Concours Commun INP (CCINP), France

Session 2017, filière TSI

Informatique

Durée : 3 heures

Ce que traite le sujet

- Partie I : modes jeu et statistiques
- Partie II : traitement du transfert des joueurs
- Partie III : évaluation des performances de l'équipe
- Partie IV : entraînement à la passe vissée

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/tsi/2017/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUNS
POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE TSI

INFORMATIQUE

Jeudi 4 mai : 14 h - 17 h

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 16 pages contenant le sujet (pages numérotées de 1/16 à 14/16) et les annexes (pages numérotées de 15/16 à 16/16) ;
- un second dossier de 8 pages constituant le document réponse (DR) à rendre en fin d'épreuve (pages numérotées de DR – 1/8 à DR – 8/8).

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

Seul le document réponse est à rendre.
Chaque partie ou sous-partie de ce sujet est indépendante.

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 doit être proposée en langage Python.

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
```

CONCEPTION D'UNE APPLICATION SPORTIVE "RUGBY MANAGER"

L'application « Rugby Manager » est un jeu destiné aux smartphones et aux tablettes. Il permet de créer une équipe, de l'entraîner et de jouer avec elle. Il est possible de jouer seul (contre l'intelligence artificielle du logiciel) ou en mode multijoueurs. Enfin, l'application est utilisable en ligne ou en Bluetooth.
L'objectif est de programmer un certain nombre de fonctions qui seront utilisées pour créer cette application.

I Étude des modes jeu et statistiques

I.1 Partie graphique du mode jeu

Dans cette sous-partie, nous souhaitons tout d'abord représenter la partie graphique du mode jeu, c'est-à-dire :

- un terrain ;
- deux équipes ;
- un ballon.

On souhaite afficher l'image du terrain de rugby suivant, enregistrée dans le répertoire "C:/CCP" sous le nom *stade.bmp*.

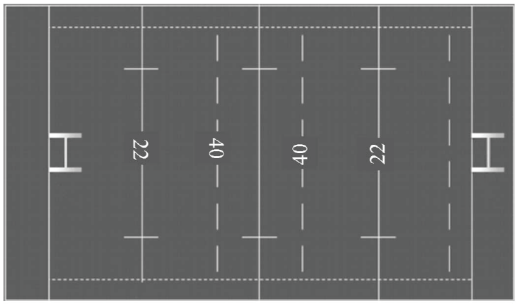


Figure 1 - Image du terrain de rugby

Objectif

Les compétences évaluées dans cette partie portent sur le traitement des images. Il s'agit d'utiliser la bibliothèque d'images décrite en **Annexe 1** pour compléter ou modifier des fonctions d'affichage de l'image du terrain et des pixels des maillots des joueurs. Ensuite, il s'agit de mettre en place un "floutage" d'une image pour la mettre en arrière-plan dans le mode statistique.

Q1. En utilisant l'**Annexe 1**, page 15, compléter le programme du document réponse qui permet :

- de se placer dans le dossier où se trouve l'image ;
- d'ouvrir l'image et de la stocker dans une variable *image_terrain* ;
- d'afficher l'image en arrière-plan.

Pour la suite et notamment pour l'affichage des joueurs à l'échelle du terrain, il est nécessaire de connaître les dimensions de l'image.

Q2. Donner les instructions qui permettent de faire cette opération et de stocker le résultat dans deux variables *dim_long* et *dim_larg* respectivement dimension en longueur et en largeur. Afficher ensuite le résultat sous la forme (*"longueur* × *largeur"*).

On propose tout d'abord de représenter les joueurs par 4 pixels disposés en carré ayant la couleur du maillot de l'équipe. Le choix des couleurs des maillots doit être laissé libre à l'utilisateur. Cependant, le vert correspondant à la couleur du terrain et le blanc correspondant à la couleur du ballon et des lignes ne pourront pas être utilisés.
Précisons que les lignes font 2 pixels de large, sur une image qui en fait 620. Il est donc facile de se positionner au centre du terrain. Les lignes numérotées 40, en pointillés, sont situées à 50 pixels de la ligne centrale.

Q3. Créer une fonction *coul( )* qui permet de connaître la couleur exacte du terrain et de la stocker dans une variable *coul_ter*. L'argument d'entrée de la fonction est la variable *image*, la sortie est la variable *coul_ter*. Attention : on choisira comme pixel de référence un des pixels le plus proche du centre du terrain.

Q4. Quel est le type de la variable *coul_ter* ?
Sur combien de bits mémoire est codé un pixel ?

Q5. Créer une fonction *maillot( )* qui utilise la fonction précédente et qui renvoie la liste des couleurs interdites pour les maillots.

I.2 Partie graphique du mode statistique

Objectif

Dans cette sous-partie, nous nous intéressons au traitement d'image évolué : la mise en place du "floutage" d'une image pour l'arrière-plan du mode statistique.

On s'intéresse donc ici à un autre mode du jeu vidéo : le mode statistique. Dans ce mode, une image de jeu floue va être placée en arrière-plan. Nous allons étudier les fonctions permettant d'effectuer ce floutage.
Pour réaliser un floutage par moyenne simple sur la matrice de pixels, il faut lui appliquer un filtre, que l'on appelle également un masque. Afin de comprendre ce principe, nous proposons d'étudier un exemple de filtrage. On considère les matrices

$$A = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

et

$$B = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \\ -5 & -6 & -7 & -8 & -9 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 3 & 3 & 4 & 4 \\ 0 & 0 & 1 & 3 & 3 & 3 \end{pmatrix}.$$

Pour chaque élément b_{ij} de B, on considère la matrice carrée de taille 3, notée B_{ij} qui l'entoure et on calcule le produit de B_{ij} par A (multiplication classique $A*B_{ij}$). A est appelé noyau. On note c_{ij} la somme des coefficients de la matrice produit obtenue (on pourra utiliser la fonction *np.sum* sur une liste).
Si b_{ij} est un élément en bordure de B, on posera $c_{ij} = b_{ij}$. On forme ainsi une nouvelle matrice C dont les éléments intérieurs sont les c_{ij} (et les éléments au bord sont les b_{ij}) :

- c_{ij} = élément se trouvant ligne i et colonne j ;
- B_{ij} la matrice carrée dont l'élément central est b_{ij}.

Ainsi, par exemple

$$c_{22} = sum(B_{22} * A) = sum\left(dot\left(\begin{pmatrix} 5 & 6 & 7 \\ -5 & -6 & -7 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}\right)\right)$$
$$= sum\begin{pmatrix} 2 & 2 & 2 \\ -2 & -2 & -2 \\ 1/3 & 1/3 & 1/3 \end{pmatrix} = 1,$$

dot et *sum* étant des fonctions du module numpy, avec *dot*(B,A) qui effectue le produit matriciel $B * A$ (s'écrit aussi $B \cdot dot(A)$) et *sum* qui somme tous les termes d'une matrice.

On dit que l'on a filtré la matrice B par la matrice A, ou que l'on a appliqué le masque (filtre) A sur l'image B.

Q6. Compléter la fonction *filtrer1(filtreA,matB)* qui prend en argument une matrice carrée *filtreA* de dimension *taille* × *taille* et une matrice quelconque *matB* de dimensions supérieures et qui renvoie la matrice C. Vous pouvez utiliser les fonctions du module numpy que vous jugez judicieuses sans préciser l'importation.

On souhaite maintenant appliquer le *filtreA* à une matrice de pixels B, c'est-à-dire aux 3 tableaux $B[:,:,0]$, $B[:,:,1]$, $B[:,:,2]$ et enregistrer le résultat dans une matrice C de même format que B.

Q7. Créer une fonction *filtrer(filtreA,matB)* qui prend en argument une matrice carrée *filtreA* de dimension *taille* × *taille* et un tableau numpy *matB* de dimensions $n \times p \times 3$ et qui renvoie le tableau C de dimensions identiques. Vous pourrez calculer successivement $C[:,:,0]$, $C[:,:,1]$, $C[:,:,2]$ à l'aide d'une boucle.

Ces matrices réalisent un floutage par moyenne simple (coefficients tous égaux, de somme 1). Plus la taille du filtre est grande, plus le flou sera fort. On peut améliorer cette technique en utilisant un flou gaussien. Son principe est de calculer une moyenne pondérée en accordant plus de poids au pixel central et en diminuant le poids des pixels périphériques.

La matrice servant de filtre est calculée selon le modèle d'une courbe de Gauss (courbe en cloche) à 2 dimensions.

La fonction de Laplace-Gauss à une dimension est $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$ et à deux dimensions est $G(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}}$.

Plus l'écart-type σ est grand, plus l'image sera floutée. En pratique, σ et la taille (impaire) du filtre étant fixés, on construit la matrice filtre terme à terme, chaque élément de la matrice ayant pour expression $k.e^{-\frac{x^2+y^2}{2\sigma^2}}$, où x et y sont les nombres de lignes et de colonnes qui séparent cet élément du centre et k un coefficient constant, tel que la somme de tous les éléments (de type float) ainsi calculés soit égale à 1.

Par exemple, pour $\sigma = 0{,}9$ et *taille* $= 5$, le filtre est proche de :

$$\frac{1}{1\,002}\begin{pmatrix} 1 & 9 & 17 & 9 & 1 \\ 9 & 58 & 107 & 58 & 9 \\ 17 & 107 & 198 & 107 & 17 \\ 9 & 58 & 107 & 58 & 9 \\ 1 & 9 & 17 & 9 & 1 \end{pmatrix}.$$

Q8. Compléter la fonction *matriceFlouGaussien(taille,sigma)* qui prend en argument la *taille* (impaire) de la matrice de floutage, *sigma* l'écart-type de déviation standard et qui retourne la matrice filtre correspondant au niveau gaussien. Encore une fois, vous pouvez utiliser les fonctions du module numpy que vous jugez judicieuses sans préciser l'importation.

Q9. Écrire ensuite la fonction *FloutageGaussien(tabPix,taille,sigma)* qui utilise la fonction *matriceFlouGaussien(taille,sigma)* et *filtrer(filtreA,matB)* et qui renvoie la matrice *tabPix* qui a été floutée grâce au filtre défini dans la fonction *matriceFlouGaussien*.

I.3 Fonctionnalités du mode statistique

Objectif

Dans cette sous-partie, nous nous intéressons à certaines fonctions du mode statistique. Nous proposons tout d'abord de mettre en œuvre les fonctions permettant de tracer les statistiques de chaque joueur à l'aide d'histogrammes, puis d'écrire la fonction donnant les performances moyennes, minimales et maximales.

Nous allons ici étudier un des tableaux qui sera utilisé dans ce mode. La matrice *resultat* permet de stocker l'ensemble des données sur chaque joueur de l'équipe. Les lignes correspondent aux numéros des joueurs et les colonnes sont respectivement : le nombre d'essais, le nombre de passes réussies, le nombre de plaquages réussis, le nombre de plaquages manqués, le nombre de franchissements de ligne, le nombre de kilomètres parcourus.

Numéro du joueur	nombre d'essais	nombre de passes réussies	nombre de plaquages réussis	nombre de plaquages manqués	nombre de franchissements de ligne	nombre de kilomètres parcourus
1						
2						
...						

On souhaite représenter un histogramme de l'équipe pour chacune des colonnes de *resultat*. Il s'agit d'une représentation graphique présentant en abscisse le numéro de maillot du joueur et en ordonnée la donnée en question.
Nous allons étudier ici l'histogramme pour le nombre de plaquages réussis, soit la 4^{e} colonne de *resultat*. L'affichage de l'histogramme ne sera pas étudié ici.

Q10. Écrire les instructions qui, à partir de la matrice *resultat*, construisent la liste des abscisses *x* de l'histogramme et la liste des ordonnées *y_plaR*.

On souhaite maintenant créer l'histogramme. Pour cela, il faut créer simultanément deux listes à partir de *x* et *y_plaR* :

- une liste *barre*, qui sera constituée, pour $i \in [0, len(x)[$, des valeurs allant de 0 à la valeur *y_plaR[i]* incluse avec des pas de *0,1* ;
- une liste *abscisse*, qui, pour $i \in [0, len(x)[$, sera composée d'autant d'éléments que *barre*, tous égaux à *x[i]*.

L'instruction *plot(abscisse,barre,'k.',markersize=10)* donnera alors l'histogramme présenté sur la **figure 2** :

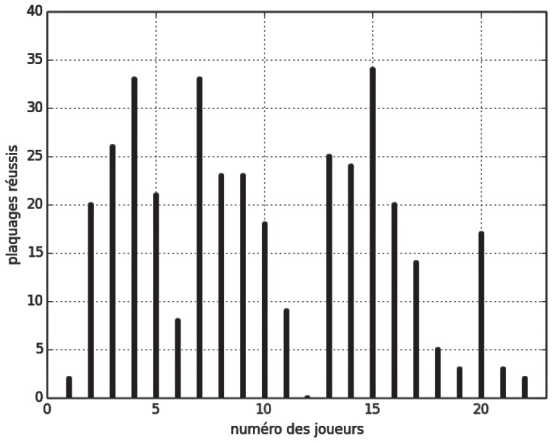


Figure 2 - Histogramme

Q11. Écrire les instructions qui, à partir de la matrice des listes *x* et *y_plaR*, construisent les listes *barre* et *abscisse*.

D'autres statistiques disponibles permettent de connaître le minimum, le maximum et la moyenne par match sur l'ensemble de la saison d'un joueur, sur la donnée qui nous intéresse. L'obtention de la liste de valeurs concernant une donnée en particulier n'est pas étudiée ici, seule la fonction qui renvoie le minimum, le maximum et la moyenne d'une liste de valeurs (float) est étudiée.
Par exemple, pour 6 matchs, la liste des valeurs des plaquages réussis pour le joueur 15 est : *valeurs = [20,10,15,22,24]*.

Q12. Écrire une fonction *minMaxMoy(valeurs)*, qui prend en entrée une liste de valeurs flottantes et qui renvoie la liste contenant le minimum des valeurs de la liste, le maximum et la moyenne.

II Traitement du transfert des joueurs

Objectif

Cette partie du sujet s'intéresse au traitement de la base de données des joueurs.
Dans le but de créer un jeu en ligne, il est nécessaire de prendre en compte les transferts de joueurs pour la création de sa propre équipe. Pour cela, on souhaite utiliser une base de données.
L'objectif est d'analyser la structure de la base de données proposée et de réaliser les requêtes permettant de créer sa propre équipe.

On possède une base de données *rugby.db3* avec 2 tables : *Joueurs* et *Clubs*.
On souhaite modéliser les transferts des joueurs de club en club comme les transferts réels. L'application donne un certain budget à son utilisateur et en fonction de ses victoires, il pourra acheter des joueurs de plus en plus chers.

Nous allons étudier comment les tables de joueurs et leurs caractéristiques sont associées aux clubs et au salaire des joueurs. Le *Salaire* annuel sera composé de sommes en euros, alors que celle des *Budget* des Clubs sera en millions d'euros. Ces attributs représentent leur valeur financière.

Joueurs							
Nom	Poste	Age (an)	Poids (kg)	Taille (cm)	VMA (km/h)	Id_Club	Salaire euros
Murel Thibault	Pilier Gauche	18	105	187	13,6	102	16 000
Nkang Jesus	Pilier Droit	27	115	176	12,9	10	35 000
Yala Olive	Troisième ligne Aile	17	90	182	14,5	45	12 000
Ramis Thomas	Arrière	20	80	178	18,5	31	40 000

Clubs		
Id_Club	Nom	Budget total
1	Stade Français	50
2	Ulster	165
31	Stade Toulousain	150
45	Racing Club de Toulon	540

Q13. Donner la requête qui permet de trouver les joueurs de plus de 23 ans qui ont une Vitesse Maximale Aérobie (notée VMA) supérieure à 13 km/h.

Q14. Donner la requête qui permet de connaître les clubs dont les joueurs ont un salaire supérieur à 30 000 euros.
Puis, déterminer la requête qui permet de connaître le nombre de joueurs étant au Stade Toulousain et jouant au poste de Talonneur.

Q15. Donner la requête qui permet de calculer le rapport entre la masse salariale des joueurs d'un club (somme des salaires de tous les joueurs) et le budget total du club. Le résultat sera donné en pourcentage du budget total du club.

III Évaluation des performances de l'équipe

Objectif

L'étape suivante permet de construire son équipe à partir de la liste des joueurs. Pour cela, il est important de pouvoir classer ses joueurs suivant ses préférences : le but est de trier la liste des joueurs issus de notre base de données de la manière la plus efficace pour l'utilisateur.
Dans cette partie, on s'intéresse donc au tri de la liste des joueurs de notre équipe en fonction de leurs performances.

Suite au transfert des joueurs dans notre équipe, nous avons créé une table *Equipe* avec les joueurs suivants :

Equipe					
Joueur	Poste	Age (an)	Poids (kg)	Taille (cm)	VMA (km/h)
Morellon Thibault	Pilier Gauche	18	105	187	13,6
Nkeno Jules	Pilier Droit	27	115	176	12,9
Yolozo Olivier	Troisième ligne Aile	17	90	182	14,5
Ramis Thomas	Arrière	20	80	178	18,5
Mechand Julien	Talonneur	20	105	181	13,6
Cazo Cyril	Deuxième ligne	20	115	197	12,9
Blue Richie	Deuxième ligne	26	112	206	15,5
Diverge Martin	Troisième ligne Aile	20	108	193	18,5
Dusaut Thierry	Troisième ligne Centre	33	100	188	13,6
Michal Pierre	Demi de mêlée/Demi d'ouverture	27	115	176	12,9
Dupont Antoine	Demi de mêlée	18	70	174	14,5
Choux Jonathan	Demi d'ouverture	30	92	188	18,5
Robot Jamie	Trois quart Centre	25	104	193	14,9
Totana Wesley	Trois quart Centre	27	93	182	18,5
Facke Gael	Trois quart Centre	21	90	190	17,5
Dupond Alex	Trois quart Aile	25	104	198	18,5
Wagon Anthony	Trois quart Aile	21	92	188	18,5
Cours Vincent	Trois quart Aile	34	80	178	18,5

Les instructions suivantes permettent d'afficher le tableau de l'équipe sous Python et de le stocker dans la variable *monequipe*.

```
1 import sqlite3 # Import des commandes permettant de manipuler la base de donnees
2 basesql = u"rugby.s3db" # Base de donnees initiale
3 cnx = sqlite3.connect(basesql )
4 curseur = cnx.cursor ()
5 requete = "SELECT * FROM Equipe"
6 curseur.execute(requete)
7 monequipe = curseur.fetchall()
8 print (monequipe)
```

Q16. Préciser le type de la variable *monequipe*.

Q17. Corriger les 2 erreurs de l'algorithme de tri suivant ainsi que l'appel de la fonction afin d'afficher les joueurs dans l'ordre de leur rapidité (rayer la ligne qui vous semble erronée et corriger à coté). La liste en entrée de *tri_1* est une liste de la forme [[joueur1, poste1, age1, poids1, taille1, VMA1], [joueur2, poste2, age2, poids2, taille2, VMA2], ...] et le paramètre critère correspond à l'indice du critère qui doit être trié.

```
1  def echange(l,i,j):
2          """ echange 2 valeurs d'une liste """
3          l[i],l[j] = l[j], l[i]
4
5  def tri_1(liste,critere):
6      """ tri la liste en fonction du critere choisi """
7      for i in range(liste):
8          mini=i
9          for j in range(i+1,len(liste)):
10                 if liste[j][critere] > liste[mini][critere]:
11                     mini=j
12         echange(liste,i,mini)
13     return liste
14
15 tri_1(monequipe,5)
16
17 print(monequipe)
```

Q18. En appelant n la taille de la *liste*, donner la complexité de la fonction *tri_1( )* en fonction de n dans le pire des cas. Critiquer le résultat.

Q19. Commenter et expliquer chaque ligne d'instruction de la fonction *tri_2( )* suivante (on ne commentera pas *segmente*).
Quelle est la particularité de cette fonction *tri_2( )*?
Modifier la fonction *tri_2( )* afin qu'elle retourne comme résultat le nombre d'appels récursifs de la fonction *tri_2( )*.

```
1  def segmente(T,val,i,j):
2      g=i+1
3      d=j
4      p=T[i][val]
5      while g<=d:
6          while d>=0 and T[d][val]>p:
7              d=d-1
8          while g<=j and T[g][val]<=p:
9              g=g+1
10         if g<d:
11             echange(T,g,d)
12             d=d-1
13             g=g+1
14     k=d
15     echange(T,i,d)
16     return k
17
18 def tri_2(L,val, i, j):
19     if i<j:
20         k=segmente(L,val,i,j)
21         tri_2(L,val,i,k-1)
22         tri_2(L,val,k+1,j)
23     return L
```

Q20. Donner alors l'instruction qui utilise cette fonction *tri_2( )* pour trier l'équipe en fonction du poids des joueurs.

Pour augmenter l'interactivité du jeu, il sera possible de rentrer ou de modifier des données de chaque joueur afin qu'elles correspondent à la réalité. Pour cela nous avons besoin de pouvoir insérer dans l'ordinateur des nombres à virgule flottante et de les convertir du décimal au binaire. Afin de ne pas gérer des formats de nombres trop grands, on choisit la norme IEE754 simple précision pour stocker les nombres à virgule flottante sur 32 bits (même si ceux-ci ne sont pas implémentés dans le langage Python).
Il n'est nul besoin d'avoir des connaissances de la norme IEE754 pour répondre aux différentes questions. Toutes les informations nécessaires pour répondre à une question seront explicitement données dans son énoncé.

Prenons un exemple pour bien comprendre le format de stockage.

Q21. Convertir le nombre décimal 74,25 en binaire.

Q22. Exprimer ce nombre au format IEE754 simple précision. On indique que les 32 bits du format IEE754 simple précision s'organisent comme suit : le bit de signe, puis l'exposant biaisé sur 8 bits (donc un biais de 127), puis la mantisse sur les 23 derniers bits.

La base de données contient 3 000 joueurs ayant chacun des caractéristiques de poids, taille et vitesse, qui peuvent être des nombres flottants.

Q23. Estimer la quantité de mémoire que représentent uniquement les données numériques. Donner le résultat en *kilo octets* (on approximera *1 ko = 1 000 o)*.
Quels sont les avantages et inconvénients du format simple précision pour la gestion des nombres flottants par rapport au format double précision ?

IV Entraînement à la passe vissée au rugby

Objectif

L'objectif de cette partie est de modéliser, puis de déterminer la trajectoire d'une balle pour l'insérer dans le mode entraînement du jeu.

Une des spécificités de cette application est qu'elle permet aussi un mode d'entraînement pour améliorer les passes. Afin de réaliser des passes réalistes, dites « vissées » au rugby, il faut modéliser l'ensemble des actions qui s'exercent sur le ballon et étudier sa dynamique.

L'un des phénomènes à modéliser est la résistance de l'air. Le ballon se déplace à sa vitesse maximale quand il vient d'être lâché par les mains, puis la résistance de l'air le ralentit immédiatement. Sa vitesse diminue jusqu'à ce qu'il atteigne sa hauteur maximale. Le ballon reprend ensuite de la vitesse quand il redescend.
Un autre phénomène physique intervient dans le comportement du ballon lors des passes : les effets donnés au ballon par les joueurs en utilisant différentes techniques de prise de ballon et de lâché. Le mouvement du ballon est alors modifié par l'effet Magnus, provoqué par la naissance, dans le sillage du ballon, de tourbillons capricieux appelés vortex.
On cherche à prendre en compte ces deux phénomènes dans l'application.
La modélisation du mouvement du ballon conduit à l'équation

$$\tau \frac{d\,v(t)}{d\,t} + v(t) = K_c\, u(t)$$

avec :

- τ, constante de temps ;
- K_c, gain statique ;
- v, vitesse du ballon ;
- u, sollicitation.

L'objectif est d'obtenir la vitesse du ballon puis de déterminer sa trajectoire ultérieurement. Cette vitesse est obtenue pour une réponse temporelle du système à une entrée échelon (entrée constante) : $u(t) = U_0$.

Q24. Écrire une fonction *liste_temps(pas,tmax)* renvoyant une liste des temps à partir du pas (intervalle entre deux instants ; cet intervalle sera pris constant) et de *tmax* (borne supérieure des temps).

Q25. Donner la solution analytique de l'équation différentielle précédente, en considérant des conditions initiales nulles (cela sera le cas dans toute la suite du sujet).
Écrire une fonction *vitesse(k,tau,u,temps)* renvoyant une liste des valeurs des vitesses.

Q26. Programmer, en utilisant la méthode d'Euler d'ordre 1, une fonction *ordre1_euler(K_c, tau, U_0, temps)*. Cette fonction prendra comme arguments d'entrée les coefficients de l'équation différentielle, la valeur de l'échelon d'entrée et la liste des temps. Cette fonction retournera une liste.

Q27. Nous voulons tester le programme pour différents pas de discrétisation, de [0,2; 0,4; 0,6]. Écrire la boucle qui permet de résoudre l'équation différentielle pour les différents pas en utilisant la fonction programmée à la question **Q26**.

La bibliothèque matplotlib est détaillée en **Annexe 2** (page 16).

Q28. À partir de la question précédente, réécrire le script qui permet d'afficher les résultats de la résolution numérique par la méthode d'Euler pour les trois pas de temps considérés.

Annexe 1 – Traitement d'images

Afficher une image

```
1  import os // Import des commandes permettant de gerer les repertoires de travail
2  os.chdir('Repertoire de travail') // on se place dans le repertoire ou se trouve la base de donnees
3
4  import scipy.misc as scm // bibliotheque image
5
6  picture=scm.imread("image.bmp") // stocker l'image dans une variable
7  imshow(picture) // afficher l'image
8
9  print (picture) // donne le nombre de pixels de l'image
```

picture est alors un tableau aux dimensions du nombre de pixels et contenant un triplet défini pour chaque pixel utilisant le format RGB (Red, Green, Blue) en français RVB (Rouge, Vert, Bleu). La couleur "Rouge" est codée entre 00 et FF en notation hexadécimale et entre 0 et 255 en notation décimale (idem pour les couleurs "Vert" et "Bleu").

```
1  picture = array ([[[138 194 138]
2    [138 194 138]
3    [138 195 138]
4    ...,
5    [138 195 138]
6    [138 195 138]
7    [138 194 138]]
8    ...,
9    [ 49 151 49]
10   [ 49 151 49]
11   [ 49 151 49]]])
```

```
1  xmax= picture.shape[1] #taille du tableau
2  ymax= picture.shape[0] #taille du tableau
3  print ("taille de l'image : %d x %d" % (xmax,ymax))
```

Pour obtenir la couleur du pixel, il suffit donc de choisir le ou les bons indices du tableau.

Annexe 2 – Bibliothèque MATPLOTLIB.PYPLOT de Python

Cette bibliothèque permet de tracer des graphiques.
Dans les exemples ci-dessous, la bibliothèque matplotlib.pyplot a préalablement été importée à l'aide de la commande :

```
1 import matplotlib.pyplot as plt
```

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

```
1 plt.plot(x,y)
```

- Arguments d'entrée : un vecteur d'abscisses x (tableau de dimension n) et un vecteur d'ordonnées y (tableau de dimension n)
- Description : fonction permettant de tracer sur un graphique n points dont les abscisses sont contenues dans le vecteur x et les ordonnées dans le vecteur y. Cette fonction doit être suivie de la fonction **plt.show()** pour que le graphique soit affiché.
- Exemple :

```
1 x= np.linspace(3,25,5)
2 y=sin(x)
3 plt.plot(x,y)
4 plt.xlabel('x')
5 plt.ylabel('y')
6 plt.show()
```

```
1 plt.xlabel(nom)
```

- Argument d'entrée : une chaîne de caractères.
- Description : fonction permettant d'afficher le contenu de nom en abscisse d'un graphique.

```
1 plt.ylabel(nom)
```

- Argument d'entrée : une chaîne de caractères.
- Description : fonction permettant d'afficher le contenu de nom en ordonnée d'un graphique.

```
1 plt.show()
```

- Description : fonction réalisant l'affichage d'un graphe préalablement créé par la commande **plt.plot(x,y)**. Elle doit être appelée après la fonction **plt.plot** et après les fonctions **plt.xlabel** et **plt.ylabel**.

FIN

IMPRIMERIE NATIONALE – 17 1195 – D'après documents fournis

DANS CE CADRE

Académie : Session :

Examen ou Concours : **Concours Communs Polytechniques** Série* :

Spécialité/option : **FILIÈRE TSI** Repère de l'épreuve :

Épreuve/sous-épreuve : **Informatique**

NOM :

(en majuscules, suivi, s'il y a lieu, du nom d'épouse)

Prénoms :

Né(e) le

N° du candidat

(le numéro est celui qui figure sur la convocation ou la liste d'appel)

NE RIEN ÉCRIRE

Examen ou Concours : **Concours Communs Polytechniques** Série* :

Spécialité/option : **FILIERE TSI**

Repère de l'épreuve : **Informatique**

Épreuve/sous-épreuve :

(Préciser, s'il y a lieu, le sujet choisi)

Si votre composition comporte plusieurs feuilles, numérotez-les et placez les intercalaires dans le bon sens.

Note : /20

Appréciation du correcteur :*

* Uniquement s'il s'agit d'un examen.

TSIIN07

CONCEPTION D'UNE APPLICATION SPORTIVE "RUGBY MANAGER"

DOCUMENT RÉPONSE

Question 1

```
1  import os
2  ...
3
4  """affichage stade"""
5  import ...
6  ...
7  ...
```

Question 2

```
1  ...
2  ...
3
4  print( ...
```

Question 3

```
1  def coul(
2  ...
3
4
5
6
7
8  ...
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 4

.....

.....

.....

Question 5

```
1  def maillot(
2
3
4
5
6
7              ...
```

Question 6

```
1  def filtrer1(filtreA,matB):
2      nA=filtreA.shape[0] #matrice carree, donc shape[0]=shape[1]
3      nb_ligneB=...
4      nb_colonneB=...
5      C=matB.copy()
6      bordure =...
7      for i ...
8          for j ...
9              Bij=...
10             C[i,j]=...
11     return ...
```

Question 7

```
1  def filtrer( ...
2
3
4
5
6
7
8
9
10
11
12     ...
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 8

```
1  def matriceFlouGaussien (taille , sigma):
2      """
3      taille : taille de la matrice ( impaire )
4      sigma : écart type (déviation standard )
5      retourne un niveau gaussien """
6      mat = zeros ([ taille , taille ])
7      taille = ...
8      for x in range (- taille , taille +1):
9          for y in range (- taille , taille +1):
10             ...
11         ...
12     return ...
```

Question 9

```
1  def FloutageGaussien ( ...
2
3
4
5
6
7
8
9
10
11
12
13
14         ...
```

Question 10

```
1  #x
2  ...
3
4
5
6  #y_plaR
7  ...
```

Question 11

```
1  # listes pour l'histogramme
2  ...
3  ...
4  ...
5  ...
6  ...
7  ...
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 12

```
1  def minMaxMoy(...
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19       ...
```

Question 13

Requete1=

Question 14

Requete2=

Requete3=

Question 15

Requete4=

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 16

type(monequipe):

Question 17

```
1  def echange(l,i,j):
2      """ echange 2 valeurs d'une liste """
3      l[i],l[j] = l[j], l[i]
4
5  def tri_1(liste,critere):
6      """ tri la liste en fonction du critère choisi """
7      for i in range(liste):
8          mini=i
9          for j in range(i+1,len(liste)):
10             if liste[j][critere] > liste[mini][critere]:
11                 mini=j
12         echange(liste,i,mini)
13     return liste
14
15 tri_1(monequipe,5)
16
17 print(monequipe)
```

Question 18

.....

.....

.....

.....

.....

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 19

```
1  def segmente(T,val,i,j):
2      g=i+1
3      d=j
4      p=T[i][val]
5      while g<=d :
6          while d>=0 and T[d][val]>p:
7              d=d-1
8          while g<=j and T[g][val]<=p:
9              g=g+1
10         if g<d:
11             echange(T,g,d)
12             d=d-1
13             g=g+1
14     k=d
15     echange(T,i,d)
16     return k
17
18 def tri_2(L,val, i, j): # ...
19
20     if i<j: # ...
21
22         k=segmente(L,val,i,j) # ...
23
24         tri_2(L,val,i,k-1) # ...
25
26         tri_2(L,val,k+1,j) # ...
27
28     return L # ...
29
30 #modifier la fonction tri_2
31 def tri_2(L,val, i, j):
32
33
34
35
36         ...
```

Question 20

Instruction : .....

Question 21

$74,25_{10}$ =.....

.....

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 22

Ecriture au format IEE754 simple précision : .....

.....

.....

.....

Question 23

Espace de stockage : .....

.....

Avantages et inconvénients du format simple ?.....

.....

.....

Question 24

```
1  def liste_temps(pas,tmax):
2  ...
3  ...
4  ...
5  ...
6  ...
```

Question 25

.....

.....

.....

.....

.....

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 26

```
1  def ordre1_euler(Kc, tau, U0, temps):
2  ...
3  ...
4  ...
5  ...
6  ...
```

Question 27

```
1  ...
2  ...
3  ...
4  ...
5  ...
```

Question 28

```
1  #importation du module graphique de python
2  ...
3  for
4  ...
5  ...
6  ...
7  ...
```

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Images : matrices de pixels, opérations, convolution
- Bases de Python : expressions, types, instructions, fonctions
- Listes et algorithmes de recherche (séquentielle, dichotomique)
- Tris par sélection, à bulles, par insertion et par comptage
- Récursivité
- Architecture de l'ordinateur et représentation des nombres
- Équations différentielles : méthode d'Euler
- Équations différentielles linéaires du premier ordre



Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/tsi