n=-130
#On donne à n une valeur non comprise entre -128 et 127 pour que la boucle
#while puisse démarrer

while n<-2**7 or n>2**7-1:    #-2**7=-128 et 2**7-1=127
    n = int(input("Entrez un nombre entier compris entre -128 et 127: "))
#Tant que l'entier saisi n'est pas compris entre -128 et 127 on recommence.

b = ""

if n<0:    #Si n est négatif, on va coder l'entier n+2**8.
    n=n+2**8


while n!= 0:    #On reprend le programme de l'exercice 6 de la séquence précédente.
    q = int(n/2)
    b = str(n-q*2)+b
    n = q

while len(b)<8:  #On ajoute des 0 sur les bits de gauche qui ne seraient pas utilisés.
    b="0"+b


print(b)
```

Séquence : *Représentation binaire approximative d'un réels : les flottants.*	**Numérique et Sciences Informatiques**	***Thème :*** *Représentation des données (types de base)*

I – Représentation de la partie à droite de la virgule d'un nombre à virgule

En notation décimale, les chiffres à gauche de la virgule représentent des entiers, des dizaines, des centaines, des milliers, … etc, et ceux à droite de la virgule représentent des dixièmes, centièmes, millièmes, etc.

Ainsi : $3{,}8125_{(10)} = 3\times10^{0} + 8\times10^{-1} + 1\times10^{-2} + 2\times10^{-3} + 5\times10^{-4}$

De la même façon, pour un nombre à virgule en base 2, on utilise les puissances négatives de 2.
Observe cet exemple :

$$\begin{aligned} 11{,}1101_{(2)} &= 1\times2^{1} + 1\times2^{0} + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} \\ &= 2 + 1 + \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16} \\ &= 2 + 1 + 0{,}5 + 0{,}25 + 0 + 0{,}0625 \\ &= 3{,}8125_{(10)} \end{aligned}$$

De la même manière, convertis le nombre suivant en base décimale : $(101{,}010101)_{2,}$

$$(\mathbf{101{,}010101})_{\mathbf{2}} = \mathbf{5{,}328125}.$$

II – Conversion en binaire de la partie à droite de la virgule

1. Observez le schéma ci-contre :

Que permet-il de faire ?

$0{,}375\times10 = \boxed{3}\ ,75$

$0{,}75\ \times10 = \boxed{7}\ ,5$

$0{,}5\ \times10 = \boxed{5}\ ,0$

2. Faites la même chose avec 0,42 et 0,8769. Comment sait-ton que l'on a terminé ?

$0{,}42\ \times10 = \boxed{4}\ ,2$

$0{,}2\ \times10 = \boxed{2}\ ,0$

$0{,}8769\ \times10 = \boxed{8}\ ,769$

$0{,}769\ \times10 = \boxed{7}\ ,69$

$0{,}69\ \times10 = \boxed{6}\ ,9$

$0{,}9\ \times10 = \boxed{9}\ ,0$

On sait qu'il fait s'arrêter lorsqu'il reste 0 à droite de la virgule.

3. Pour obtenir en binaire la partie à droite de la virgule, on procède de la même façon, mais en multipliant par 2 à chaque étape car on travaille alors en base 2.

a. En adaptant le procédé ci-dessus, déterminer la partie à droite de la virgule en base 2 de 0,375.

b. Vérifiez que le résultat obtenu en base 2, lorsqu'il est converti en base 10 redonne bien 0,375.

c. Faire la même chose avec 0,625 puis vérifier le résultat obtenu.

d. Faire de même avec 0,1. Que remarquez-vous ?

$0,375\times 2 = \boxed{0}\ ,75$

$0,75\times 2 = \boxed{1}\ ,5$

$0,5\times 2 = \boxed{1}\ ,0$

a.

$0,625\times 2 = \boxed{1}\ ,25$

$0,25\times 2 = \boxed{0}\ ,5$

$0,5\times 2 = \boxed{1}\ ,0$

$0,375_{(10)} = 0.011_{(2)}$

b. $(0,011)_2 = 2^{-2}+2^{-3} = 0,375$

c. $0,625_{(10)} = 0.101_{(2)}$ $\qquad (0,101)_2 = 2^{-1}+2^{-3} = 0,625.$

d. Développement binaire infini.
On retrouve un reste déjà rencontré. Il y a une période.
On obtient en binaire : 0,00011001100 …

4. ***a.*** On souhaite réaliser un algorithme en langage naturel qui effectue cette conversion.
En observant les conversions faites précédemment, écrire en langage naturel les instructions qui sont répétées.

```
b = a × 2
afficher la partie entière de b
si partie entière de b = 1
    a = b – 1
sinon
    a = b
Fin Si.
```

b. Peut-on prévoir à l'avance combien de fois elles vont être répétées ?
Si non, comment peut-on savoir à quel moment il faudra arrêter de les répéter ?

On ne peut pas savoir. On s'arrête dès que a=0 ou alors quand on atteint un nombre fixé de bits.

c. Rédiger un algorithme en langage naturel pour effectuer la conversion.

5. Voilà un programme Python qui met en œuvre cet algorithme.
Les bits sont stockés dans une chaîne de caractères.

```
from decimal import *

n = Decimal(input("Entrer la partie décimale : "))
en_binaire = ''

i = 0
while (n != 0) and (i < 64):
    n = n*2
    if n >= 1:
        en_binaire = en_binaire + '1'
        n = n - 1
    else:
        en_binaire = en_binaire + '0'
    i = i+1

print(en_binaire)
```

a. Testez le programme avec les valeurs du début de l'activité.
b. Modifiez le programme pour qu'il affiche les calculs intermédiaires comme ci-dessous :

$$0,375 \times 2 = 0\ ,\ 75$$
$$0,75\ \times 2 = 1\ ,\ 5$$
$$0,5\ \times 2 = 1\ ,\ 0$$

Après la ligne « `while (n != 0) and (i < 64):` », écrire « print(« n, "x2=",n*2) ».

6. Reprenez le programme Python précédent et remplacez **Decimal** par **float.**
Reprenez les tests précédents. Que remarquez-vous ?

Problème dans les calculs intermédiaires. Il va falloir se pencher sur l'encodage des nombres à virgule (norme IEEE-754)

NOM et Prénom :

L'accident de Dhahran

Dans le tableau ci-dessous, placez le développement binaire de 0,1 en prenant soin de noter sa partie entière dans le bit de poids fort et indiquez soigneusement la place de la virgule.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	...
0,	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	…

La batterie Patriot dispose d'un registre de 24 bits. Indiquez dans le tableau ci-dessus quelle est la partie du nombre qui est effectivement stockée en mémoire.

Après avoir analysé le document du groupe d'experts du GAO, écrivez dans le tableau ci-dessous l'erreur commise dans le stockage de l'unité de temps.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	...
0,	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	…

Exprimez cette erreur en base 10 :

$$\varepsilon = 2^{-24} + 2^{-25} + 2^{-28} + 2^{-29} + 2^{-32} + 2^{-33} = \frac{2^9 + 2^8 + 2^5 + 2^4 + 2 + 1}{2^{33}} = \frac{819}{8589934592} \approx 9{,}53 \cdot 10^{-8}$$

Calculez l'erreur cumulée au bout de 1h de fonctionnement : $\varepsilon_1 \approx 9{,}53 \cdot 10^{-8} \times 10 \times 3600 \approx 0{,}0034308$ s

Dans le tableau donné en annexe par le GAO (Appendix II), contrôlez les résultats affichés dans les lignes correspondant à 8h et à 100h de fonctionnement.

$\varepsilon_8 \approx 9{,}53 \cdot 10^{-8} \times 10 \times 3600 \times 8 \approx 0{,}0274464$ s et $\varepsilon_{100} \approx 9{,}53 \cdot 10^{-8} \times 10 \times 3600 \times 100 \approx 0{,}34308$ s

Sachant qu'un Scud a une vitesse d'environ 3750 mph, quelle est la distance qu'il parcourt durant le laps de temps correspondant à l'erreur commise pour une centaine d'heures de fonctionnement (1 mile ≈ 1,6 km) :

$$\Delta_{100} \approx 0{,}34308 \times \frac{3750}{3600} \times 1{,}6 \times 100 \approx 572 \text{ m}$$

A. Codage des nombres réels :

Ce qu'il faut savoir : Le codage ***sur 64 bits*** (8 octets) en binaire des nombres réels pour les représenter en machine est défini par ***la norme IEEE 754***.

Pour représenter un nombre à virgule en binaire, on va utiliser une représentation similaire à la notation scientifique en base 10, mais ici en base 2 : il s'agit de ***la représentation en virgule flottante***.

Cette représentation est $Sm\times 2^n$ où ***S*** est ***le signe*** (+ ou -), ***m*** est ***la mantisse*** ($1\leqslant m<2$) et ***n l'exposant*** qui est un nombre entier relatif.

On codera en binaire ce nombre ainsi représenté sur 64 bits ainsi :

64è bit	63è bit	61è bit	….....	53è bit	52è bit	51è bit	50è bit	49è bit	….....	3è bit	2è bit	1er bit
signe	**exposant décalé**				**m – 1 = partie décimale de la matisse**							

1 bit *(0 code + et 1 code -)*

11 bits *codent un nombre entier naturel* ***d = exposant décalé*** *= n+1 023* *(0 ≤ d ≤ 2 047).*

Si d=0 ou 2 047, cela est réservé à des situations exceptionnelles telles que ***l'infini*** *ou* ***« not a number »****.*

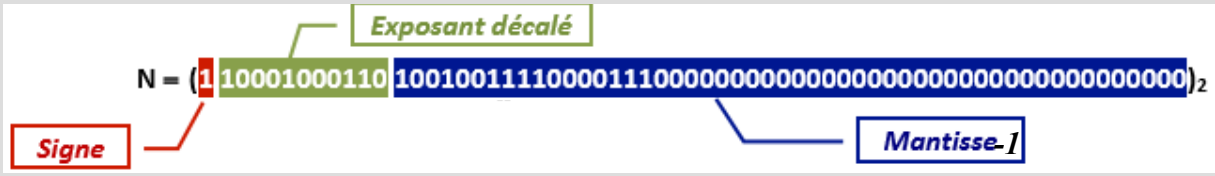
Si 1≤ d ≤ 2 046, ***on soustrait 1 023*** *pour obtenir l'exposant n qui est un entier relatif.*

52 bits *codent m-1 avec 1≤ m<2.*

La partie entière de m est toujours égale à 1, *il est inutile de coder ce chiffre et de perdre 1 bit.*

m-1, la partie décimale de m est une somme de puissances négatives de 2.

Exemple :



Signe = 1 : le nombre ***est négatif***.

Exposant décalé d = $(10001000110)_2=2^{10}+2^6+2^2+2^1=1024+64+4+2=$ 1 094.
Donc ***l'exposant n*** = 1 094-1 023=***71***.

Mantisse = $(1{,}10010011110000111000\ldots)_2=1+2^{-1}+2^{-4}+2^{-7}+2^{-8}+2^{-9}+2^{-10}+2^{-15}+2^{-16}+2^{-17}\approx 1{,}5772$.

La partie entière de la mantisse est toujours 1.

DONC ce code représente le réel : $-1{,}5772\times 2^{71}\approx -3{,}724\times 10^{21}$.

Cas particulier : On convient que le réel ***0 est codé par (0 00000000000 0000...00)$_2$***.

Activité 1 :

1. Trouve le nombre réel codé au format IEEE 754 par :

0 00100000011 11010011100101011000000000000000000000000000000000000.

Signe = 0 : le nombre est positif.

Exposant décalé d = $(00100000011)_2 = 2^8+2^1+2^0 = 259$ ***.***

Donc l'exposant n = 259-1 023=-764.

Mantisse $= (1,11010011100101011000\ldots)_2 = 1+2^{-1}+2^{-2}+2^{-4}+2^{-7}+2^{-8}+2^{-9}+2^{-12}+2^{-14}+2^{-16}+2^{-17}$

$\approx$ ***1,826 499 939***

Donc ce code représente le réel 1,826 499 939×2^{-764} $\approx$ ***1,882 361*** 254×10^{-230}

2. Inversement, on veut maintenant coder le nombre réel -118,375 au format IEEE 754.

a. Divise 118,375 par 2 autant de fois qu'il le faut afin d'obtenir un résultat supérieur ou égal à 1 et strictement inférieur à 2. Déduis-en l'écriture de -118,375 sous la forme $Sm\times2^n$ où S est le signe, m la matisse et n l'exposant.

$118,375\div2=59,1875$
$59,1875\div2=29,59375$
$29,59375\div2=14,796875$
$14,796875\div2=7,3984375$ ***donc*** $-118,375=-1,849609375\times2^6$
$7,3984375\div2=3,69921875$
$3,69921875\div2=1,849609375$

b. A quel bit correspond le signe S ? ***- correspond au bit 1.***

c. Calcule l'exposant décalé. ***L'exposant décalé est*** $n+1023=6+1023=1029$ ***.***

d. A quel code binaire correspond cet exposant décalé ?

1 029 se code sur 11 bits par : $(10000000101)_2$***.***

e. Calcule la partie décimale de la mantisse. $1,849609375-1=0,849609375$ ***.***

f. Multiplie la partie décimale de la mantisse par 2. Si le résultat est strictement inférieur à 1, le premier bit du code de la partie décimale de la mantisse est 0. En revanche, si le résultat est supérieur à 1, le premier bit est 1 et tu dois soustraire 1 au résultat.

Puis recommence ceci : multiplie le dernier résultat par 2. Si tu obtiens un résultat strictement inférieur à 1, ajoute le bit 0 à droite du code, sinon ajoute 1 à droite du code et soustrais 1 au résultat. Etc...

Tu devras t'arrêter lorsque tu obtiendras exactement 1. A ce moment, tu ajouteras le bit 1 à droite du code, puis tu compléteras par des 0 à droite du code pour qu'il y ait bien 52 bits.

Donne ainsi le code de la partie décimale de la mantisse sur 52 bits.

$0,849609375\times2=1,69921875$ ***donc le premier bit est 1.***
$0,69921875\times2=1,3984375$ ***donc le second bit est 1.***
$0,3984375\times2=0,796875$ ***donc le 3è bit est 0.***
$0,796875\times2=1,59375$ ***donc le 4è bit est 1.***
$0,59375\times2=1,1875$ ***donc le 5è bit est 1.***
$0,1875\times2=0,375$ ***donc le 6è bit est 0.***
$0,375\times2=0,75$ ***donc le 7è bit est 0.***
$0,75\times2=1,5$ ***donc le 8è bit est 1.***
$0,5\times1=1$ ***donc le 9è bit est 1, et c'est terminé, tous les bits suivants sont 0.***

Ainsi la partie décimale de la mantisse se code $(110110011000...00)_2$***.***

g. Déduis-en le code au format IEEE 754 de -118,375.
Donc le code de -118,375 au format IEEE 754 est $(1\ 10000000101\ 110110011000\ldots)_2$.

3. En reproduisant la méthode de la question 2., détermine le code au format IEEE 754 de ces nombres réels.

a. 75,281 25.

75,281 25= $1,1762695313\times 2^6$
$(0\ 10000000101\ 00101101001000\ldots)_2$

b. -0,687 5.

$-06875=-1,375\times 2^{-1}$
$(1\ 01111111110\ 011000\ldots)_2$

Activité 2 :

1. a. Comment code-t-on en binaire le nombre entier -7 (sur 8 bits) ? Et le nombre réel -7,0 (sur 64 bits) ?
Le nombre entier relatif -7 se code sur 8 bits (avec la méthode du complément à 2) par l'entier naturel $-7+2^8=249$ ***. On obtient donc*** $(11111001)_2$ ***.***

$-7=-1,75\times 2^2$ ***, ainsi le code au format IEEE 754 est :*** $(1\ 10000000001\ 11000000000000000000000000000000\ldots)_2$.

b. Pourquoi le même nombre n'est-il pas codé de la même façon ? ***Parce que les types sont différents.***
Quelle précaution faudra-t-il prendre lorsqu'on écrit un programme (notamment avec Python) ?
Lorsqu'on écrit un programme il est donc essentiel, lorsqu'on définit une variable, de savoir de quel type elle est (ici l'entier -7 est de type « integer » et le réel -7,0 est de type « float »).

2. a. Applique la méthode de l'activité 1 pour tenter de convertir le nombre réel 0,2 au format IEEE 754.
Que se passe-t-il ?
$0,2\times 2=0,4$ ***;*** $0,4\times 2=0,8$ ***;*** $0,8\times 2=1,6$ ***. Donc*** $0,2=1,6\times 2^{-3}$ ***.***
Le signe est + donc le premier bit est 0.
L'exposant décalé est -3+1 023=1 020 qui se code $(01111111100)_2$.
La partie décimale de la mantisse est 0,6.
$0,6\times 2=1,2$ ***donc le premier bit est 1.***
$0,2\times 2=0,4$ ***donc le 2è bit est 0.***
$0,4\times 2=0,8$ ***donc le 3è bit est 0.***
$0,8\times 2=1,6$ ***donc le 4è bit est 1.***
Comme 1,6-1=0,6 c'est cyclique, et les bits« 1001 »vont se répéter indéfiniment= $(10011001\ldots1001\ldots)_2$.

b. Quand les 52 bits de la mantisse ne suffisent pas, si le 53è bit est 1, on arrondit le bit de la mantisse qui est le plus à droite par excès. Mais si le 53è bit est 0, on arrondit le bit de la mantisse qui est le plus à droite par défaut. Quel sera finalement le code de 0,2 au format IEEE 754 ?
La mantisse se code $(1001100110011\ldots)_2$ ***On a donc*** $(1001\ 1001\ 1001\ldots1001\ 1\ldots)_2$ ***, en arrondissant :***
53è bit
$(10011001100\ldots1010)_2$ ***.***

Ainsi le code au format IEEE 754 de 0,2 est $(0\ 01111111100\ 10011001100\ldots1010)_2$.

c. Déduis-en le code au format IEEE 754 du réel 0,1. Puis applique la méthode de l'activité 1 pour trouver celui du réel 0,3.
On a vu que $0,2=\left(1+2^{-1}+2^{-4}+2^{-5}+2^{-8}+2^{-8}+\ldots\right)\times 2^{-3}$ ***. En divisant par 2(ou en multipliant par*** 2^{-1} ***)***
on obtient $0,1=\left(1+2^{-1}+2^{-4}+2^{-5}+2^{-8}+2^{-8}+\ldots\right)\times 2^{-4}$ ***. Donc seul l'exposant change (c'est -4).***
Ainsi l'exposant décalé est -4+1 023 = 1 019 qui se code $(01111111011)_2$ ***.***
Le code au format IEEE 754 de 0,1 est $(0\ 01111111011\ 10011001100\ldots1010)_2$.

On a $0,3\times2=0,6$; $0,6\times2=1,2$ *donc* $0,3=1,2\times2^{-2}$.
Le bit correspondant au signe + est 0.
L'exposant décalé est -2+1 023=1 021 qui se code $(01111111101)_2$ *sur 11 bits.*
La partie décimale de la mantisse est 0,2.
$0,2\times2=0,4$ *donc le premier bit est 0.*
$0,4\times2=0,8$ *donc le 2è bit est 0.*
$0,8\times2=1,6$ *donc le 3è bit est 1.*
$0,6\times2=1,2$ *donc le 4è bit est 1.*
Comme 1,2-1=0,2 c'est cyclique, et les bits« 0011 »vont se répéter indéfiniment= $(00110011...0011...)_2$.
La mantisse se code $(0011\ 0011\ 0011...0011\ 0)_2$ *, en arrondissant :* $(0011\ 0011\ 0011...0011)_2$.

53è bit

Ainsi le code au format IEEE 754 de 0,3 est $(0\ 01111111101\ 001100110011...0011)_2$.

d. Ouvre Python et tape dans la console : ***>>>(0.1+0.2)-0.3*** puis ***« Entrée »***.
Pourquoi le résultat est-il surprenant ? Comment l'expliquer ?

```
>>> (0.1+0.2)-0.3
5.551115123125783e-17
```

Ce résultat différent de 0 est surprenant car en théorie le résultat devrait être 0.
Ceci s'explique par les arrondis réalisés sur les mantisses au 52è chiffre.

Ce qu'il faut savoir :

• Il est impératif de bien distinguer le type ***« integer »*** du type ***« float »*** dans un programme. En particulier l'entier 7 et le flottant 7.0 ne se codent pas de la même manière en binaire dans une machine.

• En binaire, sur 8 octets (64 bits), avec ***la méthode du complément à 2***, les codes représentent ***exactement*** les nombres entiers compris entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807.
En revanche avec ***la norme IEEE 754*** (sur 64 bits aussi), les codes représentent ***parfois des approximations*** des nombres réels compris entre $-1,7976931348623157\times10^{308}$ et $1,7976931348623157\times10^{308}$.

• Sur machine les calculs avec les nombres entiers sont exacts, mais les calculs avec les nombres flottants peuvent présenter d'infimes erreurs d'arrondi.

B. Exercices d'application :

Exercice 1 : Détermine à quels nombres réels correspondent ces codes au format IEEE 754.

a. $(1\ 01111111110\ 1000...0)_2$.
signe = -
exposant décalé = 1 022
exposant = 1 022-1 023 = -1
mantisse $=1+2^{-1}=1,5$
le réel est $-1,5\times2^{-1}=-0,75$.

b. $(0\ 10000001000\ 0001100101011100...0)_2$.
signe = +
exposant décalé = 1 032
exposant = 1 032 – 1 023 = 9
mantisse $=1+2^{-4}+2^{-5}+2^{-8}+2^{-10}+2^{-12}+2^{-13}+2^{-14}$ ***=1,09906005859375***
le réel est $1,09906005859375\times2^{9}=562,71875$.

c. $(0\ 00000000000\ 000...0)_2$.
d'après le cours, le réel est 0.

Exercice 2 : Convertis les nombres suivants au format IEEE 754.
Pour chaque cas, précise si les codes représentent exactement ou approximativement les nombres réels.

a. 3,625. ***b.*** -4,5. ***c.*** -50. ***d***. $-\frac{11}{15}=-0,7333\ldots$ ***e.*** 1.

$3,625 \div 2^1 = 1,8125$ *(1 10000000001 00100...00)₂.* *(1 01111111110 01110111...0111)₂.*
$3,625 = +1,8125 \times 2^1$ *(1 10000000100 100100...00)₂.* *(0 01111111111 00...)₂.*
signe + : 0
expo décalé 1024 : 10...0
partie décimale mantisse 0,8125 :
$0,8125 \times 2 = 1,625$ ***donc bit 1***
$0,625 \times 2 = 1,25$ ***donc bit 1***
$0,25 \times 2 = 0,5$ ***donc bit 0***
$0,5 \times 2 = 1$ ***donc bit 1 et fin***
partie décimale de la mantisse : 110100...00

Donc 3,625 se code (0 10000000000 110100...00)₂.