DS d'informatique : corrigé

PTSI Lycée Eiffel

17 novembre 2017

Exercice 1

1. En base 10, on convertit $1101001 = 2^6 + 2^5 + 2^3 + 2^0 = 64 + 32 + 8 + 1 = 105$.
2. Par divisions euclidiennes successives :

352	|	3									
1	|	117	|	3							
		0	|	39	|	3					
				0	|	13	|	3			
						1	|	4	|	3	
								1	|	1	| 3
											1 | 0

   On peut donc écrire, en base 3, que $352 = 111001$.
3. La distinction principale est entre la mémoire vive (RAM ou Random Access Memory, dont l(accès est rapide et qui permet de stocker des données temporaires pendant l'exécution des programmes) et la mémoire morte (ROM ou Read Only Memory, dont l'accès est environ 10 fois plus lent que la RAM, et qui sert à stocker les programmes ou fichiers à long terme dans l'ordinateur). La RAM est effacée quand on éteint l'ordinateur, contrairement à la ROM.
4. Tous les périphériques sont des moyens de communication entre l'utilisateur humain et la machine. Une périphérique d'entrée permet à l'utilisateur de transmettre des informations à la machine (exemples : souris, clavier, micro), un périphérique de sortie transmet des informations de la machine vers l'utilisateur (exemples : écran, enceintes).

Exercice 2

1. Il s'agit des commandes n%p (pour le reste) et n//p (pour le quotient).
2. La très simple fonction suivante suffit :

```
def unite(n) :
    return (n%10)
```

3. Ce programme calcule le chiffre des dizaines de l'entier n. En effet, la division euclidienne par 10 revient simplement à supprimer le dernier chiffre de l'écriture binaire de n, puis le reste de la division par n donne le chiffre des unités du nombre (comme ci-dessus).
4. C'est quasiment la même chose :

```
def centaine(n) :
    a=n//100
    return (a%10)
```

5. Voilà un programme qui fera l'affaire :

```
def Armstrong(n) :
    a=unite(n)**3+dizaine(n)**3+centaine(n)**3
```

```
    if a==n :
        return True
    return False
```

6. liste=[i for i in range(1000) if Armstrong(i)]
   On obtient la liste suivante : [0,1,153,370,371,407]

Exercice 3

1. La liste étant de longueur 6, l'indice k va donc prendre les valeurs de 6 à 2 (en décroissant).
   - Lorsque $k = 6$, le plus grand élément de la liste est 9 qui est en position 1, on échange donc $L[1]$ et $L[5]$ pour obtenir $L = [5, 7, 2, 4, 1, 9]$
   - Lorsque $k = 5$, le plus grand élément parmi les cinq premiers de la liste est 7 qui est en position 1, on échange cette fois $L[1]$ et $L[4]$ pour obtenir $L = [5, 1, 2, 4, 7, 9]$.
   - Lorsque $k = 4$, le plus grand élément parmi les quatre premiers est 5 qui est en position 0, on échange $L[0]$ et $L[3]$ pour obtenir $L = [4, 1, 2, 5, 7, 9]$.
   - Lorsque $k = 3$, le plus grand des trois premiers éléments est 4 qui est en position 0, on échange $L[0]$ et $L[2]$ et on a $L = [2, 1, 4, 5, 7, 9]$.
   - Enfin, pour $k = 2$, le plus grand des deux premiers éléments est encore en position 0, donc on a un dernier échange à faire pour trouver $L = [1, 2, 4, 5, 7, 9]$.

   Il semblerait bien que cet algorithme trie la liste L (dans l'ordre croissant).
2. for i in range(n,1,-1)
3. On va recopier l'élément en position i dans la « case » $k - 1$ mais ce faisant on perd la valeur qui était en position $k - 1$ et la deuxième instruction ne sert à rien. On aura à la fin dans les deux cases la valeur qui était initialement en position i. On peut éviter cela à l'aide d'une affectation simultanée :

   ```
   L[i],L[k-1]=L[k-1],L[i]
   ```

   qui est très bien gérée par Python, ou plus classiquement en faisant intervenir une variable temporaire :

   ```
   t=L[k-1]
   L[k-1]=L[i]
   L[i]=L[k-1]
   ```

4. En s'inspirant de ce qu'on a vu en cours :

   ```
   def maxdebut(L,k) :
       if len(L)<k :
           return ('Espèce de boulet intersidéral!')
       m=L[0]
       z=0
       for i in range(1,k) :
           if L[i]>m :
               m=L[i]
               z=i
       return i
   ```

   L'énoncé suggérait d'écrire un programme qui renvoie la valeur du plus grand élément, mais c'est en fait l'indice du plus grand élément qui nous intéresse, c'est ce qui a été programmé ici.

5. Il n'y a quasiment rien à ajouter :

```
def ouaf(L) :
for k in range(n,1,-1) :
        i=maxdebut(L,k)
        L[i],L[k-1]=L[k-1],L[i]
return L
```

6. Lors de la première étape, on a besoin de $n-1$ comparaisons pour déterminer le maximum de la liste. Il n'en faut plus que $n-2$ lors de la deuxième étape (on ne tient plus en compte le dernier élément), puis $n-3$ lors de la suivante etc. On a donc au total $(n-1)+(n-2)+\cdots+1 = \dfrac{n(n-1)}{2}$ comparaisons, soit en gros $\dfrac{n^2}{2}$ comparaisons (il y a des algorithmes beaucoup plus efficaces).