CCINP Informatique PSI 2021 — Corrigé

Ce corrigé est proposé par William Aufort (professeur en CPGE) ; il a été relu par Cyril Ravat (professeur en CPGE) et Benjamin Monmege (enseignant-chercheur à l'université).

Ce sujet aborde différents aspects informatiques des montres multisport. Ces objets connectés sont devenus très populaires dans le monde du sport amateur et professionnel, notamment parce qu'ils permettent, à travers l'enregistrement de mesures physiques et physiologiques, de mesurer les performances des sportifs. Les trois parties du sujet sont indépendantes.

- La partie I est consacrée à l'acquisition et au traitement des données issues du module GPS d'une montre multisport. Celles-ci sont organisées sous la forme de trames qui suivent une norme décrite en annexe de l'énoncé. On implémente dans cette partie des fonctions permettant l'extraction et la vérification des trames, puis d'en isoler les données relatives à la localisation du coureur. Ces questions de programmation portent principalement sur les manipulations de chaînes de caractères (extraction et conversion en valeur numérique).
- La partie II se concentre sur l'acquisition du rythme cardiaque par une montre, à partir d'une analyse infrarouge de la saturation en oxygène du sang. Deux méthodes distinctes de traitement du signal sont employées, dont une utilise la transformée de Fourier discrète. La plupart des questions de cette partie demandent de compléter ou documenter un code déjà fourni dans l'énoncé.
- Enfin, la partie III introduit une base de données stockant les activités des utilisateurs de la montre. Plutôt courte (seulement trois questions) et facile, elle sert surtout de prétexte à écrire et interpréter des requêtes SQL.

La longueur du sujet est raisonnable et comparable à celles des années précédentes. Le programme de deuxième année n'est pas abordé et l'ingénierie numérique n'est évaluée que sur une seule question. Il s'agit donc avant tout de comprendre, commenter et écrire du code. Le candidat était également évalué sur sa capacité à absorber les nombreuses informations de l'énoncé et à réutiliser celles qui sont pertinentes. Il s'agit donc d'un bon sujet d'entraînement pour les étudiants de première année. On regrettera cependant l'absence de documentation de certaines fonctions spécifiques et non exigibles.

Indications

1 Pour interpréter le test de la ligne 5, il est nécessaire de lire la question 2.

2 La condition du `while` doit permettre de détecter les bons caractères de début de chaîne, par exemple à l'aide d'un slicing, ou l'arrivée à la fin de la liste. Attention à l'ordre dans lequel seront vérifiées ces deux conditions.

4 Il suffit de vérifier que la liste contient le bon nombre de chaînes et que celles-ci sont non vides.

5 Où se trouve la précision dans la liste `trame`, et sous quelle forme ?

6 Attention aux différents slicings présents dans ce code. Utiliser les annexes 2 et 3 pour comprendre la fonction `hex` et l'opérateur `^`, et les valeurs fournies par l'énoncé pour l'évaluation des `ord(ch)`.

7 Séparer les heures, minutes et secondes avec des slicings, puis effectuer les conversions nécessaires.

8 Utiliser le fait que les minutes occupent 8 caractères pour les isoler des degrés.

10 Les données fournies par l'énoncé permettent d'évaluer le nombre maximal de caractères sur une ligne. Ne pas oublier les caractères non numériques.

11 Comment peut-on représenter autrement une valeur numérique ?

12 La méthode des trapèzes permet d'obtenir l'approximation suivante

$$\int_t^{t+\mathrm{T}_e} (e(u) - s(u))\,\mathrm{d}u \simeq \mathrm{T}_e \times \frac{(e(t+\mathrm{T}_e) - s(t+\mathrm{T}_e) + e(t) - s(t))}{2}$$

13 Construire un tableau de la bonne taille et le remplir à l'aide d'une boucle en utilisant la question 12. On pourra poser $s(t_0) = e(t_0)$, non renseigné dans l'énoncé.

14 On continue d'exécuter la boucle tant que l'une des trois conditions données dans l'énoncé n'est pas respectée : attention aux opérateurs booléens utilisés.

15 Justifier notamment que la liste `pics` contient bien les indices des différents pics relevés dans la liste `signal` à l'issue de la boucle `while`.

16 Deux boucles imbriquées sont nécessaires dans cette fonction. Utiliser la constante `pi` et la fonction `exp` définies dans le module `numpy`.

19 Utiliser la fréquence d'échantillonnage pour relier l'intervalle de temps Δt aux indices `indDepart` et `indFin` du programme.

20 Utiliser le comportement du filtre aux instants T_d et T_f, ou l'expression de la fonction donnée dans l'énoncé.

23 Seule la table `activite` est nécessaire. Ne pas oublier les conversions.

24 Interpréter d'abord le contenu de la table `amis1`.

I. Acquisition et traitement des données GPS

1 L'instruction de la ligne 2 sépare les différentes lignes de la chaîne de caractères en entrée, en les stockant dans une liste nommée `chaineDecoupe`. L'instruction suivante permet de récupérer l'indice de la ligne correspondant à la trame GPGGA dans cette liste. Si cette trame est bien présente (test de la ligne 5), la ligne 6 permet de récupérer la ligne associée, à laquelle on enlève le premier symbole `$`. On en extrait ensuite les différentes informations dans une liste (ligne 7), sauf l'avant-dernière (ligne 8) qui correspond à un champ toujours vide d'après l'annexe 1.

> L'énoncé ne précisait pas à ce stade que le test de la ligne 5 permettait de vérifier la présence d'une trame GPGGA dans la liste. Cette information ne se trouve qu'à la question suivante expliquant le comportement de la fonction `indice_GPGGA` en cas d'absence d'une telle trame. D'où la nécessité de prendre un peu d'avance sur la lecture de l'énoncé (sans forcément le lire en intégralité) avant de se lancer.
>
> La méthode `pop`, utilisée à la ligne 8 pour la suppression du champ vide, n'était pas décrite dans l'annexe, probablement parce qu'elle figure implicitement au programme. En effet, lors de la représentation usuelle des piles par les listes Python, l'instruction `P.pop()` permet de dépiler un élément de `P` avec une complexité élémentaire. Néanmoins, cette méthode était utilisée ici avec un argument optionnel, correspondant à l'indice de l'élément à extraire, ce qui a pu perturber certains candidats.

2 L'énoncé impose ici l'utilisation d'une boucle `while`. Cette boucle doit s'arrêter dès que l'on arrive à la fin de la liste, ou que l'on a trouvé une chaîne commençant par les caractères `'$GPGGA'`. Ce dernier test est réalisé ici à l'aide d'un slicing récupérant les 6 premiers caractères de la chaîne à tester.

```
def indice_GPGGA(listeChaine):
    i = 0
    while i < len(listeChaine) and listeChaine[i][:6] != '$GPGGA':
        i = i + 1
    return i
```

> L'ordre des deux conditions dans cette boucle `while` est crucial. Lorsque Python évalue une expression booléenne, il le fait de façon paresseuse, c'est-à-dire que si la condition `i < len(listeChaine)` n'est pas vérifiée, la condition de droite n'est pas évaluée. Une évaluation de cette condition dans le cas où `i = len(listeChaine)` aurait déclenché une erreur car l'élément `listeChaine[i]` n'existe pas. Remarquons enfin que dans ce cas, la fonction renvoie bien `len(listeChaine)`, comme demandé dans la question.

3 La variable `donneesGPS` contient la chaîne reçue par le récepteur. On peut lui appliquer la fonction `recupererGPGGA` de la question 1 pour isoler la trame demandée.

```
listeGPGGABrute = recupererGPGGA(donneesGPS)
```

4 On teste dans un premier temps si la trame est bien constituée de 14 éléments, puis on vérifie le cas échéant si aucune donnée de la liste ne correspond à la chaîne de caractères vide.

```
def testPresence(trame):
    if len(trame) != 14:
        return False
```

```
    for chaine in trame:
        if chaine == '':
            return False
    return True
```

> L'énoncé est ambigu sur la nécessité de vérifier la première condition : la liste `trame` est supposée avoir la même forme que la liste donnée en exemple `listeGPGGABrute` contenant 14 éléments, mais la question demande tout de même de vérifier que les données sont bien présentes.

5 On remarque sur l'exemple donné dans l'énoncé que le coefficient de précision se trouve à l'indice 8 dans la liste des données de la trame, sous la forme d'une chaîne de caractères. Il suffit alors de comparer ce coefficient à la valeur 5, en prenant soin de le convertir en flottant au préalable.

```
def testPrecision(trame):
    return float(trame[8]) < 5
```

> Le coefficient de précision dont il est question ici permet de mesurer la qualité d'une mesure de position du récepteur GPS. Plus ce coefficient est grand, plus les satellites concernés dans cette détermination de position étaient proches lors de la mesure, ce qui augmente les incertitudes sur le relevé.

6 L'instruction `somme = trame[-1][1:]` permet de récupérer la dernière chaîne de la trame privée de son premier caractère, ce qui donne sur cet exemple `'A0'`.

Comme la liste `trame` ne contient que deux éléments, la boucle `for chaine in trame[:-1]` n'examine que la première chaîne de caractères de la trame, la seconde étant exclue à cause du slicing. Après initialisation, et à l'issue de la seconde boucle

```
for ch in chaine:
    cs = cs ^ ord(ch)
```

la variable `cs` contiendra, d'après l'annexe 3, le « ou exclusif » bit à bit des représentations entières des lettres de `chaine`, soit `ord('G') ^ ord('P') ^ ord('G')`. Cet opérateur est défini sur les bits par

$$\texttt{0 ^ 0 = 0}, \quad \texttt{0 ^ 1 = 1}, \quad \texttt{1 ^ 0 = 1} \quad \text{et} \quad \texttt{1 ^ 1 = 0}$$

On remarque que cette opération renvoie 1 s'il y a un nombre impair de bit égaux à 1, et renvoie 0 dans le cas contraire. À l'aide des valeurs fournies dans l'énoncé et de l'observation précédente, on obtient après calculs bit par bit :

```
cs = '0b01000111' ^ '0b01010000' ^ '0b01000111' = '0b01010000'
```

> On remarque qu'on retrouve la valeur donnée dans l'énoncé pour `ord('P')`, ce qui n'a rien d'une coïncidence. En effet, on peut montrer que l'opérateur « ou exclusif » est commutatif, associatif et vérifie `x ^ x = 0` pour tout `x` (bit ou entier), ce qui permet de simplifier le calcul précédent :
>
> `cs` = `ord('G') ^ ord('P') ^ ord('G')`
> = `(ord('G') ^ ord('G')) ^ ord('P')`
> = `0 ^ ord('P')`
> `cs` = `ord('P')`

D'après l'annexe 2, l'instruction `csHex = hex(cs)[2:]` convertit l'entier associé à l'écriture en binaire `'0b01010000'` de `cs`, soit $2^4 + 2^6 = 80$, en son expression hexadécimale sans le préfixe `'0x'` utilisé par Python, ce qui donne `'50'` car $80 = 16 \times 5$.