Conseils :

- Prenez le temps de bien faire les choses et rendez une copie propre. Des points seront mis pour le soin (environ 1,5 points sur 20). Le fait de commenter ses fonctions pour les expliquer sera aussi évalué.
- Aucun ordre n'est imposé pour la résolution, par contre, rendez les problèmes dans l'ordre. Numérotez avec rigueur les questions que vous traitez.
- Même si vous n'avez pas sû faire une question, vous pouvez utiliser la fonction que vous étiez censé coder pour répondre aux questions suivantes.
- Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- L'usage des **calculatrices est interdit**.

L'usage de fonctions de python non vues en cours telles que min/max/sum n'est pas autorisé sauf mention contraire.

I. Suppression d'objets sur une photo

Il arrive parfois que l'on souhaite prendre une photo d'un monument ou d'un paysage, mais que des personnes ou des véhicules passent sans cesse. On peut souhaiter avoir la photographie sans ces éléments mouvant.
Dans cette énoncé, on prendra l'exemple du poisson se déplaçant dans les images ci-dessous. On cherchera à la fois à récupérer "l'image sans le poisson", mais aussi les images avec uniquement le poisson, ce qui peut être utile pour du suivi d'objet en mouvement. Dans un premier temps on envisagera un algorithme simulant une technique physique utilisée depuis des décennies, puis une technique qui ne serait pas (facilement) possible sans utilisation de l'ordinateur. Vous verrez l'an prochain l'algorithme des K-plus proches voisin qui est aussi employé pour ce type d'usage.



On utilisera la bibliothèque numpy que l'on supposera préalablement importée avec la syntaxe `import numpy as np`. On rappelle que l'on peut créer un tableau n lignes et p colonnes rempli de 0 avec la syntaxe `np.zeros( (n, p))`. On travaillera avec des images en niveau de gris pour plus de simplicité (une valeur entre 0(noir) et 255(blanc) représente la luminosité du pixel). Si `Im` est une variable représentant une image, alors on peut accéder à la valeur du pixel à la ligne l et à la colonne c en utilisant la syntaxe `Im[l,c]` ou `Im[l][c]`. Pour tout ce qui nous concernera, un tableau numpy pourra être manipulé comme une liste de liste de nombre.

1 Prétraitement

La durée des traitements d'images pouvant être assez conséquente, on souhaite dans un premier temps diminuer la taille de l'image. Pour cela, on propose de remplacer des groupes de 2x2 pixels par leur moyenne. Ainsi, par exemple, les pixels (0,0),(0,1),(1,0) et (1,1) seront remplacés par un seul pixel de coordonnées (0,0) dans la nouvelle image. On divise donc le nombre de pixels par 4.

```
1 def reduitNB(im):#suppose une image N & B
2   #im : tableau numpy 2D représentant une image
3   nb_lig = len(???) #nombre de lignes
4   nb_col = len(???) #nombre de colonnes
5   res = np.zeros( (nb_lig//2, nb_col//2) ) #tableau vide
6   for i in range(0, nb_lig, 2): #que fait range(0,nb_lig,2) ?
7     for j in range(0,nb_col,2):
8       res[???,???] = (im[i,j]+im[i+1,j]+im[i,j+1]+im[i+1,j+1])*0.25
9   return res
```

Q1 Dans le code ci-dessus, que "permet" `range(0, nb_lig, 2)` ? Compléter les "???" lignes 3, 4 et 8.

Q2 Si l'on dispose d'une image enregistrée dans un tableau im0, comment en une ligne peut-on maintenant diviser le nombre de pixels de l'image par **16** ?

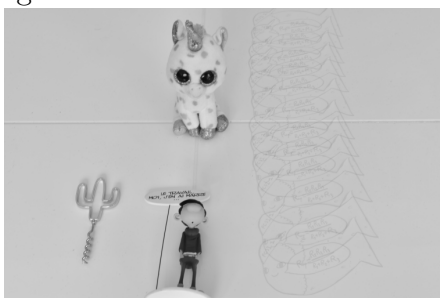
2 Avec un long temps d'exposition !

Une technique permettant de supprimer les objets en déplacement est de faire une photo avec un temps d'exposition très long. Pour faire de même informatiquement, on se propose de moyenner différentes images. On travaillera dans la suite de l'énoncé avec une liste d'image, chaque image ayant exactement les mêmes dimensions. On supposera pour simplifier deux variables globales définies, `nb_lig` et `nb_col` représentant respectivement le nombre de lignes et celui de colonnes dans chaque image.

Q3 Que signifie le terme "variable globale" ? À quoi est-il opposé ?

Q4 Écrire une fonction `moyenneImage(liste_im)` qui prend en argument une liste d'images et qui renvoie une image qui est la "moyenne" des images. C'est-à-dire que pour une position (i,j) donnée, la valeur du pixel du résultat est la moyenne des valeurs des pixels de position (i,j) dans les images de `liste_im`.

On obtient alors une image telle que celle ci-dessous où l'objet en mouvement laisse une trainée, faiblement visible mais visible malgré tout.



3 Avec un histogramme

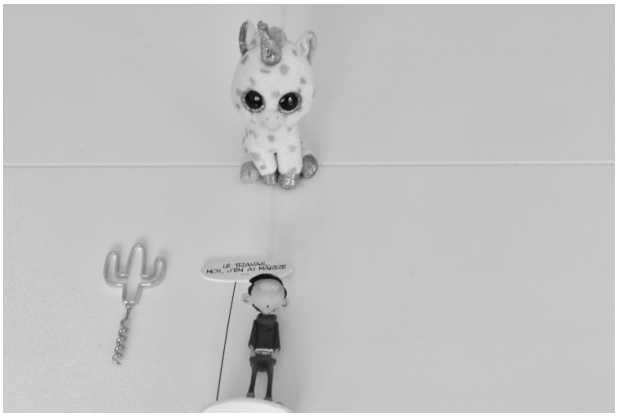
On se propose de plutôt compter le nombre de fois où un pixel a eu une valeur (entre 0 et 255). Pour cela, on va créer une liste res de 256 nombre telle que $res[i]$ est le nombre de fois où apparait la valeur i sur le pixel donné dans les différentes images. Par exemple, si dans les différentes images le pixel considérées prend les valeurs 0, 0, 1, 0, 2, 0, alors res = [4,1,1,0,0,0,...]

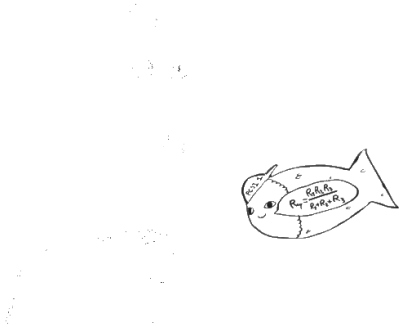
```
1 def histo(liste_im, lig, col):
2     #réaliste l'histogramme pour le pixel de coord lig,col
3     res = ??? #liste de 256 zéros
4     for im in liste_im:
5         res[ ???? ] += 1
6     return res
```

Q5 Comment compléter le code ci-dessus pour obtenir la fonction `histo` réalisant l'histogramme souhaité pour un pixel de coordonnées `lig,col` dans la liste d'image `liste_im` ?

Q6 Pour obtenir l'image sans les objets en mouvement, il suffit maintenant de regarder la valeur la plus fréquente. Écrire une fonction `ind_max(liste)` qui, étant donné une liste, donne **l'indice** du maximum. Dans le cas où le maximum est présent plus d'une fois, on souhaite obtenir le plus grand des indices correspondant. La fonction devra être en temps linéaire par rapport à la taille de la liste et une brève justification de sa complexité est attendue.

Q7 En utilisant les fonctions précédentes, écrire une fonction `backgroundImage(liste_im)` qui prend en argument une liste d'image et qui renvoie une image correspondant à la partie immobile de l'image. (Résultat ci-dessous à gauche)





Pour obtenir la partie en mouvement de l'image (ci-dessus à droite), on propose de regarder l'écart entre la valeur d'un pixel et la valeur de l'image trouvée à la fonction précédente. Lorsque cet écart est supérieur en valeur absolue à un seuil, on considère qu'un objet en mouvement est présent sur le pixel. Il faut donc créer une image vide (on utilisera `np.zeros((nb_lig,nb_col))+255` pour avoir un fond blanc) puis, pour chaque image, pour chaque pixel regarder l'écart et modifier le pixel en conséquence si cela est pertinent.

```
def mouvement(liste_im,seuil):
  fond = ??? #image correspondant à la partie immobile
  res = [] #liste d'image resultat
  for im in liste_im:
    temp = np.zeros((nb_lig,nb_col))+255 #image toute blanche
    for lig in range(nb_lig):
      for col in range(nb_col):
        if ???? :
          temp[lig,col] = ???
    res.append(temp)
  return res #liste d'image
```

Q8 Compléter le code ci-dessus pour obtenir la fonction `mouvement` qui, étant donné une liste d'image et un seuil, renvoie la liste contenant (pour chaque image) la partie en mouvement.

II. Représentation des nombres

Dans ce problème, pour chaque question **le raisonnement détaillé est attendu** (et pas seulement le résultat). On donne les puissances de deux suivantes :

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
2048	1024	512	256	128	64	32	16	8	4	2	1	0.5	0.25	0.125

A. Nombres entiers

Dans toute cette partie, la représentation des nombres sera celle du complément à 2 (c'est-à-dire celle vue en cours) et les nombres seront codées sur 8 bits.

Q9 ➢ Quelle est la représentation en mémoire du nombre 46 ?

Q10 ➢ Quelle est la représentation en mémoire du nombre -46 ?

Q11 ➢ Quelle est le nombre représenté par la suite de bit 1010 0010 ?

Q12 ➢ Quelle est le nombre représenté par la suite de bit 0001 0111 ?

Q13 ➢ Quel est, avec une telle représentation, le résultat du calcul $120 + 30$? Quel nom porte cette erreur ?

B. Nombres à virgule flottante

On considère des flottants 64 bits. Si un grand nombre de 0 apparait, il sera possible de ne pas tous les écrire mais d'indiquer à l'aide d'une accolade le nombre de 0. On rappelle les règles suivantes pour un nombre flottant codé sur 64 bits :

- l'exposant e est codé sur 11 bits et est représenté en base 2 par l'entier $n = e + 1023$;
- la mantisse est codée sur 52 bits.

Q14 ➢ Quel nombre est représenté par la suite de bits ci-dessous ? (le résultat sera donné sous forme décimale en base 10).
1 10000000101 0110000000000000000000000000000000000000000000000000

Q15 ➢ Quelle est la représentation du nombre 35,0 ?

Q16 ➢ À partir de quel entier $k > 2$ python renverra-t-il un résultat faux pour le calcul suivant : $0.25 + 2 * *(-k)$? Quel sera alors le résultat ? Quelle est le nom de cette erreur ?

Q17 ➢ À partir de quel entier $k > 2$ python renverra-t-il un résultat faux pour le calcul suivant : $2 * *(-10 * k)$? Quel sera alors le résultat ? Quelle est le nom de cette erreur ?

III. Voyageur de commerce

Le problème du voyageur de commerce est un problème classique d'optimisation en informatique. Il s'agit de trouver le chemin le plus court pour qu'un voyageur de commerce visite un ensemble de villes **et** revienne à son point de départ, sans visiter deux fois la même ville. De façon plus générale, ce problème peut s'appliquer à d'autres cas très concrets, comme le choix d'un itinéraire pour un livreur avec de nombreux colis.
On supposera avoir l'ensemble des villes sous forme de liste par exemple : `villes = ["Biarritz", "Bordeaux","Brest","Dijon","Grenoble"]`.
De plus, on dispose d'une "matrice" (liste de liste) distances, telle que `distance[i][j]` est la distance entre la ville d'indice i et celle d'indice j (en km). Par exemple, `distance[1][2]` est la distance entre la ville d'indice 1 ("Bordeaux") et la ville d'indice 2 ("Brest") soit 648 km.
Dans notre exemple, la matrice distance pourrait donc être représentée par la variable ci-dessous.

```
distance = [[  0, 182, 830, 918, 822],
            [182,   0, 648, 619, 675],
            [830, 648,   0, 793,1050],
            [918, 619, 793,   0, 279],
            [822, 675,1050, 279,   0]]
```

Q18 Le voyageur décide de partir de Biarritz, quel est le nombre de trajets possibles ? Quelle est la longueur du trajet ["Biarritz","Bordeaux","Brest","Dijon","Grenoble","Biarritz"]

Q19 Dans le cas général de n endroits à visiter, l'endroit de départ étant fixé, quel est le nombre de trajet possibles ?

Q20 On se propose d'utiliser un algorithme glouton pour trouver un trajet. Quels sont les avantages et inconvénients par rapport à un algorithme dit "force brute" qui testerait toutes les possibilités ?

Dans notre cas, la stratégie gloutonne consistera à :

1. choisir une ville de départ ;
2. visiter la ville la plus proche non encore visitée ;
3. répéter l'étape précédente tant que toutes les villes n'ont pas été visitées ;
4. revenir au point de départ.

Q21 Toujours avec Biarritz comme ville de départ, appliquer à la main l'algorithme glouton proposé et donner le trajet sous forme d'une liste de ville. Quelle serait la distance parcourue ? Commenter.

Q22 Écrire une fonction `prochaine_ville(ind, chemin, distance)` qui prend en argument :
- ➢ un entier `ind` qui est l'indice de la ville actuelle ;
- ➢ une liste `chemin` qui contient la liste des indices des villes déjà visitée (la dernière valeur de cette liste est donc `ind`) ;
- ➢ la matrice `ditance` telle que définie au début du problème ;

et qui renvoie l'indice de la prochaine ville à visiter en respectant l'algorithme glouton.

Q23 Écrire une fonction `trajet(ind_d, distance)` qui prend en argument un entier `ind_d` représentant l'indice de la ville choisie pour le départ et la matrice `distance` et qui renvoie :
- ➢ une liste contentant le chemin sous la forme des indices des villes à visiter, par exemple [0,1,2,3,4,0] pour le chemin ["Biarritz","Bordeaux","Brest","Dijon","Grenoble","Biarritz"] ;
- ➢ et la distance correspondante.

Q24 Déterminer, en la justifiant, la complexité en temps de votre fonction `trajet` en fonction de n le nombre de villes à visiter. On utilisera la notation $\mathcal{O}$.

I. Suppression d'objets sur une photo

1 Prétraitement

```
def reduitNB(im):#suppose une image N & B
  #im : tableau numpy 2D représentant une image
  nb_lig = len(im) #nombre de lignes
  nb_col = len(im[0]) #nombre de colonnes
  res = np.zeros( (nb_lig//2, nb_col//2) ) #tableau vide
  for i in range(0, nb_lig, 2):
    for j in range(0,nb_col,2):
      res[i//2,j//2] = (im[i,j]+im[i+1,j]+im[i,j+1]+im[i+1,j+1])*0.25
  return res
```

Q1 Dans le code ci-dessus, `range(0, nb_lig, 2)` permet d'aller de 0 à `nb_lig-1` par pas de 2. Voir ci-dessus pour le code complété.
On a bien $i//2$ et $j//2$ à la fin, puisque l'on divise le nombre de pixel par 4, la moyenne des pixels (0,0),(1,0),(0,1) et (1,1) donne le tout premier pixel (0,0). De même 4 pixels en dessous, c'est-à-dire (2,0),(2,1),(3,0),(3,1) donne le pixel (1,0). De façon plus général, les 4 pixels (i,j) ; (i+1,j) ; (i,j+1) et (i+1,j+1) moyenné vont donner le pixel $\boxed{(i//2, j//2)}$.

Q2 On peut faire simplement `im0 = reduitNB(reduitNB(im))`. Vous avez été nombreux à proposer de faire à nouveau *0.25 ou /4, mais cela change (éventuellement) la valeur des pixels, pas leur nombre (et cela n'a pas de sens sur une liste). Par exemple, si l'on passe de [4,4,4,4] à [1,1,1,1], on a rendu l'image plus sombre, mais on n'a pas enlevé de pixels.

2 Avec un long temps d'exposition !

Q3 Le terme variable globale est opposé à variable locale. Dans le cas d'une variable globale, elle est accessible dans tout le programme alors qu'une variable locale n'est accessible que dans la fonction où elle est définie [1]. Remarque, vous avez parfois employé le vocabulaire "on défini la variable globale dans ...". En fait, elle peut être défini en dehors ou dans une fonction la première fois. La différence principale étant que sa portée dépasse une fonction. De plus, certains ont confondus avec "constante". Une variable globale n'est pas forcément constante, nous en avons d'ailleurs utilisé en TP pour compter le nombre d'appels récursifs.

Q4 Fonction `moyenneImage(liste_im)` qui prend en argument une liste d'images et qui renvoie une image qui est la "moyenne" des images. Ci-dessous, pour chaque pixel on fait la moyenne sur les différentes images. [2]

```
def moyenneImage(liste_im):
    nb_lig,nb_col = len(liste_im[0]), len(liste_im[0][0])
    res = np.zeros((nb_lig,nb_col)) #image resultat
    for lig in range(nb_lig):
```

1. dans certains langages, on peut avoir une variable locale à une boucle for, et pas seulement à une fonction.
2. À notre niveau, l'ordre dans lequel on fait les boucles n'est pas très important, mais en terme d'accès mémoire et de temps d'exécution, il faut avoir conscience que l'ordre des boucles n'est pas anodin à cause de la gestion de la mémoire et de la mise en cache dans le processeur.

```
        for col in range(nb_col):
            for im in liste_im:
                res[lig,col] += im[lig,col]
            res[lig,col] = res[lig,col]/len(liste_im)
    return res
```

3 Avec un histogramme

```
def histo(liste_im, lig, col):
    #réaliste l'histogramme pour le pixel de coord lig,col
    res = [0]*256 #liste de 256 zéros
    for im in liste_im:
        res[ im[lig,col] ] += 1
    return res
```

Q5 Ci-dessus le code de la fonction `histo`. Remarque, il était bien demandé une liste et non un tableau numpy.

Q6 Fonction `ind_max(liste)`

```
def ind_max(liste):
    ind = 0
    maxi = liste[0]
    for i in range(len(liste)):
        if liste[i]>=maxi: #si on précise qu'on prend le dernier max,
            > si on prend le premier
            ind = i
            maxi = liste[i]
    return ind
```

La complexité est bien linéaire car on a une simple boucle for constituée d'opérations élémentaires. Remarque : il était demandé de prendre l'indice du dernier max, il faut donc justifier que votre programme respecte la contrainte. Ici, en mettant >=, on s'assure que si on retombe sur la valeur du maximum, on met à jour l'indice.

Q7 Fonction `backgroundImage(liste_im)`

```
def backgroundImage(liste_im):
    nb_lig, nb_col = len(liste_im[0]), len(liste_im[0][0])
      #ligne ci-dessus inutile pour vous car variables globales par
         hypothèses
    res = np.zeros((nb_lig,nb_col)) #image resultat
    for lig in range(nb_lig):
        for col in range(nb_col):
            res[lig,col] = ind_max(histo(liste_image,lig,col))
    return res
```

```
def mouvement(liste_im,seuil):
    fond = backgroundImage(liste_im)
    nb_lig, nb_col = len(liste_im[0]), len(liste_im[0][0])
    res = [] #liste d'image
```

```
    for im in liste_im:
        temp = np.zeros((nb_lig,nb_col))+255 #image temp toute
           blanche
        for lig in range(nb_lig):
            for col in range(nb_col):
                if abs(im[lig,col]-fond[lig,col]) >= seuil:
                    temp[lig,col] = im[lig,col]
        res.append(temp)
    return res
```

Q8 Code ci-dessus pour la fonction `mouvement`.

II. Représentation des nombres

Il était explicitement demandé de justifier, il faut donc le faire ! Un bon principe est : "pas de justification = pas de point".

Justifier a plusieurs avantages et un léger inconvénient :

- ➢ inconvénient : vous allez légèrement moins vite (mais de toutes façons il faut faire le raisonnement pour avoir la réponse)
- ➢ avantages : vous oblige à bien faire le raisonnement et permet donc de faire moins de fautes ; si vous avez faux pour une petite faute de calcul, on peut mettre une partie des points pour le raisonnement, ce que l'on ne peut pas faire s'il n'y a que la réponse.

A. Nombres entiers

- ➢ $0 \leqslant 46 < 128$ donc le nombre est directement représenté par son écriture en base 2 (en rajoutant les 0 à gauche pour être sur 8 bits) : $46_{10} = 32+8+4+2 = 2^5+2^3+2^2+2^1 = 101110_2$

Q9 donc le nombre est représenté par $\boxed{0010\,1110}$.

- ➢ $-128 \leqslant -46 < 0$ donc le nombre est représenté par l'écriture en base 2 du nombre $-46+256 = 210$. Or $210 = 128+64+16+8+4 = 2^7+2^6+2^4+2^1 = 1101\,0010_2$ donc le nombre est

Q10 représenté par $\boxed{1101\,0010}$.

- ➢ $1010\,0010$ est l'écriture en binaire du nombre $128+32+2 = 162$. Ce nombre étant plus grand

Q11 que 128, il représente l'entier négatif $162 - 256 = \boxed{-94}$.

- ➢ $0001\,0111$ est l'écriture binaire du nombre $2^4+2^2+2^1+2^0 = 23$, ce nombre étant inférieur strictement à 128 il n'y a pas d'autres modifications à faire et le nombre représenté est directement

Q12 $\boxed{23}$.

- ➢ $120 + 30 = 150 > 128$ On a donc un $\boxed{overflow}$ puisque l'on sort de l'intervalle des nombres représentables sur 8 bits par la méthode du complément à 2. Le nombre sera donc interprété

Q13 comme $150 - 256 = \boxed{-106}$.

B. Nombres à virgule flottante

- ➢ $\underbrace{1}_{s}$ $\underbrace{10000000101}_{\text{exposant}}$ $\underbrace{01100000000000000000000000000000000000000000000000}_{\text{mantisse}}$

En utilisant les notations du cours :

1. $s = 1$ donc le signe est $-$;

2. $n = 10000000101_2 = 1024 + 4 + 1$ donc $e = n - 1023 = 6$;
3. $m = 1{,}011_2$ (attention de ne pas oublier que seuls les chiffres après la virgule sont représentés et qu'il y a forcément un 1 avant) ;

Q14 Le nombre est donc $-1{,}011_2 \times 2^6 = -(1+2^{-2}+2^{-3}) \times 2^6 = -(2^6+2^4+2^3) = -(64+16+8) =$ $\boxed{-88{,}0}$.

➢ $35{,}0 = 32 + 2 + 1 = 2^5 + 2^1 + 2^0 = 2^5 \times (1 + 2^{-4} + 2^{-5})$

1. le signe est + donc $s = 0$;
2. L'exposant est $e = 5$ donc il sera codé par $n = 1023 + e = 1028 = 100\,0000\,0100_2$;
3. $m = 1{,}00011000\ldots_2$ et donc sera codé par $00011000\ldots$ (attention de ne pas oublier que seuls les chiffres après la virgule sont représentés et qu'il y a forcément un 1 avant) ;

Q15 La représentation est donc
$\boxed{0 \quad 100\,0000\,0100 \quad 00011\,\underbrace{0\ldots0}_{47 \text{ fois } 0}}$.

Q16 ➢ Il s'agit d'une erreur $\boxed{\text{d'arrondi}}$. Elle apparait lorsque le nombre de bit alloué à la mantisse n'est pas suffisant pour représenter exactement le résultat. $0.25 = 2^{-2}$ et donc $2^{-2} + 2^{-k} = 2^{-2} \times (1 + 2^{-k+2})$. Ainsi ce nombre a pour exposant -2 et pour mantisse $1 + 2^{-k+2}$. La mantisse étant codée sur 52 bits, il n'y aura aucun problème tant que $-k + 2 \geqslant -52$. On va commencer à avoir une erreur à partir de $-k + 2 = -53$, soit $\boxed{k = -55}$. À partir de là, le résultat du calcul sera égal à 0,25 par arrondi [3].

Q17 ➢ Il s'agit ici d'un $\boxed{\text{underflow}}$ ou soupassement arithmétique. Il apparait lorsque l'exposant est trop petit pour être représenté [4]. Soit pour $e = -1022 (n = 1)$. Dans notre cas, ce sera donc à partir de $k = 103$.

III. Voyageur de commerce

Q18 En partant de Biarritz, le nombre de trajets possibles est $\boxed{4! = 24}$. En effet, en partant de la première ville, on a 4 choix, puis pour continuer le chemin 3 choix (car on ne doit pas passer deux fois par la même ville), puis 2 puis 1,puis on revient au départ. On compte parfois comme étant identique les trajets dans les deux sens car ils sont de même longueurs et on divise donc par 2 le résultat précédent. Le trajet ["Biarritz","Bordeaux","Brest","Dijon","Grenoble","Biarritz"] est de longueur $182 + 648 + 793 + 279 + 822 =$ $\boxed{2724\,\text{km}}$.

Q19 Dans le cas général de n endroits à visiter, l'endroit de départ étant fixé, on a $\boxed{(n-1)!/2}$ chemins possibles. La réponse $(n-1)!$ sera aussi acceptée.

Q20 On se propose d'utiliser un algorithme glouton pour trouver un trajet. Les avantages sont que cela fournit un résultat $\boxed{\text{rapidement}}$ car on n'explore pas toutes les possibilités, mais on ne garantit pas (sauf cas spéciaux) le fait d'avoir un résultat exactement optimal [5].

Q21 Avec Biarritz comme ville de départ, la ville la plus proche est Bordeaux (182), puis de Bordeaux on va à Dijon (619), puis à Grenoble (279), Brest (1050) et enfin on revient à Biarritz (830). La distance

3. à part peut-être pour k = -55 où un arrondi pourrait être fait au dessus, mais les règles d'arrondis ne sont pas à notre programme.

4. Remarque, c'est en fait un peu plus compliqué grâce aux nombres dénormalisées qui permettent de descendre un peu plus bas, mais au prix d'une mantisse plus réduite. Ce point n'étant pas à notre programme, ce n'était pas attendu de vous.

5. Remarque : pour certains problèmes, on peut faire en sorte que l'algorithme glouton fournisse un résultat optimal (cas du rendu de monnaie vu en TP avec un système de pièce "canonique", c'est-à-dire 1-2-5-10-20-50-...)

parcourue est de $\boxed{2960\,\text{km}}$. Comme dit précédemment, ce résultat n'est pas optimal car plus grand que le chemin proposé au début.
Pour le test de si "i est dans chemin" ou pas. On peut coder une fonction annexe ou utiliser la syntaxe python dédiée, mais il faut dans les deux cas avoir conscience que cette recherche n'est pas une opération éléme

Q22 Fonction `prochaine_ville(ind, chemin, distance)` ci-dessous. On suppose qu'il reste une ville à tester et donc que $len(chemin) < nb_ville$, sinon c'est qu'il faut "rentrer" à la ville de départ. La difficulté est ici l'initialisation du minimum et le fait de ne pas reprendre les villes dans chemin. On peut éventuellement "tricher" en écrivant mini = float("inf") comme vu en cours sur les graphes, on s'assure alors que la condition $distance[ind][i] < mini$ sera vrai une fois. Je n'ai pas utilisé cette "astuce" dans le corrigé parce que je trouve intéressant de savoir faire sans.

Q23 Fonction `trajet(ind_d, distance)` ci-dessous. On initialise le trajet au point de départ, puis on cherche la ville suivante, on l'ajoute à chemin, et on recommence à partir de cette ville. Ainsi, à chaque fois on part de la dernière ville dans la liste chemin, d'où le chemin[-1] dans l'argument de *prochaine_ville*.
J'ai choisi de calculer la distance totale après, mais on peut le faire dans la même boucle for si on le souhaite.

```
def prochaine_ville(ind, chemin, distance):
    nb_ville = len(distance)
    non_visite = [] #liste des villes non explorées
    for i in range(nb_ville):
        if not i in chemin:
            non_visite.append(i)
    mini = distance[ind][non_visite[0]]
    ind_mini = non_visite[0]
    for i in non_visite:
        if distance[ind][i] < mini:
            ind_mini = i
            mini = distance[ind][i]
    return ind_mini

def trajet(ind_d, distance):
    chemin = [ind_d]
    nb_ville = len(distance)
    for i in range(nb_ville-1):
        chemin.append(prochaine_ville(chemin[-1], chemin, distance))
    chemin.append(ind_d) #retour au départ
    #
    dist = 0
    for i in range(nb_ville):#chemin est de taille nb_villes + 1
        dist += distance[chemin[i]][chemin[i+1]]
    return chemin, dist

#test (non demandé)
test,dist = trajet(0, distance)
print("dist=",dist)
for i in test:
    print(villes[i],end = " - ")
```

Q24 La complexité en temps nécessite d'étudier "prochaineVille". À cause de la première boucle, elle est quadratique en n car on fait un for avec dedans une recherche (qui est elle même en n). Dans trajet, on fait n appels de prochaineVille et on a donc un $\mathcal{O}(n^3)$ (tout le reste étant des opérations élémentaires). On pourrait facilement améliorer cela en $\mathcal{O}(n^2)$ en gardant dans une liste de taille n l'information si la ville i a déjà été visité avec un booléen. La recherche de si une ville a déjà été visitée ou non serait alors linéaire et non plus quadratique.