Représentation des nombres flottants

Q1. Que représente le nombre :
1 10000000011 0001100000000000000000000000000000000000000000000000

En utilisant les notations du cours :
(a) $s = 1$ donc le signe est -;
(b) $n = (10000000011)_2 = 1024 + 2 + 1$ donc $e = n - 1023 = 4$;
(c) $m = (1,00011)_2$ (attention de ne pas oublier que seuls les chiffres après la virgule sont représentés et qu'il y a forcément un 1 avant) ;
Le nombre est donc $-1,00011_2 \times 2^4 = -(1 + 2^{-4} + 2^{-5}) \times 2^4 = 2^4 + 2^0 + 2^{-1} = 16 + 1 + 0,5 =$ **-17,5**

Q2. Quelle est la représentation du nombre 20,125 ?

On décompose le nombre en somme de puissance de 2.
$20,125 = (16+4+0,125) = 2^4+2^2+2^{-3} = 2^4\times(1+2^{-2}+2^{-7}) = 2^4\times(1,0100001)_2$
(a) le signe est + donc $s = 0$;
(b) L'exposant est $e = 4$ donc il sera codé par $n = 1023 + e = 1027 = (100\ 0000\ 0011)_2$;
(c) $m = 1,0100001000000\ldots_2$ et donc sera codé par 01000010000000 . . . (attention de ne pas oublier que seuls les chiffres après la virgule sont représentés et qu'il y a forcément un 1 avant) ;
La représentation est donc
0 100 0000 0011 0100001 0 . . . 0

Q3. Parmi les cas suivants, quels sont ceux pour lesquels l'opération d'addition se fait sans arrondis et pourquoi :

Cas n°	*1*	*2*	*3*	*4*	*5*	*6*
x	2^{25}	2^{25}	2^{25}	2^{25}	2^{-25}	2^{-25}
y	2^{8}	2^{-20}	2^{-26}	2^{-30}	2^{-65}	2^{-125}
arrondi	*non*	*non*	*non*	*oui*	*non*	*oui*

La méthode est toujours d'écrire le nombre en notation scientifique en factorisant par la plus grande puissance de deux et en regardant ensuite si la mantisse peut s'écrire avec moins de 52 chiffres après la virgule.

Mesures de houle

Q4. Nombre d'octets correspondant à 20 minutes d'enregistrement
8 caractères par lignes sur 8 bits = 8 octets par mesures
20 minutes à 2 Hz = 20*60*2 = 2400 mesures soit 19200 octets en 20 minutes

Q5. nombre d'octets contenus dans le fichier correspondant à la campagne de mesures
En 15 jours = 15 x 24 x 2 x 19 200 = 13 824 000 octets soit 13.8 Mo (13.13 Mio)
La carte mémoire est largement suffisante.

Q6. gain relatif d'espace mémoire
Un chiffre de moins donne 1 octet de moins par mesure soit 12.5% de moins (soit 1.8Mo de moins)
RQ : s'ils veulent sauver de la place, qu'il sauvegarde ça en binaire et pas en fichier texte ! Il gagnerait facilement en précision ET en place ET en temps d'écriture

Partie II. Analyse vague par vague

Q7. fonction **moyenne**

```
1 def moyenne(liste_niveaux):
2   somme = 0
3   for niveau in liste_niveaux:
4       somme += niveau
5   return somme / len(liste_niveaux)
```

Q8. Détermination de l'indice du premier PND.

```
1 def ind_premier_pzd(liste_niveaux):
2     moy = moyenne(liste_niveaux)
3     for i in range (len(liste_niveaux)-1):
4         if liste_niveaux[i] > moy and liste_niveaux[i+1]<moy:
5             return i
6     return -1
```

RQ : remarque importante, certains ont mis dans la boucle moyenne(liste_niveau). C'est une horreur car ça fait passer la complexité de n à n². Ici, puisque la moyenne ne change pas au cours de l'exécution du programme, il FAUT la "pré"-calculer pour ne pas recalculer à chaque fois la même chose.

Q9. Détermination de l'indice du dernier PND.

```
1 def ind_dernier_pzd(liste_niveaux):
2     moy = moyenne(liste_niveaux)
4     for i in range (1,len(liste_niveaux)-1):
6         if liste_niveaux[-i] < moy and liste_niveaux[-i-1]>moy :
8             return len(liste_niveaux)-1-i
9     return -2
```

RQ : attention, la dernière valeur d'une liste L a pour indice
len(L)-1 et non len(L). De plus, si vous faites un "i+1" il faut donc commencer à len(L)-2. On peut faire des boucles for "à l'envers", l'ordre des arguments est toujours 1er (inclus), dernier (exclus), pas.
Donc range(10,4,-2) -> [10,8,6]

Q10. Liste successeurs.

```
1  def construction_successeurs(liste_niveaux):
2      n = len(liste_niveaux)
3      successeurs = [ ]
4      moy = moyenne(liste_niveau)
5      for i in range(n - 1):
6          if  liste_niveaux[i] > moy and liste_niveaux[i+1]<moy:
7              successeurs.append(i+1)
8      return succeseurs
```

Q11. Liste de vagues. Ainsi **decompose_vagues([1, -1, -2, 2, -2, -1, 6, 4, -2, -5])** *(Noter que cette liste est de moyenne nulle) retournera* **[[-1, -2, 2], [-2, -1, 6, 4]]**.

```
1 def decompose_vagues(liste_niveaux):
2     successeurs = construction_successeurs(liste_niveaux)
3     vagues = []
4     for i in range(len(successeurs-1)):
5         vagues.append(liste_niveaux[successeurs[i] :successeurs[i+1]])
6     return vagues
```

Q12. Caractérisation des vagues

```
1 def max_liste (liste) :
2     max = liste[0]
3     for elt in liste :
4         if elt > max :
5             max = elt
6     return max
7
8 def proprietes(liste_niveaux):
9     vagues = decompose_vagues(liste_niveaux)
10    liste = []
11    for i in range(len(vagues)):
12        Hi=max_liste(vagues[i]) - min(vagues[i])
13        Ti=(len(vagues[i])-1) * 0.5       # fréquence de 2 Hz , -1 pour avoir nombre d'intervalles
14        liste.append([Hi, Ti])
15    return liste
```

Q13. Tri à bulle

```
1 def tri_a_bulle (liste) :
2     for i in range(len(liste)-1)
3         for j in range(len(liste) – i – 1) :
4             if liste[j] > liste[j+1] :
5                 liste[j] ,liste[j+1] = liste[j+1] ,liste[j]
6     return liste
```

Q14. Dichotomie

```
1 def  dichotomie ( x, liste ) :
2     m = len(liste)//2
3     if len(liste) == 0 :
4         return False
5     elif x == liste[m] :
6         return False
7     elif x > liste[m] :
8         return dichotomie ( x, liste [m+1 : len(liste)])
9     else :
10        return dichotomie ( x, liste [0 : m])
```

Un peu de récursivité

Q15. Qu'est-ce qui s'afficherait dans une console python si on écrit l'instruction **affiche(5)** *, justifier ?*

```
*****
****
***
**
*
```

La fonction teste la condition de base, puis affiche le nombre d'étoiles suivant la valeur de n et enfin réalise un appel récursif avec n-1, d'où affichage avec une étoile en moins ensuite…

Q16. Fonction **affiche2** *?*

```
1 def affiche2 (n) :
2     if n == 0 :
3         return
4     affiche2 (n-1)
5     print (n* "*")
```

Q17. Deux fonctions récursives

```
1 def jeune(n):
2     if n == 0 :
3         return 2
4     else :
5         return 0.4*jeune(n-1) + 3.3*adulte(n-1)
6
7 def adulte(n):
8     if n == 0 :
9         return 4
10    else :
11        return 0.5*jeune(n-1) + 0.7*adulte(n-1)
```