# Correction du DS 1

Informatique pour tous, première année

Julien Reichert

## Exercice 1

Exemple : 338 jours. Convertissons en base 5.

Première méthode : $338 = 67 \times 5 + 3$ ; $67 = 13 \times 5 + 2$ ; $13 = 2 \times 5 + 3$ ; 2 est le dernier reste. La réponse est donc $\overline{2323}^5$ en reprenant les restes de droite à gauche.

Deuxième méthode : $338 = 2 \times 5^3 + 88$ ; $88 = 3 \times 5^2 + 13$ ; $13 = 2 \times 5 + 3$ ; 3 est le chiffre des unités. La réponse est la même en écrivant les coefficients de gauche à droite.

Additionnons en base 5 : $\overline{2323}^5 + \overline{2111}^5 = \overline{4434}^5$.

Pour convertir le résultat en binaire, on peut faire la division directement depuis la base 5, ce qui ne sera pas détaillé dans ce fichier, ou convertir d'abord en base 10, à ceci près que le nombre à convertir est plus grand que $\overline{2111}^5$ (qu'on convertira en 281) qu'on additionnera à 338 pour obtenir 619.

Au passage, dans ce cas précis la conversion de notre nombre était pratique : en additionnant 6 on obtient $\overline{10000}^5$ soit $5^4 = 625$.

La conversion en binaire se fait à l'aide d'une des deux méthodes ci-dessus, on obtient alors ici 1001101011.

## Exercice 2

Sur 32 bits, le plus grand nombre (qui s'avère être entier) exprimable s'écrit en virgule flottante `0 11111110 11111111111111111111111`, ce qui correspond au nombre $2^{127} \times (1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{23}})$, où le 1 est le nombre retiré de l'écriture de la mantisse. Le nombre peut donc également s'écrire $\sum_{i=104}^{127} 2^i$. Bien entendu, l'opposé est également exprimable en modifiant le bit de signe.

Quant à la caractérisation des entiers représentables dans l'intervalle, il suffit que l'écriture en virgule flottante ne nécessite pas d'arrondi, c'est-à-dire que le premier et le dernier `1` sont éloignés d'au plus `23` positions.

Par exemple, $2^{23} + 1$ est représentable, mais pas $2^{24} + 1$.

## Exercice 3

On écrit le programme en tant que fonction, son corps (sans l'indentation) pouvant être accepté comme un programme fonctionnant en supposant les variables `l` et `m` affectées.

```
def somme_liste(l):
    s = 0
    for x in l:
        s = s + x
    return s

def somme_matrice(m):
    s = 0
    for ligne in m:
        for x in ligne:
            s = s + x
    return s
```

Autre possibilité, en accédant aux indices des listes :

```
def somme_liste(l):
    s = 0
    for i in range(len(l)):
        s = s + l[i]
    return s

def somme_matrice(m):
    s = 0
    for i in range(len(m)):
        for j in range(len(m[i])):
            s = s + m[i][j]
    return s
```

Pour terminer, des fonctions prédéfinies permettent de raccourcir le code :

```
def somme_liste(l):
    return sum(l)

def somme_matrice(m):
    return sum([sum(ligne) for ligne in m])
```

## Exercice 4

La valeur de départ de la variable `x` est `0`, celle de `y` est `2`.

La première boucle est exécutée `42` fois, et la boucle intérieure est effectuée `i` fois, où `i` est le numéro (entre `0` et `41`) du tour de boucle. Cette boucle intérieure est donc effectuée `0 + 1 + ...+ 41`, soit `41` $\times$ `21 = 861` fois. À chaque tour, on ajoute `1` à la variable `x`, et `y` est remplacé par `2` divisés par son ancienne valeur, ce qui veut dire que ces valeurs alternent entre `2` et `1`, à ceci près que dès le premier tour la variable `y` devient un flottant (en Python3, du moins).

Ainsi, la première partie du programme se termine avec `x` valant `861` et `y` valant `1.0` (d'après la parité du nombre de tours).

La deuxième partie du programme est répétée `42` fois, où `x` est dans un premier temps augmenté de `y` à chaque tour, donc d'un au premier, ce qui lui donne la valeur `862.0` (en effet, l'addition d'un flottant transforme `x` en flottant lui-même) au premier tour ; dans un second temps, `y` devient `0` si `x` est pair, ce qui est le cas après le premier tour. Après cela, les `41` derniers tours n'ont pas d'effet, et la valeur finale de `x` est `862.0`.

On remarquera que sans connaître le nombre de tours dans la première boucle, on pouvait remarquer qu'il y avait quatre comportements possibles dans la deuxième suivant la parité de `x` et `y` :

- `x` et `y` impairs (donc `y` valant `1.0`) : `y` devient `0` après un tour et `x` est augmenté de `1.0` ;
- `x` pair et `y` impair (donc `y` valant `1.0`) : `y` devient `0` après deux tours et `x` est augmenté de `2.0` ;
- `x` impair et `y` pair (donc `y` valant `2.0`) : `y` ne devient jamais `0` et `x` est augmenté de `84.0` ;
- `x` et `y` pairs (donc `y` valant `2.0`) : `y` devient `0` après un tour et `x` est augmenté de `2.0` ;

En pratique, on peut prouver (invariant de boucle, vu plus tard dans l'année) que `x` et `y` sont toujours de même signe, ce qui exclut deux des cas.