Correction du DS 1

Julien Reichert

Exercice 1

Pour simplifier les calculs, on va échanger l'ordre des facteurs, afin d'avoir deux couples de lignes identiques.

				C	8	8	1
$\times$				B	A	B	A
			7	$D^{\cancel{7}}$	$5^{\cancel{7}}$	0	A
		8	$9^{\cancel{7}}$	$D^{\cancel{7}}$	8	B	0
	7	D	5	0	A	0	0
8	9	D	8	B	0	0	0
$9^{\cancel{1}}$	$2^{\cancel{2}}$	$3^{\cancel{1}}$	$F^{\cancel{2}}$	$6^{\cancel{1}}$	7	B	A

On notera que pour passer de la première à la deuxième des quatre lignes intermédiaires, on peut simplement additionner $\overline{C881}^{16}$ à la première ligne, vu que multiplier une valeur x par B revient à multiplier par x par A et ajouter x.

Exercice 2

Pour rappel, l'écriture en virgule flottante sur 16 bits utilise un bit de signe, puis 5 bits d'exposant et finalement 10 bits de mantisse.

Ici, plutôt que de procéder à des multiplications par deux comme dans la méthode du cours, une technique plus simple peut s'appliquer, dans la mesure où le dénominateur est une puissance de 2, en l'occurrence 2^{10}. On convertit alors le nombre 2019 en binaire, ce qui donne $\overline{11111100011}^2$ (pour accélérer le processus, $2019 = 2^{11} - 1 - 28$, et on sait écrire une puissance de deux moins un, il reste à écrire sans trop de peine 28 en binaire et le retrancher).

Par la suite, la fraction $\frac{2019}{1024}$ s'obtient en décalant la virgule de dix crans vers la gauche, et pour la sécurité on vérifie que le résultat est bien entre 1 et 2 : $\overline{1,1111100011}^2$. La puissance de 2 dans l'écriture scientifique binaire est donc zéro, qui s'encode en tant que 01111 par convention. Comme le 1 qui précède la virgule est implicite, on recopie simplement les dix bits suivants dans la mantisse, ce qui donne une écriture exacte et ne nécessite pas que la question de l'arrondi se pose. Reste à reporter un 0 pour le signe.

La représentation finale est alors 0 01111 11111100011.

Exercice 3

Pour obtenir la liste des diviseurs, puisque l'ordre n'est pas important (on s'en rend compte au moment d'utiliser la fonction), une méthode accélérant grandement le travail revient à s'arrêter à la racine du nombre et à ajouter les diviseurs par couples. Pour la rigueur, il ne faudrait pas mettre deux fois la racine du nombre s'il s'agit d'un entier, mais ce ne serait pas grave car le nombre serait de toute façon exclu et le temps de calcul n'en serait pas trop impacté.

D'éventuels problèmes d'arrondi seront ignorés sur les copies, mais il est bon de prendre le réflexe de tout faire pour ne pas leur donner l'occasion d'apparaître.

```
import math

def diviseurs(n):
    liste = []
    rac = math.sqrt(n)
    for i in range(2, math.ceil(rac)): # exclut la racine, traitée après
        if n % i == 0:
            liste.append(i)
            liste.append(n//i) # double barre sinon moche
    if int(rac) ** 2 == n: # plus sûr que rac == int(rac)
        liste.append(int(rac)) # int sinon moche
    # d'ailleurs on espère que la racine de k² n'est jamais k moins epsilon pour Python...
    return liste

def carreparfait(d):
    rac = math.sqrt(n)
    return int(rac)**2 == n # plus joli que if avec return booléen
# On note que le code est redondant avec ma version de diviseurs.
# Une idée d'amélioration : diviseurs retourne en plus l'information de la fonction carreparfait.

def LC(n):
    liste = diviseurs(n)
    if liste == []: # n est premier
        return False
    for d in liste:
        if carreparfait(d) or (diviseurs(d) == [] and (n+1) % (d+1) != 0): # ne pas oublier de parenthéser
            return False
    return True
```

Exercice 4

```
def couples(l, ll):
    reponse = []
    for i in range(len(l)):
        for j in range(len(ll)):
            if l[i] == ll[j]:
                reponse.append((i, j))
    return reponse
```

Exercice 5

```
def supprimezone(l, x, debut, fin):
    reponse = l[:debut] # gagner du temps
    i = debut
    while i < fin and l[i] != x:
        reponse.append(l[i])
        i += 1
    if i < fin:
        i += 1 # un élément à ignorer
    return reponse + l[i:] # et le reste est transféré
```