Feuille d'exercices

Représentation des entiers naturels : binaire et hexadécimal

NSI – Première – Entraînement et approfondissement

Ces exercices vont un peu plus loin que ceux du cours : ils mélangent les bases dans un même énoncé et les relient à des situations concrètes (adresses mémoire, codage des couleurs, capacité de stockage). Tous se traitent **à la main**, sans ordinateur : les fonctions Python de conversion seront écrites lors d'un prochain chapitre, une fois les fonctions vues en cours.

Exercice 1: Tableau à trois colonnes

Compléter le tableau suivant, où chaque ligne représente le **même** nombre entier écrit dans les trois bases décimale, binaire et hexadécimale. Une seule case est donnée par ligne : à vous de retrouver les deux autres (vous pouvez passer par l'écriture décimale comme intermédiaire).

Décimal	Binaire	Hexadécimal
45		
	101101	
		3E
200		
	11110000	
		B4

Correction de l'exercice 1

Décimal	Binaire	Hexadécimal
45	101101	2D
45	101101	2D
62	111110	3E
200	11001000	C8
240	11110000	F0
180	10110100	B4

Remarque : les lignes 1 et 2 donnent le même nombre (45), ce qui permet de vérifier que $(101101)_2 = 45$ par deux chemins différents.

Exercice 2: Vrai ou faux ?

Pour chacune des affirmations suivantes, dire si elle est vraie ou fausse, et **justifier** (en cas de désaccord, corriger l'affirmation).

1. $(10)_2 = 10$.
2. Le plus grand nombre entier représentable sur un octet est 256.
3. $(FF)_{16} = (11111111)_2$.
4. Pour convertir un nombre binaire en hexadécimal, on le découpe en tranches de **trois** bits, en partant du bit de poids faible.
5. Le nombre 0 s'écrit $(0)_2$ aussi bien que $(0)_{16}$.
6. Il existe deux façons différentes d'écrire le nombre 13 en binaire.

Correction de l'exercice 2

1. **Faux.** $(10)_2 = 1 \times 2^1 + 0 \times 2^0 = 2$, et non 10 (qui serait l'écriture décimale du nombre dix).
2. **Faux.** Avec 8 bits, on représente les entiers de 0 à $2^8 - 1 = 255$: le plus grand est donc 255, pas 256 (qui nécessiterait un 9^{e} bit).
3. **Vrai.** $F = 1111$ en binaire, donc $(FF)_{16} = (1111\ 1111)_2$, soit huit bits tous à 1, ce qui correspond bien à $(11111111)_2$.
4. **Faux.** Le regroupement se fait par tranches de **quatre** bits (et non trois), car $2^4 = 16$ correspond exactement à un symbole hexadécimal.
5. **Vrai.** Par convention, la représentation de 0 est $(0)_b$ quelle que soit la base b.
6. **Faux.** D'après le théorème vu en cours, l'écriture en base 2 d'un entier naturel est **unique** : il n'existe donc qu'une seule écriture binaire de 13, à savoir $(1101)_2$.

Exercice 3: Adresses mémoire

Un disque dur possède 2^{24} cases mémoire, chacune contenant un octet, numérotées de 0 à $2^{24} - 1$. Chaque case est repérée par son **adresse**, exprimée en hexadécimal.

1. Combien de bits faut-il, au minimum, pour écrire en binaire l'adresse d'une case quelconque de ce disque ?
2. En déduire sur combien de chiffres hexadécimaux s'écrit une adresse de ce disque (indication : un chiffre hexadécimal « vaut » quatre bits).
3. Donner l'adresse hexadécimale de la **première** case mémoire (numéro 0) et celle de la **dernière** case mémoire (numéro $2^{24} - 1$).
4. Ce disque dur correspond-il plutôt à une clé USB, un disque dur d'ordinateur portable, ou un serveur de données ? Justifier en calculant sa capacité en octets, puis en Mo ou Go (voir le tableau des préfixes du cours).

Correction de l'exercice 3

1. Les adresses vont de 0 à $2^{24} - 1$, il faut donc exactement **24 bits** pour écrire la plus grande d'entre elles (et donc pour écrire n'importe laquelle, en complétant par des 0 à gauche si nécessaire).
2. $24 = 6 \times 4$: une adresse s'écrit donc sur exactement **6 chiffres hexadécimaux** (24 bits regroupés en 6 tranches de 4 bits).
3. La première case (numéro 0) a pour adresse $(000000)_{16}$. La dernière case a pour numéro $2^{24} - 1$, soit 24 bits tous à 1 : $(111111111111111111111111)_2 = (FFFFFF)_{16}$.
4. $2^{24} = 16\,777\,216$ octets, soit environ 16,8 millions d'octets $\approx$ 16,8 Mo. C'est une capacité correspondant plutôt à une **très petite** clé USB des années 2000 (aujourd'hui, les disques durs et clés USB courants se comptent en Go, voire en To).

Exercice 4: Codage des couleurs

Une couleur affichée à l'écran est codée par trois composantes – Rouge, Vert, Bleu – chacune comprise entre 0 et 255, et regroupées en un seul code hexadécimal de 6 chiffres : deux pour R, deux pour V, deux pour B.

1. La couleur `#3AC1E0` est utilisée sur un site web. Donner la valeur de chacune des composantes R, V, B, d'abord en hexadécimal, puis en décimal.

2. On veut au contraire coder la couleur de composantes $R = 120$, $V = 200$, $B = 45$ (en décimal). Donner le code hexadécimal à 6 chiffres correspondant.
3. On approxime la « luminosité » d'une couleur par la moyenne de ses trois composantes décimales, $\frac{R+V+B}{3}$. Calculer la luminosité de la couleur `#3AC1E0`. Une luminosité proche de 0 correspond à une couleur sombre, proche de 255 à une couleur claire : que peut-on dire de cette couleur ?
4. Pourquoi, selon vous, utilise-t-on l'hexadécimal plutôt que le binaire pour écrire ces codes couleurs dans les feuilles de style d'un site web ?

Correction de l'exercice 4

1. En découpant `3AC1E0` par paires de chiffres hexadécimaux : $R = (3A)_{16}$, $V = (C1)_{16}$, $B = (E0)_{16}$. En décimal : $R = 3 \times 16 + 10 = 58$, $V = 12 \times 16 + 1 = 193$, $B = 14 \times 16 + 0 = 224$.
2. $120 = (78)_{16}$, $200 = (C8)_{16}$, $45 = (2D)_{16}$ (on vérifie que $45 = 2 \times 16 + 13$). Le code recherché est donc `#78C82D`.
3. Luminosité $= \frac{58+193+224}{3} = \frac{475}{3} \approx 158{,}3$. Cette valeur est assez proche du maximum (255) : la couleur `#3AC1E0` est donc plutôt **claire** (c'est en réalité un bleu ciel assez lumineux).
4. L'hexadécimal permet d'écrire chaque composante (un octet, donc 8 bits) avec seulement **2** symboles au lieu de 8 en binaire : c'est bien plus compact et plus facile à relire ou à recopier pour un humain, tout en se convertissant instantanément en binaire (par tranches de 4 bits) pour la machine.

Exercice 5: Coder un capteur

Un capteur de température extérieure renvoie une valeur entière en dixièmes de degré, comprise entre 0 et 500 (soit 0,0°C à 50,0°C). On veut coder cette valeur sur un nombre **entier** d'octets.

1. Sur combien de bits, au minimum, peut-on coder toutes les valeurs entières de 0 à 500 ?
2. En déduire le nombre **d'octets** nécessaires pour coder cette valeur (on rappelle qu'un octet vaut 8 bits, et qu'il faut un nombre entier d'octets).
3. Avec ce nombre d'octets, quel est en réalité le plus grand entier que l'on pourrait coder ? Combien de valeurs sont ainsi « gaspillées », c'est-à-dire jamais utilisées par ce capteur ?

Correction de l'exercice 5

1. Il faut que $2^m - 1 \geqslant 500$, soit $2^m \geqslant 501$. Or $2^8 = 256 < 501$ et $2^9 = 512 \geqslant 501$: il faut donc au minimum **9 bits**.
2. Un octet ne suffit pas (8 bits $<$ 9 bits nécessaires) : il faut donc **2 octets** (16 bits), puisqu'on ne peut coder que sur un nombre entier d'octets.
3. Avec 2 octets (16 bits), on pourrait coder tous les entiers de 0 à $2^{16} - 1 = 65\,535$. Puisque le capteur n'utilise que les valeurs de 0 à 500, il y a $65\,535 - 500 = 65\,035$ valeurs qui ne seront jamais utilisées : la quasi-totalité de la capacité disponible est donc « gaspillée », mais c'est inévitable puisqu'on ne peut coder que sur un nombre

entier d'octets.

Exercice 6: Défi – Un nombre étrange

On considère le nombre entier $n = 187$.

1. Écrire n en base 2, puis en base 16.
2. Un élève affirme : « l'écriture hexadécimale de n ne contient que des chiffres qui existent aussi en décimal (pas de lettre) ». A-t-il raison ?
3. Trouver, par tâtonnement ou par réflexion, un autre entier naturel dont l'écriture hexadécimale ne contient elle non plus aucune lettre parmi A, B, C, D, E, F. Un tel nombre est-il rare, ou au contraire trouve-t-on facilement des exemples ?

Correction de l'exercice 6

1. $187 = 128 + 32 + 16 + 8 + 2 + 1 = 2^7 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0$, donc $187 = (1011\ 1011)_2$. En regroupant par tranches de 4 bits : $1011 \to B$ et $1011 \to B$, donc $187 = (BB)_{16}$.
2. Non, il a tort : $(BB)_{16}$ contient deux fois la lettre B, ce n'est donc pas un chiffre décimal.
3. De nombreux exemples conviennent, par exemple $18 = (12)_{16}$, $9 = (9)_{16}$, ou encore $100 = (64)_{16}$: de tels nombres ne sont pas rares du tout. Il suffit que chacun des « paquets de 4 bits » de l'écriture binaire du nombre corresponde à une valeur entre 0 et 9 (donc que chaque tranche de 4 bits soit comprise entre 0000 et 1001) pour qu'aucune lettre n'apparaisse.