MPSI 2 **DS 2**

Informatique

Le 23/2/19

(2h)

Calculatrice interdite

Partie A (Python 3 et SQL) Quelques aspects du traitement informatique de données météorologiques

Une base de données fournie par Météo-France pour l'année 2013 est constituée de deux tables :

- la table ***villes****, qui stocke les informations sur chaque ville de France :*

insee *(chaîne de caractères) : numéro INSEE de la ville (deux villes différentes ont des numéros différents).*

nom *(chaîne de caractères) : nom de la ville.*

dpt *(chaîne de caractères) : nom du département de la ville.*

lat *(flottant) : latitude de la ville en degrés (positif si la ville est dans l'hémisphère nord).*

lon *(flottant) : longitude de la ville en degrés (positif si la ville est à l'Est du méridien de Greenwich).*

pop *(entier) : représentant la population de la ville (nombre d'habitants).*

- la table ***mesures****, qui enregistre l'évolution des températures au cours de l'année :*

ville *(chaîne de caractères) : numéro INSEE identifiant la ville.*

jour *(entier) : numéro j du jour de l'année où est faite la mesure.*

Tmin *(flottant) : température minimale mesurée le jour j en degrés Celsius.*

Tmax *(flottant) : température maximale mesurée le jour j en degrés Celsius.*

1-Rappeler la définition d'une clef primaire. Préciser pour chaque attribut de la table ***villes*** si celui-ci peut être ou non une clef primaire. Proposer une clé primaire pour la table ***mesures***. On justifiera clairement.

2-Pour stocker le numéro INSEE (code postal) d'une ville, on propose d'utiliser le type CHAR(x) ou le type VARCHAR(x). Préciser ce qui convient le mieux (justifier).

3-Décrire et comparer les affichages obtenus avec les deux requêtes suivantes :

```
SELECT nom,dpt,pop FROM villes ORDER BY pop DESC
SELECT DISTINCT nom FROM villes ORDER BY nom
```

Ecrire en langage SQL les requêtes qui permettent d'afficher :

4-a-La liste des noms de villes répertoriées en 2013 et situées dans le département des Yvelines.

4-b-Le nombre de villes répertoriées en 2013 dont la population dépasse strictement 100 000 habitants, dans un tableau nommé **grandesVilles**.

4-c-La température journalière minimum la plus élevée et la température journalière maximum moyenne relevées en 2013.

Les relevés de Météo-France de la ville de Besançon pour l'année 2013 ont été sauvegardés dans le fichier **besancon_2013.txt**. Ce fichier contient 365 lignes (une pour chaque mesure et donc jour de l'année).

Sur chaque ligne, la chaîne de caractère correspond à trois champs séparés par des points-virgules, à savoir :

– le numéro correspondant au jour de la mesure (entier naturel);
– la température minimale mesurée ce jour (en degrés Celsius, flottant);
– la température maximale mesurée ce jour (en degrés Celsius, flottant).

Extrait du fichier besancon_2013.txt

```
# Jour;Tmin;Tmax
1;2.1;7.6
2;2.3;4.9
3;-1.9;5.7
...
168;16.7;32.3
169;18.8;32.0
...
365;-3.2;1.9
```

Pour lire le fichier ***besancon_2013.txt****, on propose le programme suivant en* ***Python 3*** *:*

```
1  def lecture_fichier(fichier):
2      f = open(fichier,'□')
3      jours,Tmin,Tmax = [ ],[ ],[ ]
4      for ligne in f:
5          if ligne[0] != '#':
6              t,T1,T2 = ligne.split(';')
7              jours.append(int(t))
8              Tmin.append(float(T1))
9              Tmax.append(float(T2))
10     f.□□□□□□()
11     return jours,Tmin,Tmax
12
13 jours,Tmin,Tmax = lecture_fichier('besancon_2013.txt')
```

On fournit par ailleurs la documentation de la fonction ***split()*** *(Voir ANNEXE).*

5-1-Compléter les lignes 2 et 10 (justifier un minimum).

5-2-Pourquoi a-t-on sauté la ligne 12 ?

5-3-Donner le type des variables ***jours***, ***Tmin*** et ***Tmax***, ainsi que le type de ce que retourne la fonction ***lecture_fichier()***.

5-4-Quel est l'effet (préciser) de la ligne 13 ?

5-5-Quel est le but des lignes 4 à 9 ?

6-Ecrire une fonction ***moyenne(a,b)*** qui prend en entrée deux listes ***a*** et ***b*** de mêmes tailles et renvoie une liste de même taille contenant dans la case d'indice i la valeur moyenne des valeurs des flottants stockés dans les deux listes ***a*** et ***b*** à l'indice i. La vérification préalable que ***a*** et ***b*** sont bien deux listes de même taille sera prévue (dans ce cas, la fonction ne renvoie rien, et affiche un message prévenant l'utilisateur de l'erreur). On pourra utiliser la fonction pré-embarquée de Python ***len()***.

7-En utilisant la fonction précédente, écrire l'instruction Python qui stocke dans la variable ***Tmoy*** la liste des températures moyennes journalières à partir des données stockées dans les listes ***Tmin*** et ***Tmax***.

Pour les besoins de l'étude, on utilise la fonction ***mini(t)****, dont on donne le script écrit en Python 3, mais sans l'indentation :*

```
1    def mini(t):
2    '''Calcule le minimum d'un tableau d'entiers ou de flottants.'''
3    if len(t) == 0:
4    return None
5    p = t[0]
6    for i in range(len(t)):
7    if t[i] <= p:
8    p = t[i]
9    return p
```

8-Ecrire le script correctement indenté de la fonction ***mini(t)***.

9-Quel est le rôle de la ligne 2 ?

10-Quel est le rôle de la ligne 5 ?

Pour suivre le déroulement pas à pas lors de l'appel de la fonction ***mini(t)****, on modifie celle-ci, ce qui correspond à la fonction* ***miniprint(t)****. On passe de* ***mini(t)*** *à* ***miniprint(t)*** *en intercalant l'instruction* ***print(i,p)*** *entre les lignes 6 et 7 (cette instruction devient donc la première ligne du bloc* ***for****), et l'instruction* ***print(p)*** *entre les lignes 8 et 9 (celle-ci devient donc la seconde ligne du bloc* ***if****).*

11-Ecrire ce qu'affiche Python 3 en console, à la suite de l'appel ***miniprint([6,2,15,2,15])***.

12-Rappeler la définition d'un invariant de boucle, et les conditions que celui-ci doit vérifier. Puis proposer un invariant de boucle permettant de prouver que la boucle considérée ici renvoie bien, lorsque ***t*** est une liste non vide d'entiers ou de flottants, la valeur minimale des éléments de ***t***.

13-Évaluer la complexité temporelle de l'appel ***mini(t)*** en fonction du nombre n d'éléments de ***t***.

14-Proposer une modification de la fonction ***mini(t)*** permettant d'obtenir la fonction ***maxi(t)*** pour que la valeur renvoyée soit le maximum et non le minimum. On utilisera la numérotation des lignes pour préciser la position des modifications et ainsi éviter de réécrire toute la fonction.

<u>Partie B</u> (Algorithmique et Scilab)

On considère la fonction f définie sur [0,1] par :

$f(x) = \frac{1}{1+x^2}$. f est continue et positive sur $[0\,;\,1]$.

On note C_f la courbe de f dans le plan muni d'un repère orthogonal $\left(O\,;\,\vec{i}\,,\,\vec{j}\right)$

On souhaite estimer $I = \int_0^1 f(x)dx$ par deux méthodes différentes.

A-<u>Méthode des trapèzes</u>

1) Donner une approximation de I en calculant l'aire du trapèze obtenu en remplaçant sur $[0\,;\,1]$ la courbe C_f par un segment de droite.

2) Donner une approximation de I en calculant la somme des aires des deux trapèzes obtenus en subdivisant $[0\,;\,1]$ en deux intervalles $\left(\left[0\,;\,\frac{1}{2}\right]\right.$ et $\left.\left[\frac{1}{2}\,;\,1\right]\right)$ et en remplaçant sur chaque intervalle la courbe C_f par un segment de droite.

3) Donner une estimation de I en calculant la somme des aires des trois trapèzes obtenus en subdivisant $[0\,;\,1]$ en trois intervalles $\left(\left[0\,;\,\frac{1}{3}\right],\left[\frac{1}{3}\,;\,\frac{2}{3}\right]\text{ et }\left[\frac{2}{3}\,;\,1\right]\right)$ et en remplaçant sur chaque intervalle la courbe C_f par un segment de droite.

4) Donner une approximation de I en calculant la somme des aires des n trapèzes obtenus en subdivisant $[0\,;\,1]$ en n intervalles $\left(\left[0\,;\,\frac{1}{n}\right],\left[\frac{1}{n}\,;\,\frac{2}{n}\right],\ldots,\left[\frac{n-1}{n}\,;\,1\right]\right)$ et en remplaçant sur chaque intervalle la courbe C_f par un segment de droite.

5) Compléter l'algorithme ci-dessous qui permet de déterminer une approximation T_n de I par la méthode des trapèzes où le nombre de trapèzes n est choisi par l'utilisateur.

Variables	n, k nombres entiers naturels ; T nombre réel ; f fonction
Entrée	Saisir n
Initialisation	$T \leftarrow$ ..... (1)
Traitement	Pour k allant de 0 à ..... (2)
	$T \leftarrow$ ..... (3)
	Fin boucle pour
Sortie	Afficher T

6) Coder uniquement l'initialisation et le traitement de l'algorithme précédent en **Scilab**.

B-Méthode des points médians

On approche ici I par une somme d'aires de n rectangles obtenus en subdivisant $[0\,;\,1]$ en n intervalles $\left(\left[0\,;\,\frac{1}{n}\right],\left[\frac{1}{n}\,;\,\frac{2}{n}\right],\ldots,\left[\frac{n-1}{n}\,;\,1\right]\right)$ et en remplaçant sur chaque intervalle la courbe C_f par un segment de droite "horizontal" passant par le point de C_f dont l'abscisse est le centre de chaque intervalle.

1) Pour tout entier naturel k compris entre 0 et $n-1$, donner le centre de l'intervalle $\left[\frac{k}{n}\,;\,\frac{k+1}{n}\right]$.

2) En déduire une approximation de I par la méthode des points médians (avec n rectangles)

3) Compléter l'algorithme ci-dessous qui permet de déterminer une approximation M_n de I par la méthode des points médians où le nombre de rectangles n est choisi par l'utilisateur.

Variables	n, k nombres entiers naturels ; M nombre réel ; f fonction
Entrée	Saisir n
Initialisation	$M \leftarrow$ ..... (1)
Traitement	Pour k allant de 0 à ..... (2)
	$M \leftarrow$ ..... (3)
	Fin boucle pour
Sortie	Afficher M

C-Comparaison des deux approximations

La valeur exacte de I est $\frac{\pi}{4}$.

On souhaite comparer l'efficacité des deux méthodes étudiées précédemment.

1) On considère l'algorithme ci-dessous qui détermine le plus petit entier naturel n non nul tel que $\left|T_n - \frac{\pi}{4}\right| < 10^{-p}$ où p est un entier naturel choisi par l'utilisateur.

```
1     VARIABLES
2       n EST_DU_TYPE NOMBRE
3       T EST_DU_TYPE NOMBRE
4       p EST_DU_TYPE NOMBRE
5       k EST_DU_TYPE NOMBRE
6     DEBUT_ALGORITHME
7       LIRE p
8       n PREND_LA_VALEUR 0
9       T PREND_LA_VALEUR 10
10      TANT_QUE (abs(T-Math.PI/4)>=pow(10,-p)) FAIRE
11        DEBUT_TANT_QUE
12        n PREND_LA_VALEUR n+1
13        T PREND_LA_VALEUR 0
14        POUR k ALLANT_DE 0 A n-1
15          DEBUT_POUR
16          T PREND_LA_VALEUR T+(F1(k/n)+F1((k+1)/n))/(2*n)
17          FIN_POUR
18        FIN_TANT_QUE
19      AFFICHER n
20    FIN_ALGORITHME
21
22    Fonction numérique utilisée :
23    F1(x)=1/(1+x*x)
```

a) Expliquer l'affectation de la ligne 9.
b) Expliquer le rôle des lignes 13 à 17

2) On considère l'algorithme qui détermine le plus petit entier naturel n non nul tel que $\left|M_n - \frac{\pi}{4}\right| < 10^{-p}$ où p est un entier naturel choisi par l'utilisateur.

p	1	2	3	4	5
Plus petit entier n tel que $\left\|T_n - \frac{\pi}{4}\right\| < 10^{-p}$	1	3	7	21	65
Plus petit entier n tel que $\left\|M_n - \frac{\pi}{4}\right\| < 10^{-p}$	1	2	5	15	46

Quelle est la méthode la plus efficace pour approximer I ? Justifier.

Partie C (Python 3)

On propose de représenter :
- *un point de coordonnées (x,y) en Python par une liste de deux entiers relatifs :* ***`[x,y]`***
- *plusieurs points (certains peuvent être confondus) par une liste de listes :* ***`[[x1,y1],[x2,y2],…,[xn,yn]]`***
- *un nuage de points par une liste de listes sans répétition, ce qui correspond au modèle de l'ensemble.*

1-Ecrire une fonction ***`membre(p,q)`*** qui renvoie ***`True`*** si le point représenté par ***`p`*** est dans l'ensemble représenté par ***`q`***, et qui renvoie ***`False`*** dans le cas contraire.

2-Ecrire une fonction ***`pivot(p)`*** qui, pour la liste ***`p`*** représentant un nuage de points renvoie cette liste augmentée du point pivot correspondant. On rappelle que le point pivot a pour coordonnées les moyennes des coordonnées des points qui ont servi à l'évaluer.

3-Ecrire une fonction ***`nuage(p)`*** qui, pour ***`p`***, liste de points pouvant être confondus, renvoie la liste du nuage de points correspondant, c'est-à-dire la liste pour laquelle les répétitions ont été éliminées.

4-Ecrire une fonction ***`intersection(p,q)`*** qui renvoie une liste représentant l'intersection des ensembles représentés par ***`p`*** et ***`q`***. On implémentera l'algorithme qui consiste à itérer sur tous les points de ***`p`*** et à insérer dans le résultat ceux qui sont aussi dans ***`q`***.

5-Si la comparaison entre deux entiers est prise comme opération élémentaire, quelle est la complexité de l'algorithme de la fonction ***`intersection(p,q)`***, en fonction de la longueur de ***`p`*** et de celle de ***`q`*** ?

6-Ecrire une fonction ***`linear(p,q)`*** qui renvoie l'ordonnée à l'origine et la pente de la droite passant par les points représentés par ***`p`*** et ***`q`***.

Pour écrire les fonctions demandées, on pourra utiliser les fonctions ou méthodes suivantes : ***`len()`***, ***`append()`***, ***`deepcopy()`*** *du module* ***`copy`***, *ainsi que le test d'appartenance (mot-clé* ***`in`***). *On veillera à éviter les codes obscurs…*

ANNEXE - Partie A
Types alphanumériques

Chaînes de type texte : pour un texte de moins de 255 octets, on peut utiliser les deux types suivants.
VARCHAR(x) où x est une valeur entière : chaîne de caractères dont la longueur maximale est fixée.
CHAR(x) : chaîne de x caractères, de longueur fixe.
CHAR et **VARCHAR** diffèrent par la façon dont le stockage se fait en mémoire : **CHAR(x)** stockera toujours *x* octets, en remplissant si nécessaire le texte avec des espaces vides pour le compléter, tandis que **VARCHAR(x)** stockera le texte dans un nombre d'octets qui pourra aller jusqu'à *x*, mais stockera aussi la taille du texte stocké.

Comparaison entre CHAR(5) et VARCHAR(5) pour stocker le même texte				
Texte	**CHAR(5)**	Nombre d'octets requis	**VARCHAR(5)**	Nombre d'octets requis
'' (rien)	***' '***	5	***''***	1
'tex'	***'tex '***		***'tex'***	4
'texte'	***'texte'***		***'texte'***	6
'texte trop long'	***'texte'***		***'texte'***	6

```
split(...)
    S.split(sep=None, maxsplit=-1) -> list of strings
Return a list of the words in S, using sep as the delimiter string.  If maxsplit is given,
at most maxsplit splits are done. If sep is not specified or is None, any whitespace string
is a separator and empty strings are  removed from the result.
```

MPSI 2 **DS 2**

Informatique *Le 23/2/19*

CORRECTION

Partie A

1-La clé primaire d'une table est constituée le plus souvent d'un attribut, ou éventuellement de plusieurs, permettant d'identifier chaque enregistrement de la table de façon unique. L'attribut *insee* peut jouer ce rôle, car deux villes différentes ont des numéros INSEE différents. En revanche, deux villes différentes de régions différentes peuvent porter le même nom, se situer dans le même département, se situer à la même latitude ou à la même longitude, ou (par hasard), avoir le même nombre d'habitants : aucun de ces attributs ne peut donc servir de clé primaire. Remarque : la latitude et la longitude définissant un point unique sur le globe, les deux attributs lat et lon peuvent servir de clé primaire pour la table villes. Ceci dit, on préférera choisir un attribut unique.

Pour la table mesures, aucun attribut seul ne peut jouer le rôle de clé primaire. En revanche, les attributs villes et jours oui, car chaque jour, on enregistre la mesure de Tmin et de Tmax pour chaque ville.

2-Le code postal contient un nombre fixe de caractères : le type CHAR, qui permet de stocker un nombre fixe de caractère, est donc le mieux adapté. Dans ce cas, ce sera CHAR(5). Notons qu'il y a ici peu de différence avec VARCHAR(5) en termes de stockage : ce format pourrait convenir aussi.

3-La première requête affiche le nom, le département et le nombre d'habitants de toutes les villes répertoriées dans la table *villes*, ces lignes étant classées par valeurs décroissantes de population.

La seconde requête affiche tous les noms des villes répertoriées dans la table villes, classées par ordre alphabétique, sans doublons. On notera que les deux listes ont vraisemblablement des nombres de lignes différents (il peut y avoir des villes de même nom dans des départements différents, par exemple).

4-a-SELECT nom FROM villes WHERE dpt = `Yvelines` (= ou LIKE)

4-b-SELECT COUNT(*) AS grandesVilles FROM villes WHERE pop > 100 000

4-c-SELECT MAX(Tmin),AVG(Tmin) FROM mesures

5-1-Ligne 2 : r (pour read) et ligne 10 : close (il faut fermer le fichier, ouvert à la ligne 2)

5-2-La ligne 12 sautée permet de séparer la déclaration de la fonction du programme principal (la ligne 13). Cela ne change rien à l'exécution, cela sert simplement à rendre le code plus lisible.

5-3-La ligne 3, qui permet d'initialiser les trois variables en question montre qu'il s'agit de trois listes. La fonction retourne un tuple contenant ces trois listes (parenthèses facultatives).

5-4-La ligne 13 permet de faire l'appel de la fonction, et d'obtenir ainsi un résultat à l'exécution du programme.

5-5-Ces lignes permettent de récupérer le contenu du fichier pour constituer les listes jours, Tmin et Tmax, grâce à la fonction split().

6-
```
def moyenne(a,b) :
    if len(a)!=len(b) :
        print('Attention, les deux listes ne sont pas de même taille')
        return None
    else :
        m=[]
        for i in range(len(a)) :
            m=m+(a[i]+b[i])/2
        return m
```
Des variantes sont possibles, bien entendu.

7->> *Tmoy=moyenne(Tmin,Tmax)*

8-Indentation correcte de la fonction *mini(t)* :

```
1  def mini(t):
2      """Calcule le minimum d'un tableau d'entiers ou de flottants."""
3      if len(t) == 0:
4          return None
5      p = t[0]
6      for i in range(len(t)):
7          if t[i] <= p:
8              p = t[i]
9      return p
```

9-La ligne 2 permet de documenter la fonction.

10-La ligne 5 permet d'initialiser la variable p (avec la valeur du premier élément de la liste t).

11-Affichage obtenu à l'appel de la fonction *miniprint()* (on pouvait justifier en faisant un tableau) :

```
>>> miniprint([6,2,15,2,15])
0 6
6
1 6
2
2 2
3 2
2
4 2
2
```

```
X-TPP13.py   <tmp 1>   TPP13.py
1  def miniprint(t):
2      """Calcule le minimum d'un tableau d'entiers ou de flottants."""
3      if len(t) == 0:
4          return None
5      p = t[0]
6      for i in range(len(t)):
7          print(i,p)
8          if t[i] <= p:
9              p = t[i]
10             print(p)
11     return p
```

12-Un invariant de boucle (E) est une expression qui permet de calculer une solution partielle à chaque itération de la boucle, ce qui permet d'atteindre le but recherché lorsque toutes les boucles nécessaires ont été exécutées. (E) doit vérifier les conditions : (E) vraie à l'entrée dans la première boucle, (E) vraie pendant chaque nouvelle itération et (E) vraie à la sortie de la dernière boucle. Ici, ce sont les lignes 7 et 8 qui jouent le rôle d'invariant de boucle.

13-Pour une liste de n éléments, il y aura n boucles exécutées : le programme est donc d'ordre O(n) autres instructions d'effet négligeable en termes de complexité).

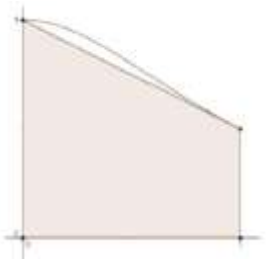
14-Il suffit de remplacer dans la ligne 1 : mini par maxi
dans la ligne 2 : minimum par maximum
dans la ligne 7 : <= par >=

Sujet B

I-

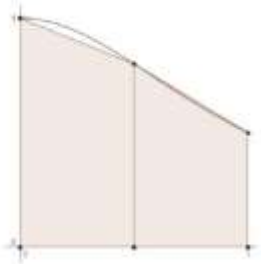
1)

$I \approx \frac{(f(0)+f(1))\times 1}{2}$ donc $I \approx \frac{3}{4}$



2)

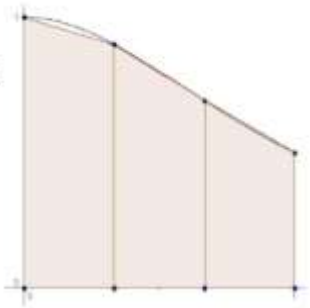
$I \approx \frac{\left(f(0)+f\left(\frac{1}{2}\right)\right)\times\frac{1}{2}}{2} + \frac{\left(f\left(\frac{1}{2}\right)+f(1)\right)\times\frac{1}{2}}{2}$ donc $I \approx \frac{31}{40}$



3)

$$I \approx \frac{\left(f(0)+f\left(\frac{1}{3}\right)\right)\times\frac{1}{3}}{2} + \frac{\left(f\left(\frac{1}{3}\right)+f\left(\frac{2}{3}\right)\right)\times\frac{1}{3}}{2} + \frac{\left(f\left(\frac{2}{3}\right)+f(1)\right)\times\frac{1}{3}}{2}$$

donc $I \approx \frac{203}{260}$



4) approximation de I

$$I \approx \frac{1}{2n}\sum_{k=0}^{n-1}\left(f\left(\frac{k}{n}\right)+f\left(\frac{k+1}{n}\right)\right)$$

5)

Variables	n, k nombres entiers naturels ; T nombre réel ; f fonction
Entrée	Saisir n
Initialisation	$T \leftarrow$ [0] (1)
Traitement	Pour k allant de 0 à [$n-1$] (2)
	$T \leftarrow$ [$T + \frac{f\left(\frac{k}{n}\right)+f\left(\frac{k+1}{n}\right)}{2n}$] (3)
	Fin boucle pour
Sortie	Afficher T

II Méthode des points médians

1) Pour tout entier naturel k compris entre 0 et $n-1$, centre de l'intervalle $\left[\frac{k}{n} ; \frac{k+1}{n}\right]$

$$\frac{\frac{k}{n}+\frac{k+1}{n}}{2}=\frac{2k+1}{2n}$$

2) approximation de I par la méthode des points médians (avec n rectangles)

$$I \approx \frac{1}{n}\sum_{k=0}^{n-1} f\left(\frac{2k+1}{2n}\right)$$

3)

Variables	n, k nombres entiers naturels ; M nombre réel ; f fonction
Entrée	Saisir n
Initialisation	$M \leftarrow$ 0 (1)
Traitement	Pour k allant de 0 à $n-1$ (2)
	$M \leftarrow M + \frac{f\left(\frac{2k+1}{2n}\right)}{n}$ (3)
	Fin boucle pour
Sortie	Afficher M

III Comparaison des deux approximations

1) a) Au début de l'algorithme, on affecte 10 à la variable T pour être sûr de "rentrer" dans la boucle tant que (car $\left|10-\frac{\pi}{4}\right| \geq 10^{-p}$ pour tout entier naturel p)

b) Dans les lignes 13 à 17, on calcule T_n.

2) La méthode des points médians est la plus efficace dans ce cas. (même précision pour moins d'itérations)

Partie C

Des variantes possibles :

1-

```
def membre(p,q):
    return p in q
```

2-

```
def pivot(p):
    from copy import deepcopy
    pp=deepcopy(p)
    x=0
    y=0
    for point in pp:
        x=x+point[0]
        y=y+point[1]
    pp.append([x/len(p),y/len(p)])
    return pp
```

3-

```
def nuage(p):
    N=[]
    for x in p:
        if x not in N:
            N.append(x)
    return N
```

4-

```
def intersection(p,q):
    intersection=[]
    for x in p:
        if x in q:
            intersection.append(x)
    return intersection
```

5-complexité : O(len(p).len(q)) (boucles imbriquées avec la boucle for on itère sur p, avec le test if, on itère au pire sur q)

6-

```
def linear(p,q):
    xp,yp=p[0],p[1]
    xq,yq=q[0],q[1]
    if xp==xq:
        return None
    else:
        pente=(yq-yp)/(xq-xp)
        return pente,yp-pente*xp
```