Données 4 - Encodage des entiers naturels

I – Encodage en bits

Les entiers naturels sont ceux de l'ensemble ℕ. On dit qu'ils sont non signés ou _____.
Le nombre de cas différents est 2^X
La valeur maximale encodable avec **X** bits est $2^X - 1$

Ex 01 (8 bits) : Que vaut **1111 1111** $_2$ en base 10 ? Quelle formule permet de retrouver cela ? Convertir ensuite en base 10 les deux octets suivants : **0000 1111** $_2$ **0000 1001** $_2$

Ex 02 (9 bits) : Que valent **1 0000 0000** $_2$ et **1 0000 0001** $_2$ en base 10 ?

Fonction native `bin()`

La fonction native **`bin()`** fournit un **string** commençant par **`0b`** et contenant les bits du nombre fourni .

Ex 03 : Compléter les réponses que va fournir Python :

```
>>> bin(255)
'0b11111111'
```

```
>>> bin(256)
'0b100000000'
```

```
>>> bin(257)
```
_____

```
>>> bin(255)[2:]
```
_____

A. Qu'est-ce qui indique clairement que la réponse de la fonction native **bin()** est de type string ?

B. Dans cette réponse, qu'est-ce qui indique qu'il s'agit d'un contenu affiché en binaire ?

Expression binaire en Python

Ex 04-05 : Compléter les réponses que va fournir Python :

```
>>> 0b100000001
257
```

```
>>> 0b11
```
_____

```
>>> int('0b11')
```
_____

```
>>> int('11', 2)
```
_____

II – Encodage d'un entier naturel sur plusieurs octets

Avec 2 octets (16 bits), on obtient **2^{16}= 65536** cas différents de **0 à 65535**.

2.1 Encodage avec 2 octets (méthode 1) : lecture directe bit par bit

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

Que vaut le nombre entier N positif encodé ? _____

2.1 Encodage avec 2 octets (méthode 2) : lecture comme une suite de deux octets Y Z

On lit par bloc de 8 bits à la fois puis on calcule N =

La valeur de cet octet **Y** est à multiplier par 256							
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	**0**	**0**	**0**	**0**	**1**	**1**	**1**
Valeur base 10 de **Y** :							

La valeur de cet octet Z est à multiplier par 1							
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	**1**	**1**	**1**	**0**	**0**	**0**	**0**
Valeur base 10 de **Z** :							

Que vaut le nombre entier positif encodé ? _____

Ex 06 : Déterminer les valeurs encodées.

- Nombre N_A : **6 - 4**
- Nombre N_B : **3 - 4**

Ex 07 : Déterminer les valeurs encodées.

- Nombre N_C : **5 - 130**
- Nombre N_D : **130 - 5**

Question : l'ordre des octets reçus a-t-il un impact sur le nombre décodé ?

Ex 08 : Déterminer les valeurs encodées par les suites d'octets suivantes :

- Nombre N_E : **0 - 65**
- Nombre N_F : **65 - 0**

Question : si on veut exprimer l'entier 10 sur deux octets, doit-on écrire :

0000 0000 0000 1010 ou
0000 1010 **0000 0000**

2.2 Encodage avec 4 octets W X Y Z

```
N =  W * 2**24   +  X * 2**16   + Y * 2**8    +  Z
N =  W * 256**3  +  X * 256**2  + Y * 256**1   +  Z
```

Ex 09 : Déterminer la valeur encodée par la suite d'octets suivante : **N = 1 - 2 - 3 - 4**
Ex 10 : Déterminer la valeur encodée par la suite d'octets suivante : **N = 10 - 20 - 255 - 0**
Ex 11 : Déterminer la valeur encodée par la suite d'octets suivante : **N = 255 - 255 - 255 - 255**
Question supplémentaire : comparer votre résultat à $2^{32}-1$. Justifiez l'égalité.
Ex 12 : Combien d'octets faut-il pour encoder un entier naturel valant 6 milliards :
A. 1 octet B. 2 octets C. 4 octets D. 8 octets

2.3 Tableau statique : toutes les cases doivent contenir le même nombre d'octets

Ex 13 : Quelle place réserver pour un tableau de 100 cases contenant des valeurs allant jusqu'à 200 ?
Ex 14 : Quelle place réserver pour un tableau de 100 cases contenant des valeurs allant jusqu'à 65535 ?
Ex 15 : Quelle place réserver pour un tableau de 100 cases contenant des valeurs allant jusqu'à 4G ?

III – Dépassement

3.1 Addition de deux entiers sur un nombre de bits fixés

Pour savoir réaliser les additions, il faut se souvenir de 4 cas :

Base 10 : 0+0+0 = 0	Base 2 : **0+0+0 = 0**
Base 10 : 0+0+1 = 1	Base 2 : **0+0+1 = 1**
Base 10 : 0+1+1 = 2	Base 2 : **0+1+1 = 10** (on a donc une retenue)
Base 10 : 1+1+1 = 3	Base 2 : **1+1+1 = 11** (on a donc une retenue)

Exercice d'application sur l'addition **C = A + B** en réalisant la somme bit à bit.

A		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
B		0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
Retenue																	
C = A+B																	

Ex 16 : Réaliser l'addition suivante sur 16 bits.

A		0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
B		0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0
Retenue																	
C = A+B																	

Ex 17 : Réaliser l'addition suivante. Sur 16 bits

A		1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
B		0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0
Retenue																	
C = A+B																	

3.2 Dépassement :

On parle de **dépassement** lorsque la somme réelle ***devrait*** donner un résultat supérieur à la valeur maximale encodable avec ce nombre d'octets.

Exemple : **A = 16384** = 01000000 00000000 mais **C = A + B = 2** sur 16 bits

B = 49154 = 11000000 00000010

A		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B		1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Retenue		+1															
C = A+B	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

3.3 Dépassement avec Python ? Non, Python gère cela à l'interne.

3.4 Détection d'un dépassement lors d'une addition **C = A + B**

Après avoir fait l'addition, on vérifie les deux assertions suivantes : **C > A** **C > B**

Si **l'une des assertions est évaluée** à **FAUX**, c'est qu'il y a dépassement.

Remarque ***:*** *une assertion est une proposition qu'on considére VRAI.*

Exemple : **16384** + **49154** donne **2** sur 16 bits. **2** n'est pas supérieur à **16384**. Dépassement détecté.

3.5 Détection d'un dépassement lors d'une multiplication **C = A . B**

Après avoir fait la multiplication, on vérifie l'assertions suivante **C // A == B** **si A != 0**

Si **C // A n'est pas évaluée à B** , c'est qu'il y a eu un problème de dépassement.

Exemple : **100 * 50** qui donne **5000**. **5000 // 50** donne **100** : ok, pas de dépassement

Exemple : **400 * 500** qui donne **3392**. **3392 // 400** donne **8** : on n'obtient pas 500. Dépassement.

Ex 18 : **Expliquer si il y a dépassement (ou pas) sur les cas suivants (calcul sur 16 bits)**

1. On considère l'addition de A = 16000 par B = 2000. On obtient C = 18000.
2. On considère la multiplication de A = 16000 par B = 2000. On obtient C = 18432.
3. On considère la multiplication de A = 1600 par B = 2000. On obtient C = 54272.
4. On considère la multiplication de A = 160 par B = 2000. On obtient C = 57856.
5. On considère la multiplication de A = 16 par B = 2000. On obtient C = 32000.

Ex 19 : Soit la multiplication de **A = 22000** par **B = 1250** sur 32 bits. On obtient **C = 27500000**.

19A - Expliquez comment on peut détecter s'il y a eu dépassement sans faire le calcul direct.

19B - Simuler ensuite le résultat du calcul sur 16 bits (2 octets) :

```
>>> C = (16000*2000) % (2**16)
```

Le calcul de C est-il bon ? Donner une explication.

Ex 20 : Voici un exercice de synthèse final.

On veut multiplier des entiers dont on vous donne l'encodage sur 4 octets.

Questions :

A. A combien de bits correspondent 4 octets ?
B. Donner la valeur de A encodé par les 4 octets suivants : **200 50 0 250** .
C. Donner la valeur de B encodé par les 4 octets suivants : **5 10 20 150** .
D. Expliquer s'il y a eu dépassement en considérant que la valeur C fournie par un programme C = A.B = **556 014 204**.
E. Donner le résultat C = A * B en le réalisant sur la console Python. Vérifier que Python réponde bien **C = 283 968 746 947 877 500**. Expliquer s'il y a eu dépassement.
F. Combien d'octets sont nécessaires à ce calcul ? 1, 2, 4, 8 ou 16 ?