Autotest n°4 – Correction

Exercice 1 : *Cet exercice porte sur la programmation en Python, la récursivité et la méthode "diviser pour régner".*

Une ligne polygonale est constituée d'une liste ordonnée de points, appelés sommets, joints par des segments. L'algorithme de Douglas-Peuker permet de simplifier une ligne polygonale en supprimant certains de ses sommets. L'effet de l'algorithme appliqué aux lignes polygonales du contour de la France métropolitaine est illustré ci-dessous.

Avant application de l'algorithme

Après application de l'algorithme

On implémentera cet algorithme dans la dernière question de l'exercice. Pour cela nous allons d'abord implémenter des fonctions auxiliaires.

On suppose dans la suite que les sommets sont des points du plan dont les coordonnées (x,y) dans un repère orthonormé fixé sont représentées par des tuples de longueur 2.

1) La distance qui sépare deux points A et B de coordonnées (x_A,y_A) et (x_B,y_B) donnée par la formule $\sqrt{(x_B-x_A)^2+(y_B-y_A)^2}$.
   On rappelle que la fonction `sqrt` du module `math` de Python renvoie la racine carrée d'un nombre positif ou nul.
   a) Écrire une instruction qui permet d'importer la fonction `sqrt` du module `math`.
   Solution : `from math import sqrt`
   b) Supposant l'import réalisé, écrire une fonction `distance_points(a,b)` qui prend en argument deux tuples `a` et `b` représentant les coordonnées de deux points et renvoie la distance qui les sépare.

```
def distance_points(a, b):
    return sqrt((b[0]-a[0])**2+(b[1]-a[1])**2)
# Ou alors
def distance_points(a, b):
    xA, yA = a
    xB, yB = b
    return sqrt((xB-xA)**2+(yB-yA)**2)
```

2) On dispose d'une fonction `distance_point_droite(p, a, b)` qui prend en argument les tuples représentant les coordonnées respectives des points P, A et B, et qui renvoie la distance du point P à la droite (AB). L'exécution de cette fonction produit une erreur dans le cas où les points A et B sont égaux.

À l'aide des fonctions `distance_points` et `distance_point_droite`, écrire une fonction `distance(p, a, b)` qui renvoie la distance entre le point P et la droite (AB) si les points A et B sont distincts et la distance AP sinon.

```
def distance(p, a, b):
    if a == b:
        return distance_points(p, a)
    else:
        return distance_point_droite(p, a, b)
```

Dans la suite, on dira que la fonction `distance` calcule la distance entre le point P et les points A et B, éventuellement confondus.

3) On a besoin d'une fonction `le_plus_loin(ligne)` qui prend en argument une liste de tuples représentant les coordonnées des points d'une ligne polygonale.
Cette fonction doit renvoyer un tuple composé de :
- l'indice du point de coordonnées `p` de la ligne polygonale d'extrémités `deb` et `fin`, pour lequel la distance `distance(p, deb, fin)` est la plus grande ;
- la valeur correspondante de cette distance.

On fournit le code incomplet suivant :

```
def le_plus_loin(ligne):
    n = len(ligne)
    deb = ligne[0]
    fin = ligne[n-1]
    dmax = 0
    indice_max = 0
    for idx in range(1, n-1):
        p = ligne[idx]
        d = distance(p, deb, fin)
        if d > dmax:
            dmax = d
            indice_max = idx
    return (indice_max, dmax)
```

Recopier et compléter le code de cette fonction.

4) Écrire une fonction `extrait(tab, i, j)` qui renvoie la copie du tableau `tab` des cases d'indice `i` inclus à `j` inclus pour $0 \leq i \leq j <$ `len(tab)`.
L'appel `extrait([7, 4, 9, 12], 2, 3)` renverra ainsi `[9, 12]`.

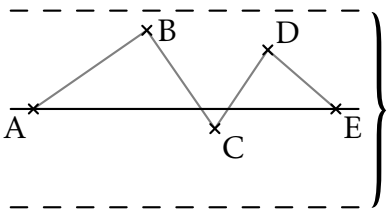
```
def extrait(tab, i, j):
    tab2 = []
    for k in range(i, j+1):
        tab2.append(tab[k])
    return tab2
# Ou alors :
def extrait(tab, i, j):
    return [tab[k] for k in range(i, j+1)]
```

L'algorithme de Douglas-Peuker repose sur une stratégie de type "diviser pour régner". Pour éliminer des sommets "proches de l'alignement", un seuil est fixé.

Étant donnée une ligne polygonale, le principe de l'algorithme est le suivant :
- si la ligne ne contient qu'un ou deux sommets, l'algorithme se termine ;
- sinon, on considère la droite formée par les extrémités de la ligne (son premier et dernier sommet), et on sélectionne le point le plus éloigné de cette droite (dans le cas où les

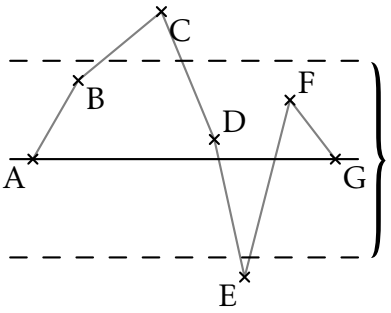
extrémités sont confondues, on sélectionne le point le plus éloigné de celles-ci) :

- si la distance entre le point sélectionné et la droite (ou les extrémités lorsqu'elles sont confondues) est inférieure au seuil fixé, on ne conserve que les extrémités de la ligne polygonale ;



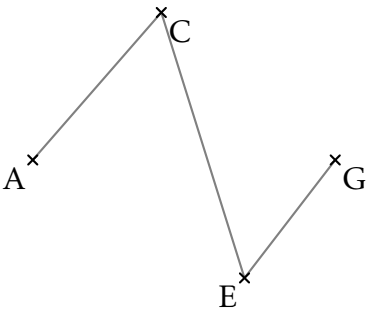
Distances inférieures au seuil
on ne conserve que les points A et E

- sinon, on applique l'algorithme de manière récursive aux deux parties de la ligne polygonale formées de la séquence formée du premier sommet jusqu'au sommet sélectionné d'une part et de la séquence formée du sommet sélectionné jusqu'au dernier sommet d'autre part. L'algorithme renvoie alors la concaténation des séquences simplifiées ainsi obtenues.



seuil

L'algorithme appelé sur la ligne polygonale [A,B,C,D,E,F,G] ci-dessus, va récursivement être appelé sur les lignes polygonales [A,B,C] et [C,D,E,F,G]. La ligne polygonale que l'on obtiendra à la fin de l'algorithme sera [A,C,E,G].



5) L'algorithme de Douglas-Peuker est implémenté par la fonction `simplifie` ci-dessous, qui prend en argument la ligne polygonale et le seuil choisi.

```
def simplifie(ligne, seuil):
    n = len(ligne)
    if n <= 2:
        return ligne
    else:
        indice_max, dmax = le_plus_loin(ligne)
        if dmax <= seuil:
            return [ligne[0], ligne[n-1]]
        else:
            ligne1 = simplifie(extrait(tab, 0, indice_max), seuil)
            ligne2 = simplifie(extrait(tab, indice_max, n-1), seuil)
            return extrait(ligne1, 0, len(ligne1)-2) + ligne2
```

Recopier et compléter le code de cette fonction.

Exercice 2 : *Cet exercice traite de programmation orientée objet en Python et d'algorithmique.*
Un pays est composé de différentes régions. Deux régions sont voisines si elles ont au moins une frontière en commun. L'objectif est d'attribuer une couleur à chaque région sur la carte du pays sans que deux régions voisines aient la même couleur et en utilisant le moins de couleurs possibles.
La figure 1 ci-dessous donne un exemple de résultat de coloration des régions de la France

métropolitaine.

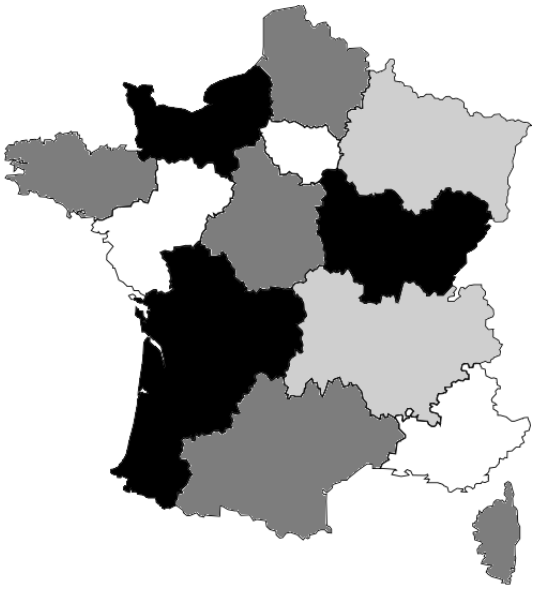


Figure 1 – Carte coloriée des régions de France métropolitaine

On rappelle quelques fonctions et méthodes des tableaux (le type `list` en Python) qui pourront être utilisées dans cet exercice :

- `len(tab)` : renvoie le nombre d'éléments du tableau `tab` ;
- `tab.append(elt)` : ajoute l'élément `elt` en fin de tableau `tab` ;
- `tab.remove(elt)` : enlève la première occurrence de `elt` de `tab` si `elt` est dans `tab`. Provoque une erreur sinon.

Exemples :

- `len([1, 3, 12, 24, 3])` renvoie `5` ;
- avec `tab=[1, 3, 12, 24, 3]`, l'instruction `tab.append(7)` modifie `tab` en `[1, 3, 12, 24, 3, 7]` ;
- avec `tab=[1, 3, 12, 24, 3]`, l'instruction `tab.remove(3)` modifie `tab` en `[1, 12, 24, 3]`.

Les deux parties de cet exercice forment un ensemble. Cependant, il n'est pas nécessaire d'avoir répondu à une question pour aborder la suivante. En particulier, on pourra utiliser les méthodes des questions précédentes même quand elles n'ont pas été codées.
Pour chaque question, toute trace de réflexion sera prise en compte.

Partie 1

On considère la classe `Region` qui modélise une région sur une carte et dont le début de l'implémentation est :

```
1  class Region:
2      '''Modélise une région d'un pays sur une carte.'''
3      def __init__(self, nom_region):
4          '''
5          initialise une région
6          : param nom_region (str) le nom de la région
7          '''
8          self.nom = nom_region
9          # tableau des régions voisines, vide au départ
10         self.tab_voisines = []
11         # tableau des couleurs disponibles pour colorier la région
12         self.tab_couleurs_disponibles = ['rouge', 'vert', 'bleu',
13                                          'jaune', 'orange', 'marron']
```

```
14          # couleur attribuée à la région et non encore choisie au départ
15          self.couleur_attribuee = None
```

1) Associer, en vous appuyant sur l'extrait de code précédent, les noms `nom`, `tab_voisines`, `tab_couleurs_disponibles` et `couleur_attribuee` au terme qui leur correspond parmi : *objet*, *attribut*, *méthode* ou *classe*. Ce sont des attributs.
2) Indiquer le type du paramètre `nom_region` de la méthode `__init__` de la classe `Region`. C'est un texte de type `str`.
3) Donner une instruction permettant de créer une instance nommée `ge` de la classe `Region` correspondant à la région dont le nom est "Grand Est".
   `ge = Region("Grand Est")`
4) Recopier et compléter la ligne 7 de la méthode de la classe `Region` ci-dessous :

```
1       def renvoie_premiere_couleur_disponible(self):
2           '''
3           Renvoie la première couleur du tableau des couleurs
4           disponibles supposé non vide.
5           : return (str)
6           '''
7           return ...
```

```
    def renvoie_premiere_couleur_disponible(self):
        return self.tab_couleurs_disponibles[0]
```

5) Recopier et compléter la ligne 6 de la méthode de la classe `Region` ci-dessous :

```
1       def renvoie_nb_voisines(self) :
2           '''
3           Renvoie le nombre de régions voisines.
4           : return (int)
5           '''
6           return ...
```

```
    def renvoie_nb_voisines(self):
        return len(self.tab_voisines)
```

6) Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 7 :

```
1       def est_coloriee(self):
2           '''
3           Renvoie True si une couleur a été attribuée à cette
4           région et False sinon.
5           : return (bool)
6           '''
7           ...
```

```
    def est_coloriee(self):
        return self.couleur_attribuee is not None
```

7) Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 9 :

```
1       def retire_couleur(self, couleur):
2           '''
3           Retire couleur du tableau de couleurs disponibles de
```

```
4         la région si elle est dans ce tableau. Ne fait rien sinon.
5         : param couleur (str)
6         : ne renvoie rien
7         : effet de bord sur le tableau des couleurs disponibles
8         """
9         ...
```

```
    def retire_couleur(self, couleur):
        if couleur in self.tab_couleurs_disponibles:
            self.tab_couleurs_disponibles.remove(couleur)
```

8) Compléter la méthode de la classe `Region` ci-dessous, à partir de la ligne 8, en utilisant une boucle :

```
1     def est_voisine(self, region):
2         """
3         Renvoie True si la region passée en paramètre est une
4         voisine et False sinon.
5         : param region (Region)
6         : return (bool)
7         """
8         ...
```

```
    def est_voisine(self, region):
        for v in self.tab_voisines:
            if v == region:
                return True
        return False
```

Partie 2

Dans cette partie :

- on considère qu'on dispose d'un ensemble d'instances de la classe `Region` pour lesquelles l'attribut `tab_voisines` a été renseigné ;
- on pourra utiliser les méthodes de la classe `Region` évoquées dans les questions de la partie 1 :
  - `renvoie_premiere_couleur_disponible`
  - `renvoie_nb_voisines`
  - `est_coloriee`
  - `retire_coule`

On a créé une classe `Pays` :

- cette classe modélise la carte d'un pays composé de régions ;
- l'unique attribut `tab_regions` de cette classe est un tableau (type **`list`** en Python) dont les éléments sont des instances de la classe `Region`.

9) Recopier et compléter la méthode de la classe `Pays` ci-dessous à partir de la ligne 7 :

```
1     def renvoie_tab_regions_non_coloriees(self):
2         """
3         Renvoie un tableau dont les éléments sont les régions
4         du pays sans couleur attribuée.
5         : return (list) tableau d'instances de la classe Region
6         """
7         ...
```

```
    def renvoie_tab_regions_non_coloriees(self):
        resultats = []
        for region in self.tab_regions:
            if not region.est_coloriee:
                resultats.append(region)
        return resultats
```

10) On considère la méthode de la classe Pays ci-dessous.

```
1     def renvoie_max(self):
2         nb_voisines_max = -1
3         region_max = None
4         for reg in self.renvoie_tab_regions_non_coloriees():
5             if reg.renvoie_nb_voisines() > nb_voisines_max:
6                 nb_voisines_max = reg.renvoie_nb_voisines()
7                 region_max = reg
8         return region_max
```

a) Expliquer dans quel cas cette méthode renvoie **None**.
La méthode renvoie **None** lorsque toutes les régions ont été coloriées.

b) Indiquer, dans le cas où cette méthode ne renvoie pas **None**, les deux particularités de la région renvoyée.
La région renvoyée n'est pas coloriée et a le nombre de voisines maximal.

11) Coder la méthode `colorie(self)` de la classe Pays qui choisit une couleur pour chaque région du pays de la façon suivante :

- On récupère la région non coloriée qui possède le plus de voisines.
- Tant que cette région existe :
  - La couleur attribuée à cette région est la première couleur disponible dans son tableau de couleurs disponibles.
  - Pour chaque région voisine de la région :
    - si la couleur choisie est présente dans le tableau des couleurs disponibles de la région voisine alors on la retire de ce tableau.
  - On récupère à nouveau la région non coloriée qui possède le plus de voisines.

```
    def colorie(self):
        region = self.renvoie_max()
        while region is not None:
            couleur = region.renvoie_premiere_couleur_disponible()
            region.couleur_attribuee = couleur
            for voisine in region.tab_voisines:
                voisine.retire_couleur(couleur)
            region = self.renvoie_max()
```

Exercice 3 : *Cet exercice traite du thème base de données, et principalement du modèle relationnel et du langage SQL.*

L'énoncé de cet exercice peut utiliser les mots du langage SQL suivants :

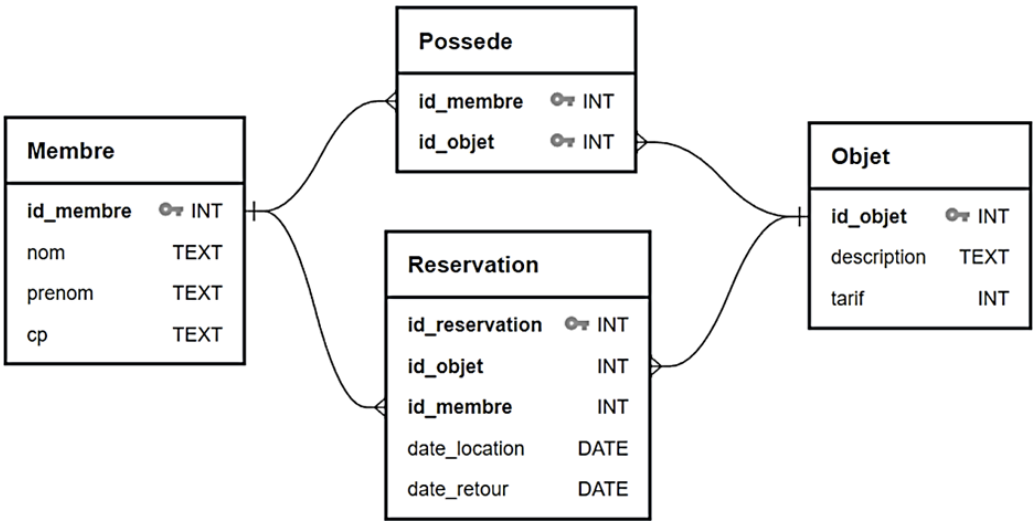
SELECT, **FROM**, **WHERE**, **JOIN ON**, **INSERT INTO**, **VALUES**, **UPDATE**, **SET**, **DELETE**, **COUNT**, **AND**, **OR**

Un site permet à ses membres de proposer à la location du matériel et de louer du matériel. Ceci permet de mutualiser du matériel entre membres et au propriétaire de rentabiliser cet achat. Le temps d'utilisation du matériel s'en trouve ainsi augmenté et le nombre d'appareils diminué.

Le modèle relationnel est donné par le schéma ci-dessous.

La table `Membre` contient les informations de chaque utilisateur du site (nom, prénom et code postal). La table `Objet` décrit le type d'objet à la location ainsi que son tarif de location journalier.

La table `Reservation` répertorie toutes les réservations effectuées par les membres du site avec notamment leur date de début et de fin de location. La table `Possede` permet de lier les tables `Membre` et `Objet`.



Conventions utilisées pour le schéma :

- Les clés primaires et étrangères sont mises en gras ;
- un symbole identifie une clé primaire ;
- un symbole entre deux attributs indique qu'ils doivent partager les mêmes valeurs et qu'ils sont reliés de la manière suivante : le côté indique la clé primaire et le côté indique la clé étrangère.

On donne à la fin de cet exercice les tables à un instant donné.

1) Dans cette partie, on ne demande pas de requête SQL. En étudiant le contenu des tables :
   a) Indiquer quels sont les prénoms et noms du ou des membres du site qui proposent la location d'un appareil à raclette ;
   Ali Mohamed et Alonso Fernando ont tous les deux un appareil à raclette.
   b) Donner le prénom et le nom du membre qui ne propose pas d'objet à la location.
   Ferrand Pauline et Kane Harry ne proposent aucun objet à la location.
2) a) Donner le résultat de la requête suivante :

```
SELECT nom, prenom FROM Membre WHERE cp = "69003";
```

   On obtient Dupont Antoine et Kane Antoine dont le code postal est 69003.
   b) Écrire une requête permettant de connaître le tarif de location d'une scie circulaire.
   `SELECT tarif FROM Objet WHERE description = "Scie circulaire";`

c) Écrire une requête permettant de modifier le tarif de location d'un nettoyeur à haute pression pour le passer à 15 € par jour au lieu de 20 € par jour.

```
UPDATE Objet SET tarif=15 WHERE description = "Nettoyeur haute pression";
```

d) Écrire une requête SQL permettant d'ajouter Wendie Renard habitant à Villeurbanne (code postal 69100) dans la table `Membre`, avec un `id_membre` de 6.

```
INSERT INTO Membre VALUES (6, "Renard", "Wendie", "69100");
```

3) a) Expliquer la limitation importante d'utilisation du service offert par le site si l'on utilisait le couple de clés étrangères (`id_objet`, `id_membre`) en tant que clé primaire de la relation `Reservation`.

Pour ne pas avoir de doublons, un même membre ne peut pas proposer deux objets du même type à la location.

b) Mohamed Ali décide de ne plus être membre du site. Il faut donc le supprimer de la table `Membre` à l'aide de la requête :

```
DELETE FROM Membre
WHERE nom = "Ali" AND prenom ="Mohamed";
```

Expliquer pourquoi cette requête produit une erreur.

Puisqu'il y a l'`id_membre` de Mohamed Ali qui est utilisé dans les tables `Possede` et `Objet`, la suppression n'est pas possible afin de maintenir l'intégrité de la base.

c) Proposer une suite de requêtes utilisant le mot clé **DELETE**, précédant la requête ci-dessus pour supprimer correctement Mohamed Ali, dont l'`id_membre` est 1, de la base de données. Rappel : la relation `Objet` décrit le type d'objet à la location. Il n'y a pas d'objet à supprimer dans cette table lors du départ d'un membre.

Il faut supprimer les enregistrements liés dans `Possede` et `Reservation` :

```
DELETE FROM Possede WHERE id_membre = 1;
DELETE FROM Rervation WHERE id_membre = 1;
DELETE FROM Membre WHERE nom = "Ali" AND prenom ="Mohamed";
```

4) Dans cette partie, les requêtes utilisent des jointures entre tables. On supposera donc les numéros `id_membre` et `id_objet` non connus.

a) Écrire une requête permettant de compter le nombre de réservations réalisées par Fernando Alonso.

```
SELECT COUNT(*) FROM Reservation
JOIN Membre ON Membre.id_membre = Reservation.id_membre
WHERE nom = "Alonso" AND prenom ="Fernando";
```

b) Écrire une requête permettant de connaître les noms et prénoms des membres possédant un appareil à raclette.

```
SELECT nom, prenom FROM Membre
JOIN Possede ON Membre.id_membre = Possede.id_membre
JOIN Objet ON Possede.id_objet = Objet.id_objet
WHERE description = "Appareil à raclette";
```

Membre

id_membre	nom	prenom	cp
1	"Ali"	"Mohamed"	"69110"
2	"Alonso"	"Fernando"	"69005"
3	"Dupont"	"Antoine"	"69003"
4	"Ferrand"	"Pauline"	"69160"
5	"Kane"	"Harry"	"69003"

Possede

id_membre	id_objet
1	4
1	6
2	4
3	3
3	5
4	1
4	2

Objet

id_objet	description	tarif
1	"Nettoyeur haute pression"	20
2	"Taille-haie"	15
3	"Perforatrice"	15
4	"Appareil à raclette"	10
5	"Scie circulaire"	15
6	"Appareil à gaufre"	10

Reservation

id_reservation	id_objet	id_membre	date_location	date_retour
1	4	5	2022-02-18	2022-02-19
2	1	2	2022-05-05	2022-05-06
3	3	1	2022-07-10	2022-07-12
4	3	1	2022-08-12	2022-08-14
5	2	2	2022-10-20	2022-10-22
6	2	2	2022-10-20	2022-10-22