Exercice 2 (6 points)

Cet exercice porte sur la programmation orientée objet, la récursivité et les algorithmes gloutons.

Une entreprise souhaite gérer les colis qu'elle expédie à l'aide d'une application informatique. On sait que chaque colis a un identifiant unique, un poids, une adresse de livraison et un état. Pour chacun d'entre eux, trois états sont possibles : "préparé", "transit" ou "livré".
Pour cela, on a créé une classe `Colis` avec les attributs suivants :

- `id` : un identifiant unique (de type `str`) ;
- `poids` : le poids du colis en kilogrammes (de type `float`) ;
- `adresse` : l'adresse de destination (de type `str`) ;
- `etat` : l'état du colis (de type `str` parmi `'préparé'`, `'transit'`, `'livré'`).

Lorsque l'on crée une instance de la classe `Colis`, l'attribut `etat` est initialisé à `'préparé'` tandis que les valeurs des autres attributs sont passées en paramètres. Voici le début du code Python de la classe `Colis` :

```
1 class Colis:
2     def __init__(self, id, poids, adresse):
3           self.id = id
4           self.poids = poids
5           self.adresse = adresse
6           self.etat = 'préparé'
```

On crée, par exemple, les deux colis suivants :

```
colisA = Colis('AC12', 5.0, '20 rue de la paix 57000 Metz')
colisB = Colis('AF34', 10.25, '32 rue du centre 57000 Metz')
```

1. Écrire la méthode `passer_transit` de la classe `Colis` qui permet de mettre l'état du colis à la valeur `'transit'`.

```
1 class Colis:
2     def __init__(self, id, poids, adresse):
3           self.id = id
4           self.poids = poids
5           self.adresse = adresse
6           self.etat = 'préparé'

7     def passer_transit(self):
8           self.etat = 'transit'
```

Par exemple, après l'exécution des trois instructions suivantes, on a ajouté les deux colis crées précédemment à la liste `liste_colis` :

```
1 liste_colis = []
2 ajouter_colis(liste_colis, colisA)
3 ajouter_colis(liste_colis, colisB)
```

2. Dans cette question uniquement, on considère que l'acheminement des colis de plus de 25 kg est refusé par le transporteur.

Recopier et modifier le code de la fonction `ajouter_colis` afin qu'elle ajoute le colis à la liste si son poids est inférieur ou égal à 25 kg et qu'elle affiche le message "Dépassement du poids maximal autorisé" sinon.

```
def ajouter_colis(liste, colis):
    if colis.poids <= 25:
        liste.append(colis)
    else:
        print("Dépassement du poids maximal autorisé")
```

3. Écrire une fonction `nb_colis` qui prend en paramètre une liste d'objets de la classe `Colis` et qui renvoie le nombre de colis présents dans cette liste.

```
def nb_colis(liste):
    return len(liste)
```

4. Recopier et compléter les lignes 2 et 4 du code ci-après de la fonction `poids_total` qui prend en paramètre une liste d'objets de la classe `Colis` et qui renvoie le poids total de l'ensemble des colis de cette liste.

```
1 def poids_total(liste):
2     total = 0
3     for c in liste :
4         total = total + c.poids
5     return total
```

5. Écrire une fonction `liste_colis_etat` qui prend en paramètres une liste d'objets de la classe `Colis` et une chaîne de caractères `statut` (parmi `'préparé'`, `'transit'` ou `'livré'`) et qui renvoie une nouvelle liste contenant l'ensemble des colis de cette liste dont l'état est le même que `statut`.

```
def liste_colis_etat(liste, statut):
    # Initialiser une liste vide pour stocker les colis correspondant au statut
    colis_correspondants = []
    # Parcourir chaque colis dans la liste
    for colis in liste:
        # Vérifier si l'état du colis correspond au statut recherché
        if colis.etat == statut:
            # Ajouter le colis à la liste des colis correspondants
            colis_correspondants.append(colis)

    # Retourner la liste des colis correspondants
    return colis_correspondants
```

ou

```
def liste_colis_etat(liste, statut):
    return [colis for colis in liste if colis.etat == statut]
```

6. Donner le nom du tri utilisé dans la fonction `tri_decroissant` ainsi que son coût dans le pire des cas.

```
1 def tri_decroissant(liste):
2  n = len(liste)
3  for i in range(n - 1):
4        min_pos = i
5        for j in range(i + 1, n):
6              if liste[j].poids > liste[min_pos].poids:
7                    min_pos = j
8        # Échanger les éléments
9        temp = liste[i]
10             liste[i] = liste[min_pos]
11             liste[min_pos] = temp
12       return liste
```

Le tri utilisé dans la fonction **tri_decroissant** est le tri par sélection. Son coût dans le pire des cas est quadratique, soit en $O(n^2)$.

7. Citer un autre algorithme de tri qui aurait pu être utilisé, ainsi que son coût dans le pire des cas.

Le tri par insertion en $O(n^2)$ également….

8. Recopier et compléter le code ci-dessus de la fonction `chargement_glouton`.

```
1 def chargement_glouton(liste, rang, capacite):
2     if rang == len(liste):
3           return []
4     elif liste[rang].poids <= capacite:
5           return [liste[rang]] + chargement_glouton(liste, rang + 1, capacite -
                                                              liste[rang].poids)
6     else:
7           return chargement_glouton(liste, rang + 1, capacite )
```

9. Expliquer brièvement pourquoi, lors d'un appel à la fonction `chargement_glouton`, on peut obtenir l'erreur suivante.

   `RecursionError: maximum recursion depth exceeded while calling a Python object.`

L'erreur `RecursionError: maximum recursion depth exceeded while calling a Python object` se produit lorsque la profondeur maximale de récursion est atteinte. Cela peut se produire si la liste de colis est très grande ou si la capacité du camion est très petite, ce qui entraîne un grand nombre d'appels récursifs.

10. Écrire une fonction `chargement_glouton2` **itérative** (sans récursivité) qui prend en paramètres `liste` une liste de colis triés par poids décroissants et `capacite` la capacité du camion exprimée en kilogrammes, et qui renvoie la liste des colis à charger pour maximiser le poids total sans dépasser la capacité.

On pourra créer une liste `colis_a_charger`, puis parcourir les colis triés en les ajoutant à cette liste tant que le poids total n'excède pas la capacité du camion.

```
def chargement_glouton2(liste, capacite):
    colis_a_charger = []
    poids_total = 0
    for colis in liste:
        if poids_total + colis.poids <= capacite:
            colis_a_charger.append(colis)
            poids_total += colis.poids
    return colis_a_charger
```