Exercices

Doc. 4,6

Objectif

S'entraîner sur toutes les représentations des nombres étudiées dans le chapitre, en choisissant soi-même la méthode adaptée à chaque situation.

Les exercices mélangent volontairement les notions des Doc. 4,1 à 4,5. Avant de calculer, il faut chaque fois se demander : s'agit-il d'un entier positif, d'un entier relatif ou d'un nombre à virgule ? Sur combien de bits ? Les exercices 1 à 5 se font sans ordinateur ; la console sert ensuite à vérifier les réponses.

§1. Exercice 1 : quelle méthode ?

Q1. Pour chaque demande, indiquer la méthode utilisée, puis répondre.

1. Écrire 200 en base 2 sur 8 bits.
2. Quel entier relatif l'octet 11001000 représente-t-il en complément à 2 ?
3. Combien de bits faut-il pour écrire 1000 ?
4. Écrire $(7\text{F})_{16}$ en base 2, puis en base 10.
5. Écrire 0,375 en base 2.

Réponse :

1. Divisions successives par 2 : les restes lus de bas en haut donnent $200 = (11001000)_2$. L'écriture tient exactement sur 8 bits.
2. L'octet commence par 1 : il est négatif. Complément à 2 : inversé 00110111, plus 1 : 00111000 $= 56$. Il représente -56. On peut aussi calculer $200 - 256 = -56$.

   Les réponses 1 et 2 montrent que le même octet 11001000 représente 200 ou -56 selon la convention choisie.
3. Encadrement par des puissances de 2 : $2^9 = 512 \leq 1000 < 1024 = 2^{10}$, il faut 10 bits.
4. Paquets de 4 bits : 7 donne 0111 et F donne 1111, donc $(7\text{F})_{16} = (01111111)_2$. En base 10 : $7 \times 16 + 15 = 127$, le plus grand entier positif d'un octet en complément à 2.
5. Multiplications successives par 2 : $0{,}375 \times 2 = 0{,}75$ (0), $0{,}75 \times 2 = 1{,}5$ (1), $0{,}5 \times 2 = 1$ (1). Donc $0{,}375 = (0{,}011)_2$, écriture finie car $0{,}375 = \frac{3}{8}$.

§2. Exercice 2 : des couleurs en hexadécimal

En HTML, la couleur `#2E8B57` est formée de trois nombres écrits en base 16 sur deux chiffres chacun : `2E` pour le rouge, `8B` pour le vert et `57` pour le bleu.

Q2.

1. Combien de bits occupe chacun des trois nombres ? Combien de valeurs différentes peut-il prendre ?
2. Convertir les trois nombres en base 10.
3. Combien de couleurs différentes peut-on écrire de cette façon ? Donner le résultat sous la forme d'une puissance de 2.

Réponse :

1. Deux chiffres hexadécimaux correspondent à $2 \times 4 = 8$ bits, c'est-à-dire un octet : $2^8 = 256$ valeurs, de 0 à 255.

2. $(2E)_{16} = 2 \times 16 + 14 = 46$, $(8B)_{16} = 8 \times 16 + 11 = 139$ et $(57)_{16} = 5 \times 16 + 7 = 87$. C'est un vert assez sombre.
3. Une couleur occupe $3 \times 8 = 24$ bits : on peut écrire $2^{24} = 16\,777\,216$ couleurs.

§3. Exercice 3 : vrai ou faux ?

Q3. Dire si chaque affirmation est vraie ou fausse, et justifier en une phrase, avec un exemple ou un calcul.

1. La somme de deux entiers positifs écrits sur 16 bits tient toujours sur 16 bits.
2. Le produit de deux entiers positifs écrits sur un octet tient toujours sur deux octets.
3. Sur 8 bits, l'opposé de chaque entier relatif peut lui aussi s'écrire sur 8 bits.
4. En Python, `0.5 + 0.5 == 1` vaut `True`.
5. Ajouter un 0 à droite de l'écriture binaire d'un entier positif le multiplie par 10.

Réponse :

1. Faux : la somme tient sur 17 bits. Par exemple, $65\,535 + 1 = 65\,536 = 2^{16}$, qui demande 17 bits.
2. Vrai : le produit de deux entiers de 8 bits tient sur $8 + 8 = 16$ bits. Le plus grand produit est $255 \times 255 = 65\,025 < 65\,536 = 2^{16}$.
3. Faux : sur 8 bits, on écrit les entiers de -128 à 127. L'opposé de -128 est 128, qui ne s'écrit pas sur 8 bits.
4. Vrai : $0{,}5 = (0{,}1)_2$ s'écrit exactement en base 2, et le calcul est exact. Cela ne signifie pas qu'on peut tester l'égalité de deux flottants en général.
5. Faux : ajouter un 0 à droite multiplie par 2 en base 2. Par exemple, $(101)_2 = 5$ et $(1010)_2 = 10$.

§4. Exercice 4 : un dépassement de capacité

Un programme range des entiers relatifs sur 8 bits, en complément à 2. Il calcule $100 + 50$.

Q4.

1. Écrire 100 et 50 sur 8 bits, puis poser leur addition.
2. Quel entier relatif l'octet obtenu représente-t-il ?
3. Expliquer pourquoi le résultat est faux, alors qu'aucune retenue n'a été perdue.

Coup de pouce

Quel est le plus grand entier relatif que l'on peut écrire sur 8 bits ?

Réponse :

1. $100 = (01100100)_2$ et $50 = (00110010)_2$. Leur somme est $(10010110)_2$.
2. L'octet commence par 1 : il est négatif. $150 - 256 = -106$. La machine affirme que $100 + 50 = -106$.
3. Le résultat exact, 150, est plus grand que 127, le plus grand entier relatif écrit sur 8 bits. Il ne peut pas être représenté : c'est un dépassement de capacité. Il ne se manifeste pas ici par une retenue perdue, mais par un bit de poids fort qui passe à 1, ce qui change le signe du résultat. L'addition de deux nombres positifs qui donne un résultat négatif est le signe d'un dépassement.

§5. Exercice 5 : tables de vérité

Q5.

1. Dresser la table de vérité de `a and (b or c)`, qui dépend de trois variables. Combien de lignes faut-il ?
2. Dresser celle de `(a and b) or (a and c)`. Que conclure ?
3. Écrire le programme Python qui affiche la première table, et vérifier la réponse.

Réponse :

1. Avec trois variables, il y a $2^3 = 8$ combinaisons, donc 8 lignes :
2.

a	b	c	`a and (b or c)`	`(a and b) or (a and c)`
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

   Les deux colonnes sont identiques : les expressions sont équivalentes. On peut « développer » `and` sur `or`, comme on développe $a \times (b + c)$ en calcul.

3. Python
```
for a in (False, True):
    for b in (False, True):
        for c in (False, True):
            print(int(a), int(b), int(c), int(a and (b or c)))
```

§6. Exercice 6 : programmer

Q6. Le **poids** d'un entier positif est le nombre de chiffres 1 de son écriture binaire. Écrire la fonction `poids` sans utiliser `bin`, en s'inspirant de la méthode des divisions successives :

Python
```
def poids(n: int) -> int:
    """
    Retourne le nombre de chiffres 1 de l'écriture en base 2
    de l'entier positif n.

    >>> poids(13)
    3
    >>> poids(255)
    8
    >>> poids(0)
    0
    """
```

Réponse :

Python
```
def poids(n: int) -> int:
    compteur = 0
    while n > 0:
        compteur = compteur + n % 2
```

```
        n = n // 2
    return compteur
```

Chaque reste `n % 2` est un chiffre de l'écriture binaire, égal à 0 ou à 1 : en les additionnant, on compte les 1. Contrairement à `decompose`, il n'est pas nécessaire de construire la chaîne.

Q7. On veut convertir une écriture hexadécimale en entier, sans utiliser `int(..., 16)`.

1. Écrire la fonction `valeur_chiffre` :

```python
CHIFFRES = "0123456789ABCDEF"


def valeur_chiffre(c: str) -> int:
    """
    Retourne la valeur du chiffre hexadécimal c,
    c'est-à-dire son indice dans CHIFFRES.

    >>> valeur_chiffre("7")
    7
    >>> valeur_chiffre("B")
    11
    """
```

1. En s'inspirant de la fonction `recompose` du Doc. 4,1, écrire la fonction `hexa_vers_decimal` :

```python
def hexa_vers_decimal(hexa: str) -> int:
    """
    Retourne l'entier dont hexa est l'écriture en base 16.

    >>> hexa_vers_decimal("2E")
    46
    >>> hexa_vers_decimal("7E3")
    2019
    """
```

Coup de pouce

1. Parcourir les indices de `CHIFFRES` et retourner l'indice dès que le caractère correspond.
2. Dans `recompose`, on multipliait par 2 à chaque nouveau chiffre. Par combien faut-il multiplier en base 16 ?

Réponse :

```python
CHIFFRES = "0123456789ABCDEF"


def valeur_chiffre(c: str) -> int:
```

```
    for i in range(len(CHIFFRES)):
        if CHIFFRES[i] == c:
            return i


def hexa_vers_decimal(hexa: str) -> int:
    n = 0
    for c in hexa:
        n = 16 * n + valeur_chiffre(c)
    return n
```

Pour `"7E3"` : n vaut successivement 7, puis $16 \times 7 + 14 = 126$, puis $16 \times 126 + 3 = 2019$.

§7. Exercice 7 : une moyenne qui ne tombe pas juste

Un élève a écrit le programme suivant :

```python
moyenne = (12.3 + 11.9 + 12.1) / 3
if moyenne == 12.1:
    print("La moyenne vaut 12,1")
else:
    print("La moyenne ne vaut pas 12,1")
```

Q8.
1. Calculer la moyenne à la main. Quel message l'élève attend-il ?
2. Exécuter le programme. Quel message s'affiche ? Expliquer.
3. Corriger le programme.

Réponse :
1. $12{,}3 + 11{,}9 + 12{,}1 = 36{,}3$ et $\frac{36{,}3}{3} = 12{,}1$: il attend « La moyenne vaut 12,1 ».
2. Le programme affiche « La moyenne ne vaut pas 12,1 ». `moyenne` vaut `12.100000000000001` : les nombres 12,3, 11,9 et 12,1 n'ont pas d'écriture binaire finie, et les erreurs d'arrondi des calculs rendent le résultat légèrement différent du flottant `12.1`.
3. On compare l'écart à une petite valeur au lieu de tester l'égalité :

```python
moyenne = (12.3 + 11.9 + 12.1) / 3
if abs(moyenne - 12.1) < 1e-9:
    print("La moyenne vaut 12,1")
else:
    print("La moyenne ne vaut pas 12,1")
```

§8. Bilan

À l'issue de cette activité, je sais…

- identifier, devant une écriture binaire, la convention à utiliser (entier positif, complément à 2, nombre à virgule) et le nombre de bits ;

- convertir un entier entre les bases 2, 10 et 16, et un nombre à virgule simple en base 2 ;
- prévoir la taille d'une somme ou d'un produit, et reconnaître un dépassement de capacité, y compris en complément à 2 ;
- dresser la table de vérité d'une expression de trois variables et comparer deux expressions ;
- programmer des calculs sur l'écriture binaire ou hexadécimale d'un entier ;
- corriger un programme qui teste l'égalité de deux flottants.