# DS 1 Terminale Correction

# 2025 sujet 01 — Correction de l'exercice 2

Correction question par question.

## Question 1

La méthode `passer_transit` met simplement l'attribut `etat` à la valeur `'transit'`.

```
class Colis:
    def __init__(self, id, poids, adresse):
        self.id = id
        self.poids = poids
        self.adresse = adresse
        self.etat = "préparé"

    def passer_transit(self):
        self.etat = "transit"
```

## Question 2

On modifie la fonction `ajouter_colis` pour refuser les colis de plus de 25 kg.

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

## Question 3

Le nombre de colis est la longueur de la liste.

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

## Question 4

On additionne les poids des colis de la liste.

```
def poids_total(liste):
    total = 0
    for c in liste:
        total = total + c.poids
    return total
```

## Question 5

On construit une nouvelle liste contenant seulement les colis dont l'état est égal au statut demandé.

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

## Question 6

La fonction `tri_decroissant` utilise un **tri par sélection**.

À chaque étape, on cherche le colis de poids maximal dans la partie non triée de la liste, puis on l'échange avec l'élément courant.

Dans le pire des cas, le coût en comparaisons est :

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$$

Le coût est donc **quadratique**, c'est-à-dire en $O(n^2)$.

# Question 7

On aurait pu utiliser le **tri fusion**.

Son coût dans le pire des cas est en :

$O(n\log_2(n))$

# Question 8

On complète la fonction récursive.

Cas d'arrêt : si `rang == len(liste)`, il n'y a plus de colis à étudier.

Si le colis courant rentre dans la capacité restante, on le prend et on continue avec une capacité diminuée.

Sinon, on passe au colis suivant.

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

# Question 9

L'erreur `RecursionError: maximum recursion depth exceeded` peut se produire si le nombre d'appels récursifs devient trop grand.

Ici, cela peut arriver avec une très longue liste de colis, car la fonction fait un nouvel appel récursif pour chaque rang. Python limite la profondeur maximale de récursion.

# Question 10

Version itérative de l'algorithme glouton.

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

# Script complet de l'exercice 2

```
class Colis:
    def __init__(self, id, poids, adresse):
        self.id = id
        self.poids = poids
        self.adresse = adresse
        self.etat = "préparé"

    def passer_transit(self):
        self.etat = "transit"

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

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

def poids_total(liste):
    total = 0
    for c in liste:
        total = total + c.poids
    return total

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

def tri_decroissant(liste):
    n = len(liste)
    for i in range(n - 1):
        min_pos = i
        for j in range(i + 1, n):
            if liste[j].poids > liste[min_pos].poids:
                min_pos = j
        temp = liste[i]
        liste[i] = liste[min_pos]
        liste[min_pos] = temp
    return liste

def chargement_glouton(liste, rang, capacite):
    if rang == len(liste):
        return []
    elif liste[rang].poids <= capacite:
        return [liste[rang]] + chargement_glouton(liste, rang + 1, capacite - liste[rang].poids)
    else:
        return chargement_glouton(liste, rang + 1, capacite)

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

colisA = Colis("AC12", 5.0, "20 rue de la paix 57000 Metz")
colisB = Colis("AF34", 10.25, "32 rue du centre 57000 Metz")
liste_colis = []
ajouter_colis(liste_colis, colisA)
ajouter_colis(liste_colis, colisB)
print(nb_colis(liste_colis))
print(poids_total(liste_colis))
```