CONCOURS COMMUNS
POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE TSI

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites.

L'épreuve est composée de deux dossiers :
- un premier dossier de 20 pages contenant le sujet (pages numérotées de 1/20 à 16/20) et les annexes (pages numérotées de 17/20 à 20/20) ;
- un second dossier de 12 pages constituant le document réponse (DR) à rendre en fin d'épreuve (pages numérotées de DR – 1/12 à DR – 12/12).

Les consignes permettant de compléter le document réponse (DR) sont données à la page suivante.

Seul le document réponse est à rendre.

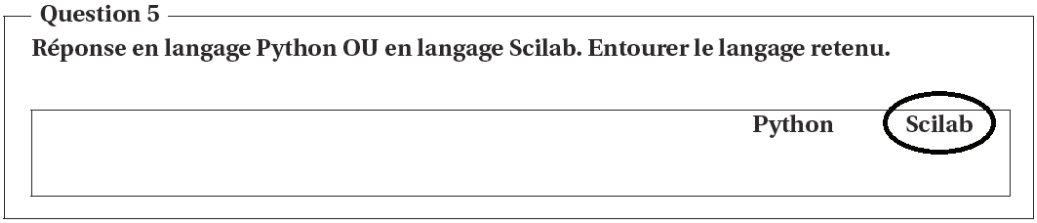
Remarques générales

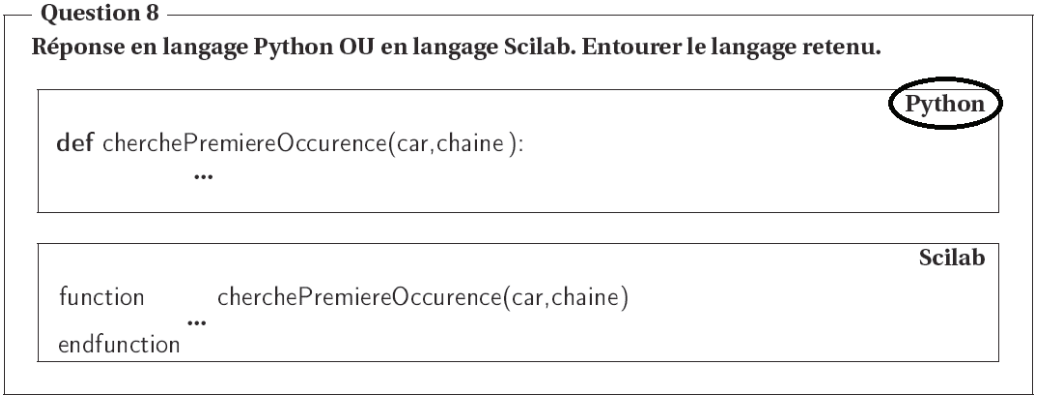
Les réponses aux questions sont à rédiger sur le document réponse (DR) et ne doivent pas dépasser les dimensions des cadres proposés.

Si la réponse attendue est spécifique à un langage, la réponse peut être proposée en langage Python ou en langage Scilab. Vous devez alors clairement entourer le langage retenu pour répondre à la question. Il est possible de changer de langage au cours de l'épreuve.

On distingue deux cas de figure sur les documents réponse : un premier cas où le cadre de réponse propose deux langages. Un second où il existe deux cadres spécifiques.

Les exemples proposés ci-dessous issus respectivement de la question 5 (dans laquelle le langage retenu pour la réponse serait le langage Scilab) et de la question 8 (dans laquelle le langage retenu pour la réponse serait le langage Pyhton) illustrent le principe d'indication du langage support de la réponse.





Quel que soit le langage choisi, les structures algorithmiques doivent être clairement identifiables par des indentations visibles ou par des barres droites entre le début et la fin de la structure comme proposé ci-dessous :

Si *(Condition)*
| **Alors**
| | *Instructions*
| **Sinon**
| | *Instructions*
Fin Si

ACQUISITION ET EXPLOITATION D'UN DEPLACEMENT A PARTIR D'UN RECEPTEUR GPS

Le sujet consiste en l'analyse et l'exploitation de données GPS (Global Positioning System) acquises par un récepteur équipant un véhicule. Ces dernières sont obtenues via un dispositif de réception installé à bord et fonctionnant à une fréquence allant de 1 à 4 Hertz. Chacune des mesures reçues contient (entre autre) des informations de positionnement (latitude, longitude) et de vitesse.

Le décodeur de trames inclus dans le récepteur GPS envoie les informations au calculateur embarqué sur un bus de communication suivant un protocole particulier. Dans le cadre de ce sujet, le protocole retenu est le protocole NMEA 0183 (National Marine Electronics Association), utilisant des trames de type GPRMC (Recommended minimum specific GPS/Transit data) et utilisable avec la plupart des récepteurs GPS. Le format des trames GPRMC est défini en annexe 1 (page 17).

Problématique

Comment afficher sur une carte le parcours d'un véhicule ainsi que sa vitesse moyenne à partir de la latitude, de la longitude et de la vitesse extraites de trames GPS émises par un satellite ?

1 Acquisition des trames NMEA via une liaison série

L'objectif de cette première partie est d'enregistrer les trames GPS reçues par le décodeur dans un fichier texte nommé tramesNMEA.txt. Ce fichier sera utilisé en temps différé pour afficher le parcours réalisé.

Le décodeur GPS reçoit des trames provenant des satellites à la fréquence de 1 Hertz. Elles sont ensuite transmises sur un port série dans le but d'être traitées par le calculateur embarqué. La figure 1 ci-dessous présente le principe de fonctionnement du système étudié.

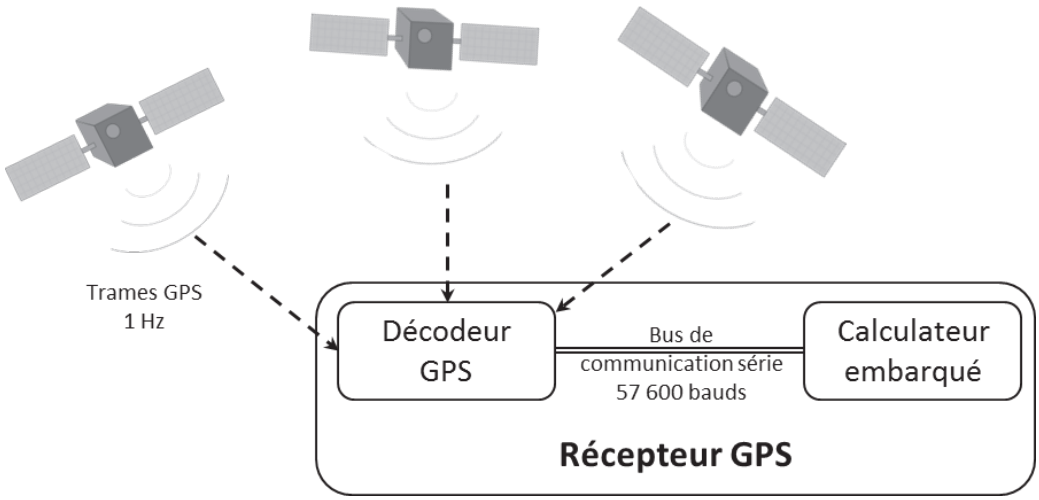


FIGURE 1 – Communication entre les différents composants du système

1.1 Liaison série

Le décodeur GPS envoie sur la liaison série de type RS-232C une trame GPRMC.

De manière générale, une liaison série permet de véhiculer, sur un nombre réduit de supports physiques, des informations élémentaires les unes à la suite des autres.

Dans le protocole RS-232C, pour chaque octet à transmettre, 1 bit de start, 8 bits de données (correspondant à l'octet à transmettre), puis 1 bit de stop sont envoyés sur le bus de communication. Ainsi, pour chacun des octets d'une trame GPRMC, 10 bits sont transmis. Le signal de communication est bivalent, chaque bit est codé par un signal électrique haut ou bas dont la durée dépend du débit choisi pour la transmission. Au débit de 57 600 bauds choisi ici, 57 600 bits sont donc transmis à chaque seconde.

Nous faisons l'hypothèse que la liaison est fiable, c'est-à-dire que l'on considère qu'il n'y a pas de panne du récepteur ou de déconnexion physique de la liaison série.

On considère une trame de 76 caractères codés en ASCII. On donne en annexe 1 (page 17) un exemple de trame et en annexe 2 (page 18) la table ASCII.

Question 1 *Au vu de la problématique du sujet, préciser combien d'octets sont nécessaires dans une trame GPS de type GPRMC pour encoder la latitude et la longitude avec leurs précisions Nord – Sud ou Est – Ouest et la vitesse en nœuds. – Dans le dénombrement les séparateurs de champs (virgules) ne seront pas comptabilisés. –*

Question 2

1. *Quelle est la durée de transmission d'une trame de 76 caractères sur le bus RS-232C ?*
2. *Le débit de transmission du bus série est-il suffisant pour traiter en temps réel les trames provenant des satellites ?*

1.2 Lecture des trames GPS

On souhaite utiliser une bibliothèque permettant la gestion des entrées/sorties sur une liaison série. Un extrait de la spécification de cette bibliothèque est donné ci-après en Python dans la table 1 (ci-dessous) et en Scilab dans la table 2 (page 5).

Python

```
def initSerie(port,bauds) :
    """
    Initialise un port série et retourne un
     identifiant de port série idport :
     Entrées :
        * port : int, entier >=0
        * bauds : int, débit en bauds du port
     Sortie :
        * idport : int, l'identifiant du port
    """
```

Python

```
def lireSerie(idport) :
    """
     Attend l'arrivée d'un octet sur le port série
     et le retourne sous la forme d'un entier
     (code ASCII).
     Entrée :
        * idport : int, identifiant de port série
     Sortie :
        * car : int, valeur de l'octet lu sur le
                    port série
    """
```

TABLE 1 – Extrait de la spécification d'une bibliothèque de gestion de liaison série en Python

Scilab

```
1 function [idport]= initSerie (port,bauds)
2 // Initialise un port serie et retourne un
3 // identifiant de port serie idport
4 // Entrées :
5 // * port : int, entier >=0
6 // * bauds : int , debit en bauds du port
7 // Sortie :
8 // * idport : int, l'identifiant du port
```

Scilab

```
1 function [car]= lireSerie (idport)
2 // Attend l'arrivée d'un octet sur le port série
3 // et le retourne sous la forme d'un entier
4 // (code ASCII).
5 // Entrée :
6 // * idport : int, identifiant de port série
7 // Sortie :
8 // * car : int, valeur de l'octet lu sur le
9 // port série
```

TABLE 2 – Extrait de la spécification d'une bibliothèque de gestion de liaison série en Scilab

Question 3 *Donner l'instruction permettant d'initialiser le port série numéroté « 0 » à une vitesse de 57 600 bauds. L'identifiant du port sera stocké dans une variable nommée* identifiant.

Question 4

1. *En utilisant les annexes 1 et 2 (pages 17 et 18), déterminer le dernier caractère de la trame. Préciser quel est son code ASCII sous forme décimale.*
2. *A partir des informations contenues dans l'annexe 3 (page 19), indiquer de quelle manière obtenir un caractère à partir de son code ASCII exprimé sous forme décimale.*

Définitions

On appelle **signature** d'une fonction le nom de la fonction ainsi que l'ensemble des paramètres reçus et éventuellement retournés par cette dernière. Cette signature doit être syntaxiquement correcte vis-à-vis du langage retenu (exemple : lignes 1 des tables 1 (page 4) et 2 (page 5) pour la fonction initSerie()).

On appelle **spécification** d'une fonction la description de l'objectif de la fonction ainsi que la description des paramètres d'entrée et de sortie. Pour les paramètres d'entrée, la description comprend également les contraintes liées à ces paramètres qui assurent le bon fonctionnement de la fonction (exemples : lignes 2 à 10 de la table 1 (page 4) pour la fonction initSerie() en langage Python et lignes 2 à 8 de la table 2 (page 5) pour la fonction initSerie() en langage Scilab).

La spécification apparaît en commentaire dans le début du corps d'une fonction juste après sa signature.

On donne dans la table 3 (page 6) l'algorithme en pseudo-code de la fonction lireTrame() permettant de lire et de retourner une trame à partir d'un port série préalablement ouvert.

On se propose d'écrire l'implémentation de cette fonction en langage Python ou en langage Scilab à partir des annexes 3, 4 ou 5 (page 19).

Question 5 *Suivant le langage retenu, traduire l'algorithme proposé table 3 en précisant bien la signature et la spécification de la fonction.*

Algorithme : lecture et affichage d'une trame

Fonction lireTrame(*Paramètre à définir* **)**

Entrée : *à définir*
Sortie : *à définir*
trame ← chaîne de caractères vide
lu ← 0
Tant que lu ≠ nombre entier correspondant au caractère de fin de trame en ASCII **faire**
lu ← lire octet série correspondant à un caractère
ajouter en fin de trame le caractère correspondant à lu
Fin Tant que
Afficher (trame)
Retourner (trame)

Fin Fonction lireTrame

TABLE 3 – Algorithme en pseudo-code de la fonction lireTrame()

1.3 Enregistrement dans un fichier

Nous souhaitons maintenant enregistrer toutes les trames envoyées par le récepteur GPS dans un fichier texte nommé tramesNMEA.txt situé dans le dossier courant. Le fichier texte sera ouvert en début de programme de sorte à être vidé s'il existait déjà auparavant. Ici, le récepteur GPS envoyant des trames en continu, le programme à réaliser n'a pas de fin et ne se terminera que lorsqu'il sera explicitement fermé par l'utilisateur. Pour implémenter ce comportement, on pourra utiliser l'expression « while (True) :» en Python ou « while (%T) » en Scilab qui mettent en place, dans chacun des langages, une boucle infinie.

Les annexes 4 et 5 (page 19) présentent des fonctions en langages Python et Scilab permettant de manipuler les fichiers.

Question 6 *En utilisant les fonctions précédentes, écrire dans le langage de votre choix le programme principal qui, après initialisation du port de la liaison série, permet de lire les trames sur le port série et d'enregistrer ces dernières, telles qu'elles sont reçues, dans le fichier tramesNMEA.txt, au format texte.*

2 Exploitation du fichier de trames

L'objectif de cette partie est d'extraire et d'enregistrer une trame dans une structure de données cohérente afin de pouvoir exploiter aisément les mesures.

On fait l'hypothèse que les trames transmises ne contiennent pas d'erreur.

2.1 Extraction d'une trame

La description d'une trame GPMRC, donnée en annexe 1 (page 17), indique que les valeurs des différentes informations de la trame sont comprises entre les caractères « $ » et «*».

Nous souhaitons disposer d'une fonction cherchePremiereOccurrence() susceptible de retourner la position de la première occurrence d'un caractère ASCII (exemple d'utilisation de la fonction : rechercher la position de la première occurrence de « $ » dans une trame).

On donne, dans la table 4, le corps de la fonction isCharInString() en langages Python et Scilab. Cette fonction permet de déterminer si un caractère particulier, passé en paramètre, est présent dans une chaîne de caractères, elle aussi passée en paramètre.

Python

```
1  def isCharInString(car,chaine):
2      """
3      Recherche si un caractère est dans une
4      chaîne de caractères.
5      Entrées :
6        * car : str, caractère
7        * chaine : str, chaîne de caractères
8      Sortie :
9        * un booléen : True si le caractère est
10       présent, False sinon
11     """
12     n = len(chaine)
13     i=0
14     while i<n :
15         if chaine[i]==car:
16             return True
17     return False
```

Scilab

```
1  function [bool] = isCharInString(car,chaine)
2  // Recherche si un caractère est dans une
3  // chaîne de caractères.
4  // Entrées :
5  // * car : str, caractère
6  // * chaine : str, chaîne de caractères
7  // Sortie :
8  // * un booléen : True si le caractère est
9  // présent, False sinon
10     n = length(chaine);
11     i=1;
12     bool = %F;
13     while i<=n
14         if (part(chaine,[i])==car) then
15             bool = %T
16             break
17         end
18     end
19 endfunction
```

TABLE 4 – Code de la fonction isCharInString() de recherche d'un caractère dans une chaîne de caractères en langages Python et Scilab

Question 7

1. *Préciser pour quelle raison cette première version de la fonction isCharInString(), proposée table 4 (ci-dessus), ne fonctionne pas lors des tests de validation.*
2. *Proposer une modification du corps de la fonction en précisant les numéros de lignes de code modifiées ou créées.*

Question 8 *En vous appuyant sur l'algorithme proposé table 4 (ci-dessus), compléter le corps de la fonction cherchePremiereOccurrence() qui retourne l'indice de la première occurrence d'un caractère passé en paramètre. Il convient de choisir et de justifier ce que la fonction retourne dans le cas où le caractère recherché n'est pas présent dans la chaîne.*

On cherche à concevoir l'algorithme de la fonction extraireTrame().

Cette fonction doit permettre d'extraire et de retourner, au format chaîne de caractères, le contenu d'une trame (nommée NMEA) situé strictement entre les caractères «$» et «*» en s'assurant au préalable que ces caractères sont bien présents dans la chaîne. La trame extraite se nomme NMEA_extraite.

A titre d'exemple, le contenu de la trame NMEA_extraite issue de la fonction extraireTrame(), basée sur la trame donnée en exemple dans l'annexe 1 (page 17) est de la forme :

GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A

On précise que G est en première position de la trame extraite, c'est-à-dire en position 0 en langage Python et en position 1 langage en Scilab.

L'annexe 6 (page 20) indique la manière dont sont indexées les chaînes de caractères en langage Python et en langage Scilab. Cette annexe précise aussi comment extraire des sous-chaînes de caractères dans une chaîne de caractères.

Question 9

1. *Indiquer quelles sont les conditions nécessaires et suffisantes que doit respecter une trame quelconque afin de pouvoir être extraite grâce à la fonction extraireTrame().*
2. *A partir des fonctions proposées dans cette partie, établir, sous forme algorithmique, une implémentation de la fonction extraireTrame() dans laquelle la chaîne d'entrée se nomme NMEA et la chaîne de sortie NMEA_extraite. Dans le cas où la chaîne ne peut pas être extraite, la fonction doit retourner une chaîne vide.*

2.2 Structure de données

Dans le cadre des traitements ultérieurs que nous souhaitons effectuer, il nous faut représenter une mesure GPS, que nous nommerons par la suite pointGPS, par sa latitude, sa longitude et sa vitesse.

Dans la trame GPS, les longitudes et latitudes sont données en degrés et minutes. Cette représentation ne facilitant pas les calculs, il est nécessaire de les convertir en minutes (1 degré étant égal à 60 minutes). Par ailleurs, on opte pour la convention suivante :

– pour la latitude, le signe positif est choisi pour le Nord ;
– pour la longitude, le signe positif est choisi pour l'Est.

Ainsi par exemple, si la trame contient la latitude «0314.2500» suivie de l'indicateur «S», la séquence «03» correspond à 3 degrés soit 180 minutes d'angle, la séquence «14.2500» correspond à 14,25 minutes d'angle et l'indicateur «S» implique que la latitude doit être précédée du signe moins. Ainsi, pour la latitude, on stockera -194.25 dans le pointGPS. Cette conversion est assurée par la fonction conversionLongLat2Min() dont la signature et les spécifications sont données dans la table 5 (ci-dessous).

Python

```
def conversionLongLat2Min(ll,orientation) :
    """
    Conversion de la la longitude ou de la latitude en minutes (nombre positif ou négatif)
    Entrées :
        * ll : str, latitude sous la forme ddmm.mmmm ou longitude sous la forme dddmm.mmmm
        * orientation : str, "N", "S", "E" ou "W".
    Sortie :
        * res : float : résultat en minutes d'angle
    """
```

Scilab

```
function [res]= conversionLongLat2Min(ll,orientation)
// Conversion de la la longitude ou de la latitude en minutes (nombre positif ou négatif)
// Entrées :
// * ll : str, latitude sous la forme ddmm.mmmm ou longitude sous la forme dddmm.mmmm
// * orientation : str, "N", "S", "E" ou "W".
// Sortie :
// * res : float, résultat en minutes d'angle
```

TABLE 5 – Signature et spécification de la conversion de la latitude ou de la longitude en nombre exploitable en langages Python et Scilab

Afin de stocker un pointGPS on utilise une liste qui présente la structure interne suivante :

pointGPS = [latitude, longitude, vitesse].

Cette liste sera renseignée par les latitudes, longitudes et les vitesses extraites des différentes trames envoyées par le décodeur et traitées par les fonctions précédentes.

Il existe deux principales méthodes pour extraire ces données, une première repose sur le comptage des séparateurs et peut s'adapter à des longueurs variables pour un même champ. La seconde, plus simple à mettre en œuvre, repose sur l'hypothèse que toutes les trames présentent exactement la même structure et que toutes les longueurs de champs restent identiques entre les différentes trames reçues.

Dans la suite du sujet, nous ferons l'hypothèse que toutes les longueurs de champs sont toujours identiques dans les différentes trames à traiter.

On rappelle qu'en appliquant la fonction extraireTrame() (définie dans la partie précédente) à la trame donnée en annexe 1 (page 17), on obtient :

GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A

Question 10

1. *A partir de la trame extraite issue de l'annexe 1 (page 17) et rappelée ci-dessus, ainsi que des précisions apportées sur l'annexe 6 (page 20), compléter le tableau donnant l'index de début et l'index de fin des informations nécessaires à l'extraction de la latitude (y compris l'orientation), de la longitude (y compris l'orientation) et de la vitesse.*
2. *Compléter la fonction creationPointGPS() dont la spécification et la signature sont données dans la table 6 (ci-dessous). On utilisera pour cela la fonction conversionLongLat2Min() ainsi que les informations données précédemment.*

Python

```
def creationPointGPS(NMEA_extraite) :
"""
Stocke les valeurs de latitude, longitude et vitesse dans une structure de type pointGPS
Entrée :
  * NMEA_extraite : chaîne de caractères, contenu de la trame (entre $ et *)
      (résultat de la fonction extraireTrame())
Sortie :
  * pointGPS : liste de la forme [latitude, longitude, vitesse]
"""
```

Scilab

```
function [pointGPS]=creationPointGPS(NMEA_extraite)
// Stocke les valeurs de latitude, longitude et vitesse dans une structure de type pointGPS
// Entrée :
// * NMEA_extraite : chaîne de caractères, contenu de la trame (entre $ et *)
// (résultat de la fonction extraireTrame())
// Sortie :
// * pointGPS : liste de la forme [latitude, longitude, vitesse]
```

TABLE 6 – Spécification de la fonction creationPointGPS() en Python et Scilab

2.3 Affichage du fichier

Nous disposons maintenant de toutes les fonctions qui permettent d'exploiter le fichier de trames créé dans la première partie (tramesNMEA.txt). Nous souhaitons à présent afficher successivement chacun des pointGPS présents dans le fichier sur une console texte.

Pour la gestion des fichiers, on pourra utiliser, suivant le langage cible, les fonctions close(), open(), readline() et write() dont les descriptions sont proposées en annexe 4 (page 19) ou les fonctions file(), read() et write() dont les descriptions sont proposées en annexe 5 (page 19). On précise qu'il n'est pas indispensable d'utiliser toutes les fonctions.

Question 11 *Le fichier tramesNMEA.txt contient les éléments correspondant aux pointsGPS. En utilisant le langage de votre choix, établir le programme principal qui affiche sur la console chacun de ces éléments. Chaque ligne de la console devra être de la forme suivante :*

```
latitude_Pt_1    longitude_Pt_1    vitesse_Pt_1
latitude_Pt_2    longitude_Pt_2    vitesse_Pt_2
```

Chaque espace correspond à une tabulation.

3 Structure avancée de données

L'objectif de cette partie est de stocker les points dans un arbre afin de pouvoir accéder aux pointsGPS ayant la plus grande vitesse de façon optimale.

Un tas binaire est une structure de données informatiques qui permet d'accéder au maximum (respectivement minimum) d'un ensemble de données en temps constant. On peut la représenter par un arbre binaire vérifiant deux contraintes :

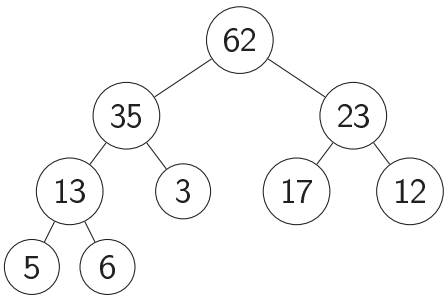
- c'est un arbre binaire parfait : tous les niveaux de l'arbre (excepté le niveau le plus bas) sont totalement remplis. Si le dernier n'est pas totalement rempli alors il doit l'être de gauche à droite ;
- c'est un tas : une clé est associée à chaque nœud de l'arbre. Cette dernière doit être supérieure ou égale aux clés de chacun de ses fils.

Ainsi, lorsque les clés sont des nombres (valeurs) et quand la relation d'ordre choisie est l'ordre naturel, on parle alors de tas-max (ou max-heap).

Un tas binaire étant un arbre binaire complet, on peut l'implémenter à l'aide d'un tableau tel que :

- la racine de l'arbre (niveau le plus haut) se trouve à l'index absolu 1 ;
- en considérant un nœud à l'index absolu i :
  - son fils gauche se trouve à l'index absolu $2i$;
  - son fils droit se trouve à l'index absolu $2i+1$;
- en considérant un nœud à l'index absolu $i > 1$:
  - son père se trouve à l'index absolu $i/2$, le symbole / désignant ici la division entière.

La figure 2 (page 11) illustre un arbre binaire (trié) et le tableau qu'on peut lui associer avec les index associés (index absolu, index en langage Python, index en langage Scilab).



Valeur dans l'arbre (clé)	62	35	23	13	3	17	12	5	6
Index absolu dans l'arbre	1	2	3	4	5	6	7	8	9
Index du tableau en Python	0	1	2	3	4	5	6	7	8
Index du tableau en Scilab	1	2	3	4	5	6	7	8	9

Le nœud ayant la valeur 62 est la racine de l'arbre (niveau le plus haut). Les nœuds ayant les clefs 5 et 6 sont au niveau le plus bas.

FIGURE 2 – Arbre binaire et son implémentation en tableau

Dans un tas binaire, on insère l'élément à la prochaine position libre (la position libre la plus à gauche possible sur le dernier niveau) puis on effectue l'opération suivante (nommée percolateUp()) pour rétablir si nécessaire la propriété d'ordre du tas binaire.

Le fonctionnement de cette fonction peut être appréhendé de la manière suivante : tant que l'élément n'est pas la racine de l'arbre et que la clé de l'élément est strictement supérieure à son père, on échange les positions entre l'élément et son père. L'exemple de la figure 3 (ci-dessous) montre l'insertion d'un élément ayant une clé de valeur 72.

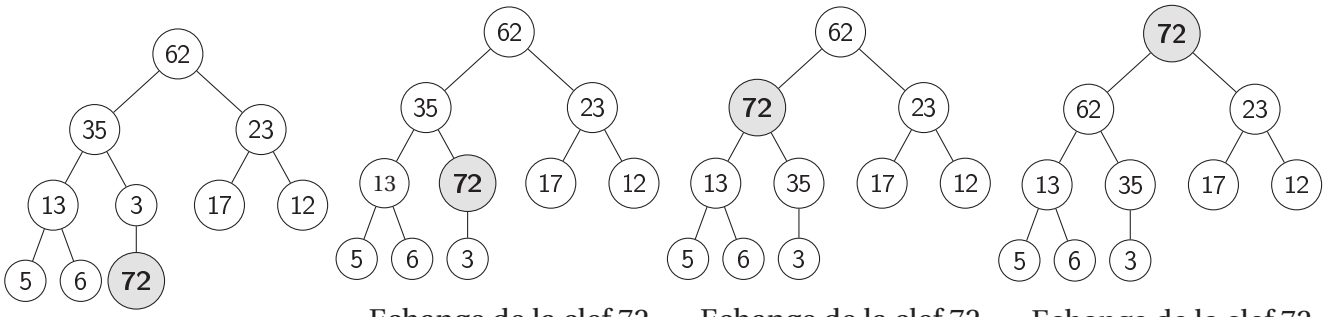


FIGURE 3 – Insertion d'un élément de clef 72

La table 7 (page 12) propose le code de la fonction percolateUp() en langages Python et Scilab.

On souhaite stocker les pointGPS dans un tas binaire. La liste (tas) sera donc de la forme suivante :

[[latitude_0, longitude_0, vitesse_0],[latitude_1, longitude_1, vitesse_1],...].

Quand un tas binaire est réalisé sur un ensemble structuré de variables comme c'est le cas sur les variables de type pointGPS, un des paramètres de cette variable structurée devient la clé. Les comparaisons présentes dans la fonction percolateUp() se font alors sur cette clé.

Python

```
1  def percolateUp (tas) :
2      index = len(tas)-1
3      val = tas[index]
4      while index>0 :
5          index_pere = int((index-1)/2)
6          pere = tas[index_pere]
7          if pere > val :
8              break
9          else :
10             tas[index] = pere
11             tas[index_pere] = val
12             index = index_pere
```

Scilab

```
1  function [tas]=percolateUp(tas)
2      index = size(tas);
3      val=tas(index);
4      index_pere = (index)/2;
5      while index_pere>=1
6          pere = tas(index_pere);
7          if pere>val then
8              break;
9          else
10             tas(index)=pere;
11             tas(index_pere)=val;
12             index=index_pere;
13             index_pere = (index)/2;
14         end
15     end
16 endfunction
```

TABLE 7 – Fonction percolateUp() en langages Python et Scilab

Question 12 *Modifier la fonction percolateUp() de façon à stocker des variables de type pointGPS dans les nœuds de l'arbre. Pour cela, on barrera la (ou les) ligne(s) modifiée(s) du document réponse et on écrira la modification en vis-à-vis (en précisant le (ou les) numéro(s) de ligne(s) modifiée(s)). Le paramètre vitesse, stocké dans un pointGPS (à la position 2 en langage Python et à la position 3 en langage Scilab), jouera le rôle de clé.*

Question 13 *Proposer une fonction insererPointGPS() qui agit sur le tas binaire passé en paramètre et qui permet d'insérer un point GPS, passé en paramètre, dans ce dernier.*

Question 14

1. *Proposer l'implémentation de la fonction extractMax() qui attend le tas passé en paramètre et qui permet d'extraire de ce dernier, sans le modifier, le pointGPS possédant la plus grande vitesse.*
2. *Donner la complexité de cette opération.*

4 Interface graphique

L'objectif de cette partie est d'extraire les points contenus dans le fichier tramesNMEA.txt et de les tracer sur un fond de carte. La couleur de chaque segment du tracé doit dépendre de la vitesse du véhicule.

Tout d'abord, nous allons étudier un principe de lissage de la vitesse afin d'y associer un code couleur et dans un second temps nous étudierons la mise au point du tracé de chaque segment sur le fond de carte sélectionné.

4.1 Etude du principe de calcul des vitesses lissées et association d'un code couleur à chaque vitesse lissée.

La vitesse d'un véhicule est soumise à des aléas fréquents. Elle peut varier fortement entre deux points successifs de mesure. Pour minimiser ces écarts, on souhaite réaliser un lissage sur 4

points. Pour cela, en un point considéré, on réalise la moyenne des vitesses entre les deux points mesurés précédemment, la vitesse du point en cours et la vitesse du point suivant.

Dans un premier temps, on considère que les vitesses sont stockées dans la liste présentée dans la table 8 (ci-dessous), dont on donne l'index des valeurs.

Valeur de vitesse	0.0	1.1	2.2	3.4	4.5	5.3	4.6	3.8	2.6	1.2
Index absolu dans le tableau	1	2	3	4	5	6	7	8	9	10
Index du tableau en langage Python	0	1	2	3	4	5	6	7	8	9
Index du tableau en langage Scilab	1	2	3	4	5	6	7	8	9	10

TABLE 8 – Exemple de tableau de vitesses

Question 15

1. *En utilisant la liste proposée table 8, préciser à partir de quel index absolu et jusqu'à quel index absolu il est possible de déterminer une moyenne glissante.*
2. *Pour la liste proposée table 8, en déduire le nombre de moyennes glissantes différentes qu'il est possible de déterminer.*
3. *Pour une liste de n vitesses, combien de moyennes glissantes serait-il possible de déterminer ?*

La table 9 donne l'algorithme partiel de la fonction moyenneGlissante() qui permet de réaliser une moyenne glissante sur 4 valeurs consécutives.

Cet algorithme est établi pour les index absolus (le codage de cet algorithme prend bien sûr en compte les spécificités de chaque langage).

Algorithme : calcul de la moyenne glissante

```
Fonction moyenneGlissante(vit)
    Entrée : vit : liste de vitesses instantanées
    Sortie : res : liste des vitesses lissées sur 4 valeurs
    res ← liste vide
    Pour i allant de index_i_absolu_début à index_i_absolu_fin inclus par pas de 1
        Som ← 0
        Moy ← 0
        Pour j allant de index_j_absolu_début à index_j_absolu_fin inclus par pas de 1
            A compléter
            A compléter
        Fin Pour
        Ajouter Moy dans la liste res
    Fin Pour
    Retouner res
Fin Fonction moyenneGlissante
```

TABLE 9 – Algorithme partiel en pseudo-code de la fonction moyenne glissante

Question 16

1. *Pour l'algorithme proposé table 9 (page 13), déterminer les valeurs de* index_i_absolu_début *et* index_i_absolu_fin.
2. *Préciser quelles sont les valeurs de* index_j_absolu_début *et* index_j_absolu_fin *en fonction de* i.
3. *Compléter les lignes manquantes dans cet algorithme.*

Désormais, on dispose d'une liste de vitesses lissées (compte tenu de la méthode de calcul, par moyenne glissante, cette liste est toutefois de taille plus réduite que celle des vitesses initiales).

A chacune de ces vitesses (lissées), on souhaite associer un code couleur. Le code couleur est un entier variant de 0 à 255. Le code couleur 0 correspond à la vitesse la plus faible (dans la liste des vitesses lissées). Le code couleur 255 correspond à la vitesse la plus élevée (dans la liste des vitesses lissées). Le code couleur évolue linéairement en fonction de la vitesse. On cherche à obtenir une liste constituée des couples *(vitesse_lissee, code_couleur)*.

Question 17 *Dans la liste des vitesses lissées, indiquer quelles opérations de recherche sont nécessaires pour associer respectivement le code couleur 0 et le code couleur 255.*

Question 18 *En déduire la fonction mathématique à établir pour associer le code couleur à n'importe quelle vitesse lissée présente dans la liste.*

4.2 Application : affichage du tracé

On souhaite maintenant représenter le déplacement réel de manière graphique, sur une carte. On dispose pour cela d'une bibliothèque graphique détaillée en annexe 7 (page 20). Le principe de fonctionnement consiste à initialiser une carte (un fichier image de type JPEG ou PNG) en lui associant une échelle et une origine. Cette origine est saisie par l'utilisateur du programme. L'initialisation est réalisée par l'appel de la fonction initCarte() (voir annexe 7 page 20).

Afin de faciliter le tracé du déplacement, nous souhaitons implémenter une fonction ajoutePoint() qui trace un nouveau point sur la carte et le relie au précédent par un segment si ce dernier n'est pas le premier point tracé. S'il s'agit du premier point du déplacement, la fonction se contente de tracer ce point sur la carte.

On précise que cette fonction est en mesure de s'adapter aux deux cas de figure énoncés ci-dessus grâce à l'utilisation de paramètres par défaut. Ces derniers apparaissent directement dans la signature de la fonction en Python ou sont traités dans le corps de la fonction en Scilab.

Les spécifications de la fonction ajoutePoint() sont données table 10 (page 15) pour les deux langages (pour la version en langage Scilab, les premières lignes de code gérant les valeurs par défaut sont également présentes).

On suppose que la surface de la carte à afficher est réduite (quelques centaines de mètres carrés). La détermination de l'affichage des points GPS sur cette carte relève alors d'une opération de conversion. A partir de la circonférence de la Terre, on montre qu'un degré de latitude correspond à 111,199233 km soit 111 199,233 mètres et qu'un degré de longitude correspond à $111\,199{,}233 \cdot \cos(\text{Latitude})$ mètres.

Ces opérations de conversions sont réalisées dans le corps des fonctions tracePoint() et traceSegment() présentées en annexe 7 (page 20).

La couleur du tracé du segment dépend uniquement du code couleur associé au second point.

Python

```
def ajoutePoint (id_carte,lat_Pt_1,long_Pt_1,lat_Pt_2=0,long_Pt_2=0,coul=255) :
"""
Trace le point Pt_1 de latitude lat_Pt_1 et de longitude long_Pt_1 sur la carte correspondant à
l'identifiant id_carte. Trace un segment entre le point Pt_1 et le point Pt_2 si ce dernier existe.
Entrées :
   * id_carte : int, identifiant de la carte
   * lat_Pt_1, long_Pt_1 : flt, latitude et longitude du point Pt_1
   * lat_Pt_2, long_Pt_2 : optionnel, flt, latitude et longitude du point Pt_2
   * coul : optionnel, int, code couleur entre 0 et 255
"""
```

Scilab

```
function ajoutePoint (id_carte, lat_Pt_1, long_Pt_1, lat_Pt_2, long_Pt_2, coul)
// Trace le point Pt_1 de latitude lat_Pt_1 et de longitude long_Pt_1 sur la carte correspondant à
// l'identifiant id_carte. Trace un segment entre le point Pt_1 et le point Pt_2 si ce dernier existe.
// Entrées :
// * id_carte : int, identifiant de la carte
// * lat_Pt_1, long_Pt_1 : double, latitude et longitude du point Pt_1
// * lat_Pt_2, long_Pt_2 : optionnel, double, latitude et longitude du point Pt_2
// * coul : optionnel, int, code couleur entre 0 et 255
    if exists("lat_Pt_2","local")==0 then
        lat_Pt_2 = 0 ; end // Valeur par défaut
    if exists("long_Pt_2","local")==0 then
        long_Pt_2 = 0 ; end // Valeur par défaut
    if exists("coul","local")==0 then
        coul = 255 ; end // Valeur par défaut
```

TABLE 10 – Spécification de la fonction ajoutePoint() en Python et Scilab

Question 19 *Sachant que le point de départ sera affiché en rouge, proposer une implémentation de la fonction ajoutePoint().*

On dispose de la fonction trameGPS2Liste() (dont les spécifications sont données dans la table 11, page 16). Elle permet d'extraire les informations à partir du fichier tramesNMEA.txt et de retourner une liste de points de type pointGPSLisse (lire point GPS lissé) dans une liste en ayant calculé la vitesse lissée ainsi que le code couleur correspondant.

On précise que le calcul de la vitesse lissée par moyenne glissante est un peu plus complexe que le principe étudié dans la partie précédente car chaque point dispose d'une vitesse lissée et d'un code couleur même si le calcul direct n'a pas été possible. Le premier point (origine du déplacement) possède une vitesse lissée nulle. La vitesse du second point est la moyenne entre celle du premier point et celle du troisième point. La vitesse du dernier point est égale à celle de l'avant dernier point.

Ainsi un pointGPSLisse a la structure suivante :

pointGPSLisse = [latitude, longitude, vitesseLissee, codeCouleur].

La fonction trameGPS2Liste() retourne donc une liste de la forme suivante :

[pointGPSLisse_0,pointGPSLisse_1,...].

Question 20 *Compléter le programme principal qui affiche sur une carte le déplacement correspondant à la suite des trames GPS contenues dans le fichier tramesNMEA.txt. Le nom du fichier contenant la carte ainsi que l'échelle et l'orientation seront saisis à la console par l'utilisateur.*

Python

```
def trameGPS2Liste (fichier) :
    """
    Ouvre un fichier de trames et retourne une liste de points GPS lissés
    [pointGPSLisse_0,pointGPSLisse_1,...].
    Entrée :
        * fichier : str, nom du fichier contenant les trames
    Sortie :
        * res : list, liste de quadruplets
    """
```

Scilab

```
function [res] = trameGPS2Liste (fichier)
// Ouvre un fichier de trames et retourne une liste de points GPS lissés
// [pointGPSLisse_0,pointGPSLisse_1,...].
// Entrée :
// * fichier : str, nom du fichier contenant les trames
// Sortie :
// * res : list, liste de quadruplets
```

TABLE 11 – Spécification de la fonction trameGPS2Liste() en Python et Scilab

5 Synthèse des activités

Question 21 *Proposer une synthèse des activités permettant d'expliquer comment la problématique initiale a été résolue. Préciser éventuellement, quels peuvent être les avantages d'organiser les données sous la forme d'un tas binaire, comme présenté dans la partie 3 (page 10).*

Fin de l'énoncé

ANNEXES

Annexe 1 – Le protocole NMEA 0183, type GPRMC

Dans le protocole NMEA 0183, chaque trame est constituée de chaînes de caractères, terminées par un retour chariot et un saut de ligne. Les caractères constituant la trame ont un code ASCII compris entre 20 et 127 mais sont codés sur un octet. Toute trame commence par les caractères $GP, suivis de trois caractères définissant le type de trame. Chaque trame de type GPRMC est donc préfixée par $GPRMC.

On donne ci-dessous un exemple de trame :

$GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A*65

Les trames (cf exemple ci-dessus), sont constituées de 13 champs texte, séparés par des virgules et terminés par une somme de contrôle de parité, résultat d'un « ou exclusif » de chaque octet compris strictement entre le caractère initial "$" et le caractère "*". L'interprétation de chaque champ dépend alors de sa position.

Position	Exemple	Interprétation
0	$GPRMC	Type de trame. Nous considérons uniquement le type RMC.
1	071139.988	Heure UTC de la mesure sous la forme hhmmss.sss (avec hh heures, mm minutes dans l'heure, ss.sss secondes et secondes décimales)
2	A	"A" pour mesure valide, "V" pour mesure invalide
3	4639.8232	Latitude donnée sous la forme ddmm.mmmm (avec dd degrés, mm.mmmm minutes et minutes décimales)
4	N	Complète le champ 3 pour préciser s'il s'agit de la latitude Nord ("N") ou Sud ("S")
5	00021.6810	Longitude donnée sous la forme dddmm.mmmm
6	E	Complète le champ 5 pour préciser s'il s'agit de la longitude Est ("E") ou Ouest ("W")
7	8.95	Vitesse en nœuds
8	184.14	Cap du déplacement par rapport au pôle Nord magnétique en degrés décimaux
9	141014	Date universelle donnée sous la forme jjmmaa (avec jj jour du mois, mm mois, aa année)
10	1.1	Variation magnétique sous la forme d.d en degrés décimaux.
11	W	Sens de la variation magnétique ("E" = Est, "W" = Ouest)
12	A *65	Mode de fonctionnement du récepteur Somme de contrôle exprimée en base hexadécimale sur deux chiffres
fin	<CR><LF>	La trame se termine par deux caractères : un retour chariot (code ASCII 13) puis un saut de ligne (code ASCII 10).

Annexe 2 – Table ASCII

Code décimal	Caractère ASCII	Description	Décimal	Caractère	Décimal	Caractère	Décimal	Caractère
0	NUL	Null	32	Space	64	@	96	‘
1	SOH	Start of heading	33	!	65	A	97	a
2	STX	Start of text	34	”	66	B	98	b
3	ETX	End of text	35	#	67	C	99	c
4	EOT	End of transmission	36	$	68	D	100	d
5	ENQ	Enquiry	37	%	69	E	101	e
6	ACQ	Acknowledge	38	&	70	F	102	f
7	BEL	Bell	39	’	71	G	103	g
8	BS	Backsapce	40	(	72	H	104	h
9	TAB	horizontal tab	41	)	73	I	105	i
10	LF	New line feed, new line	42	*	74	J	106	j
11	VT	Vertical tab	43	+	75	K	107	k
12	FF	NP form feed, new page	44	‘	76	L	108	l
13	CR	Carriage return	45	-	77	M	109	m
14	SO	Shift out	46	.	78	N	110	n
15	SI	Shift in	47	/	79	O	111	o
16	DLE	Data link espace	48	0	80	P	112	p
17	DC1	Device control 1	49	1	81	Q	113	q
18	DC2	Device control 2	50	2	82	R	114	r
19	DC3	Device control 3	51	3	83	S	115	s
20	DC4	Device control 4	52	4	84	T	116	t
21	NAK	Negative acknowledge	53	5	85	U	117	u
22	SYN	Synchronous idle	54	6	86	V	118	v
23	ETB	End of trans. block	55	7	87	W	119	w
24	CAN	Cancel	56	8	88	X	120	x
25	EM	End of medium	57	9	89	Y	121	y
26	SUB	Substitute	58	:	90	Z	122	z
27	ESC	Escape	59	;	91	[	123	{
28	FS	File separator	60	<	92	\	124	|
29	GS	Group separator	61	=	93	]	125	}
30	RS	Record separator	62	>	94	^	126	~
31	US	Unit separator	63	?	95	_	127	DEL

Annexe 3 – Fonctions communes aux deux langages

On considère que les fonctions suivantes sont communes au langage Python et au langage Scilab.

chr(i) : Return the string representing a character whose Unicode codepoint is the integer i. For example, chr(97) returns the string 'a'. This is the inverse of ord(). The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). ValueError will be raised if i is outside that range.

Remarque : pour i compris entre 0 et 127, la fonction retourne le caractère correspondant au code ASCII.

ord(c) : Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, ord('a') returns the integer 97 and ord('\u2020') returns 8224. This is the inverse of chr().

Annexe 4 – Fonctions Python de traitement des fichiers

close() : Flush and close this stream. This method has no effect if the file is already closed. Once the file is closed, any operation on the file (e.g. reading or writing) will raise a ValueError. As a convenience, it is allowed to call this method more than once ; only the first call, however, will have an effect.

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None) : Open file and return a corresponding file object. If the file cannot be opened, an OSError is raised.

file is either a string or bytes object giving the pathname.

mode is an optional string that specifies the mode in which the file is opened. It defaults to 'r' which means open for reading in text mode. Other common values are 'w' for writing (truncating the file if it already exists), 'x' for exclusive creation and 'a' for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position).

readline(size=-1) : Read until newline or EOF and return a single str. If the stream is already at EOF, an empty string is returned. If size is specified, at most size characters will be read.

write(s) : Write the string s to the stream and return the number of characters written.

Annexe 5 – Fonctions Scilab de traitement des fichiers

[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format]) allows to open a file with specified properties and to get the associated unit number unit.

- file-name : string, file name of the file to be opened ;
- status : string, the status of the file to be opened :
  - "new" : file must not exist new file (default) ;
  - "old" : file must already exists ;
  - "unknown" : unknown status ;
  - "scratch" : file is to be deleted at end of session.

file(action,unit) : allows to close the file , or move the current file pointer. action : is one of the following strings :

- "close" : closes the file(s) given by the logical unit descriptors given in units ;
- "rewind" : puts the pointer at beginning of file ;
- "backspace" : puts the pointer at beginning of last record ;
- "last" : puts the pointer after last record.

[x] = read(file-desc,m,n,[format]) :reads row after row the mxn matrix x (n=1 for character chain) in the file file-desc (string or integer). Each row of the matrix x begin in a new line of file-desc file. Depending on format, a given row of the x matrix may be read from more than one line of file-desc file.

write(file-desc,a,[format]) : writes row-by-row a real matrix or a column vector of character strings in a formatted file. Each row of the a argument begin in a new line of file-desc file. Depending on format a given row of the a argument may be written in more than one line of file-desc file.

Annexe 6 – Précisions sur les chaînes de caractères en langage Python et en langage Scilab

Le stockage des chaînes de caractères n'est pas strictement identique dans les deux langages et les fonctions de traitement de chaînes ne sont pas non plus rigoureusement identiques. Pour lever toute ambiguïté, le tableau ci-contre précise les index de début et de fin de la chaîne ''abcde'' dans les langages Python et Scilab.

	Python	Scilab
Index de début	0	1
Index de fin	4	5

La table 12 précise comment extraire la sous-chaîne ''bcd'' de la chaîne ''abcde'' dans les deux langages.

Python

```
1 >>> chaine = "abcde"
2 >>> sous_chaine = chaine[1:4]
3 >>> print(sous_chaine)
4       'bcd'
```

Scilab

```
1 --> chaine = "abcde"
2 --> sous_chaine = part(chaine,2:4)
3 --> printf(sous_chaine)
4       bcd
```

TABLE 12 – Extraction de chaîne de caractère en langages Python et Scilab

Annexe 7 : API d'une bibliothèque graphique

Nous donnons ci-dessous la description d'une bibliothèque graphique utilisable dans les langages Python et Scilab.

Fonction	**Description**
initCarte(carte,echelle,Lat,long)	Initialise une carte en fixant l'échelle de celle-ci ainsi que son orientation. Entrées : – carte : str, chemin du fichier image ; – echelle : flt en Python, double en Scilab, échelle en pixel/m ; – Lat : flt en Python, double en Scilab, latitude de l'origine ; – long : flt en Python, double en Scilab, longitude de l'origine. Sortie : – idcarte : int, identifiant de la carte.
afficheCarte(idcarte)	Affiche la carte ayant l'identifiant idcarte à l'écran. Entrée : – idcarte : int, identifiant de la carte à afficher à l'écran.
tracePoint(idcarte,X1,Y1)	Calcule la projection et trace un point de coordonnées (X1,Y1) et de couleur rouge sur la carte d'identifiant idcarte. Entrées : – idcarte : int, identifiant de la carte ; – X1 : flt en Python, double en Scilab, abscisse du point ; – Y1 : flt en Python, double en Scilab, ordonnée du point.
traceSegment(idcarte,X1,Y1,X2,Y2,coul)	Calcule les projections et trace un segment de couleur coul entre les points de coordonnées (X1,Y1) et (X2,Y2) sur la carte d'identifiant idcarte. Suivant la vitesse du véhicule, la couleur peut varier de gris clair à noir. Entrées : – idcarte : int, identifiant de la carte ; – X1 : flt en Python, double en Scilab, abscisse du point 1 ; – Y1 : flt en Python, double en Scilab, ordonnée du point 1 ; – X2 : optionnel, flt en Python, double en Scilab, abscisse du point 2 ; – Y2 : optionnel, flt en Python, double en Scilab, ordonnée du point 2 ; – coul : optionnel, int, couleur du segment entre 0 et 255 (255 par défaut).

ACQUISITION ET EXPLOITATION D'UN DEPLACEMENT A PARTIR D'UN RECEPTEUR GPS

DOCUMENT REPONSE

Question 1

.....

.....

.....

Question 2

1. .....

.....

.....

2. .....

.....

.....

Question 3

.....

Question 4

1. .....

.....

2. .....

.....

Question 5

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python **Scilab**

Question 6

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python **Scilab**

Question 7

1. .....

   .....

   .....

2.

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python **Scilab**

Question 8

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python

```
def cherchePremiereOccurence(car,chaine):
    """ Retourne l'indice de la première occurrence d'un caractère.
    Entrées :
      * car : str, caractère
      * chaine : str, chaine de caractères
    Sortie(s) :

    """
    n = len(chaine)

    while
        if chaine[ ]==car:
```

Scilab

```
function cherchePremiereOccurence(car,chaine)
//  Retourne l'indice de la première occurrence d'un caractère.
// Entrées :
// * car : str, caractère
// * chaine : str, chaine de caractères
//  Sortie :
//
    n=length(chaine)


    while
        if part( )==car then




endfunction
```

.....

.....

Question 9

1. .....

.....

.....

2.

Question 10

1. GPRMC,071139.988,A,4639.8232,N,00021.6810,E,8.95,184.14,141014,1.1,W,A

Python **Scilab**

	Latitude	**Orientation**	**Longitude**	**Orientation**	**Vitesse**
Index début					
Index fin					

2. **Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.**

Python

```
def creationPointGPS(NMEA_extraite) :
    latitude =
    longitude =
    vitesse =
    pointGPS =
    return pointGPS
```

Scilab

```
function [pointGPS] = creationPointGPS(NMEA_extraite)
    latitude =
    longitude =
    vitesse =
    pointGPS =
endfunction
```

Question 11

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python **Scilab**

Question 12

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python

```
def percolateUp (tas) :
    index = len(tas)-1
    val = tas[index]
    while index>0 :
        index_pere = int((index-1)/2)
        pere = tas[index_pere]
        if pere > val :
            break
        else :
            tas[index] = pere
            tas[index_pere] = val
            index = index_pere
```

Scilab

```
function [tas]=percolateUp(tas)
    index = size(tas);
    val=tas(index);
    index_pere = (index)/2;
    while index_pere>=1
        pere = tas(index_pere);
        if pere>val then
            break;
        else
            tas(index)=pere;
            tas(index_pere)=val;
            index=index_pere;
            index_pere = (index)/2;
        end
    end
endfunction
```

Question 13

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

```
def insererPointGPS( ) :
```

Python

```
function [tas]=insererPointGPS( )
```

Scilab

Question 14

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

1.

```
def extractMax( ) :
```

Python

```
function [ ]=extractMax( )
```

Scilab

2. Complexité de l'opération d'extraction : .....

Question 15

1. Index absolu de début : .....

   Index absolu de fin : .....

2. Nombre de moyennes glissantes : .....

3. Nombre de moyennes glissantes en fonction de n : .....

Question 16

1. *index_i_absolu_début* = ...

   index_i_absolu_fin = ...

2. *index_j_absolu_début* = ...

   index_j_absolu_fin = ...

3.

```
Fonction moyenneGlissante(vit)
    Entrée : vit : liste de vitesses instantanées
    Sortie : res : liste des vitesses lissées sur 4 valeurs
    res ← liste vide
    Pour i allant de index_i_absolu_début à index_i_absolu_fin inclus par pas de 1
        Som ← 0
        Moy ← 0
        Pour j allant de index_j_absolu_début à index_j_absolu_fin inclus par pas de 1
            .....
            .....
        Fin Pour
        Ajouter Moy dans la liste res
    Fin Pour
    Retouner res
Fin Fonction moyenneGlissante
```

Question 17

.....

.....

.....

.....

Question 18

.....

.....

.....

.....

.....

.....

.....

.....

Question 19

Réponse en langage Python OU en langage Scilab. Entourer le langage retenu.

Python

```
def ajoutePoint(id_carte,lat_Pt_1,long_Pt_1,lat_Pt_2=0,long_Pt_2=0,coul=255) :
```

Scilab

```
function ajoutePoint (id_carte, lat_Pt_1, long_Pt_1, lat_Pt_2, long_Pt_2, coul)
    if exists("lat_Pt_2","local")==0 then
        lat_Pt_2 = 0 ; end // Valeur par défaut
    if exists("long_Pt_2","local")==0 then
        long_Pt_2 = 0 ; end // Valeur par défaut
    if exists("coul","local")==0 then
        coul = 255 ; end // Valeur par défaut
```

Question 20

Python

```
# Saisie du nom de fichier


# Saisie du nom de la carte
carte = input("Saisir le nom de la carte : ")
# Saisie de l'échelle (stockée sous forme de chaîne de caractères)
echelle = input("Saisir l'échelle : ")
echelle = flt(echelle) # Conversion de la chaîne de caractères en flottant
# Saisie de la latitude de l'origine de la carte
lat0 = flt(input("Saisir la latitude de l'origine : "))
# Saisie de la longitude de l'origine de la carte
long0 = flt(input("Saisir la longitude de l'origne : "))
# Récupération des points àpartir du fichier de trame tramesNMEA.txt


#Initialisation de la carte


# Affichage de la carte


# Ajout de points sur la carte àpartir de la liste de points pts
for i
```

Encadré pour répondre en langage Scilab à la page suivante.

Question 20 – Suite

Scilab

```
// Saisie du nom de fichier


// Saisie du nom de la carte
file = input("Saisir le nom de la carte : ")
// Saisie de l'échelle (le type de la variable correspond au type saisi)
echelle = input("Saisir l'échelle : ")
// Saisie de la latitude de l'origine de la carte
lat0 = input("Saisir la latitude de l'origine : ")
// Saisie de la longitude de l'origine de la carte
long0 = input("Saisir la longitude de l'origine : ")
// Récupération des points àpartir du fichier de trame tramesNMEA.txt


//Initialisation de la carte


// Affichage de la carte


// Ajout de points sur la carte àpartir de la liste de points pts
for i
```

Question 21

.....

.....

.....

.....

.....

.....

.....