DS - Développement Python (DEVE514)

Gaël Roustan (Argonaultes)

2024 - 2025

Abstract

Ce document contient les instructions du devoir surveillé relatif au cours sur le développement en Python (DEVE514).

Contents

Introduction

Durée de l’évaluation

Vous avez 1h00 pour répondre au sujet.

Modalités de rendu

Le rendu se fait via un repo GIT accessible en public. Vos fichiers doivent être à la racine de ce repo GIT.

Les corrections sont automatisées, il est donc important de bien respecter les noms des fichiers demandés ainsi que les nom des fonctions et des classes. Dans le cas contraire, ce sera automatiquement 0 puisque la correction ne pourra pas être réalisée.

L’adresse de votre repo GIT doit être fournie via le formulaire en ligne dont l’adresse vous est donnée en séance.

Chaque rendu est individuel.

Pour réaliser le travail, vous n’avez pas de restriction sur les ressources à votre disposition.

Barème

Pour chaque exercice, le nombre de points est indiqué.

Pour certaines questions, il est possible d’obtenir des points bonus en réalisant un travail complémentaire ou en répondant à quelques questions en option sur le questionnaire.

Pour chaque exercice, il existe plusieurs tests automatisés qui vous permettront d’avoir une partie des points.

Echauffements (20 points + 2 points bonus)

Dans un unique fichier `functions.py`, implémenter toutes les fonctions demandées. Vous pouvez ajouter des appels à vos fonctions mais ces appels ne doivent pas être déclenchés si on importe vos fonctions dans un autre script.

Pour cela, utiliser le bloc de code suivant :

```
if __name__ == '__main__':
    pass
```

nom du fichier pour le rendu : `functions.py`

Fonction several_zeros (4 points)

Votre fonction doit suivre le modèle de déclaration suivant :

```
def several_zeros():
    pass
```

La fonction ne prend pas de paramètre et produit à chaque appel une liste de 10 zéros.

Fonction several_zeros_custom (3 points + 1 point bonus)

Votre fonction doit suivre le modèle de déclaration suivant :

```
def several_zeros_custom(nb_zeros):
    pass
```

La fonction prend un paramètre et produit à chaque appel une liste de `nb_zeros` zéros.

BONUS : Modifier la déclaration de la fonction pour qu'un appel à la fonction `several_zeros_custom` déclenche le même résultat que si on l'avait appelé avec la valeur `10`.

Fonction matrix (4 points)

Votre fonction doit suivre le modèle de déclaration suivant :

```
def matrix(rows, cols):
    pass
```

La fonction prend 2 paramètres, `rows` pour le nombre de lignes de la matrice, et `cols` pour le nombre de colonnes de la matrice. Le résultat est une matrice remplie de `0`. Utiliser uniquement la structure liste pour produire le résultat attendu.

Ci-dessous quelques exemples d'utilisation attendu :

```
result = matrix(2, 3)
print(result[2][3]) # error
print(result[1][2]) # 0
print(result[0]) # [0, 0, 0]
```

Objet matrix (9 points)

Plutôt qu'une simple fonction, utiliser la POO pour remplir la même fonctionnalité. Le travail est guidé à travers les 3 sous-parties suivantes :

1. Déclaration
2. Utilisation des valeurs
3. Egalité de 2 matrices

Tout comme pour les fonctions, merci de bien respecter les noms des classes, des attributs et des méthodes proposés.

Déclaration (3 points)

Créer une classe `Matrix` qui prend dans le constructeur 2 paramètres : `rows` et `cols`. Une matrice de taille `rows` x `cols` est alors créée et initialisée à `0`.

Cette classe `Matrix` a un attribut qui représente la matrice. Attention à bien nommer l'attribut pour ne pas exposer facilement cet attribut pour des modifications.

Utilisation des valeurs (3 points)

Créer une méthode d'instance `get_value(self, row, col)` qui permet de récupérer la valeur à la position `row`, `col`.

Egalité de 2 matrices (3 points)

Le fait d'encapsuler nos matrices dans une classe nous a fait perdre le test d'égalité. Etant donné que la création d'une matrice produit toujours une matrice à zéros, il faudrait que le test d'égalité refonctionne. Pour cela, aidez vous de la méthode `__eq__(self, other)`. Vous trouverez ci-dessous un exemple de code à faire fonctionner :

```
m1 = Matrix(2, 3)
m2 = Matrix(2, 3)
print(m1 == m2) # True
```

Supervision (5 points)

nom du fichier pour le rendu : `disk.py`

Modélisation

Vous travaillez sur un outil de supervision. Vous devez faciliter le travail de collecte et de stockage des informations relatives à votre environnement supervisé.

Nous nous occupons ici de l'espace stockage et plus particulièrement des disques.

Créer une classe `Disk` qui représente un disque de stockage et qui a les propriétés suivantes lors de sa création :

- espace total : `total`
- espace occupé : `used`

Des nouveaux accesseurs

Créer un accesseur pour obtenir l'espace libre (`free`). Attention, cette valeur ne doit jamais être stockée.

Affichage

Modifier la classe `Disk` pour permettre l'affichage d'une instance qui renseigne les informations suivantes du disque

```
disk = Disk(10, 2)
print(disk) # Disk[total: 10 Gb, used: 2 Gb]
```

Tri personnalisé

Pour terminer, nous souhaitons trier une liste de disques en fonction de l'espace disque utilisé. A partir d'une liste a priori non trié, nous voulons obtenir une liste de disques rangée par ordre croissant de ratio d'espace disque utilisé.

Pour réaliser ce tri, nous utiliserons la fonction `sorted`. Il faut donc modifier la classe `Disk` pour permettre ce tri.

Regardez du côté de la documentation technique de `datamodel`

Exemple d'utilisation

Exemple de code opérationnel avec la classe à créer

```
disk1 = Disk(total = 10, used = 2)
disk2 = Disk(total = 20, used = 5)
print(disk1.free) # 8
print(disk2.free) # 15
print(disk1.used_perc) # 0.2
print(disk2.used_perc) # 0.25
disks = [disk1, disk2]
disks_sorted = sorted(disks) # trié selon le pourcentage
 ↪  d'espace utilisé
```

Bonus 1 : Mini casse tête (3 points bonus)

nom du fichier pour le rendu : `ascii.py`

Un texte se cache derrière cette liste de nombres.

```
[84, 104, 105, 115, 32, 101, 120, 101, 114, 99, 105, 99,
 ↪  101, 32, 105, 115, 32, 97, 119, 101, 115, 111, 109,
 ↪  101, 32, 33]
```

Créer une fonction `decrypt`, respectant la déclaration ci-dessous, qui prend en paramètre une liste de nombres et renvoie une chaîne de caractères.

```
def decrypt(message : [int]) -> str:
    return ''
```

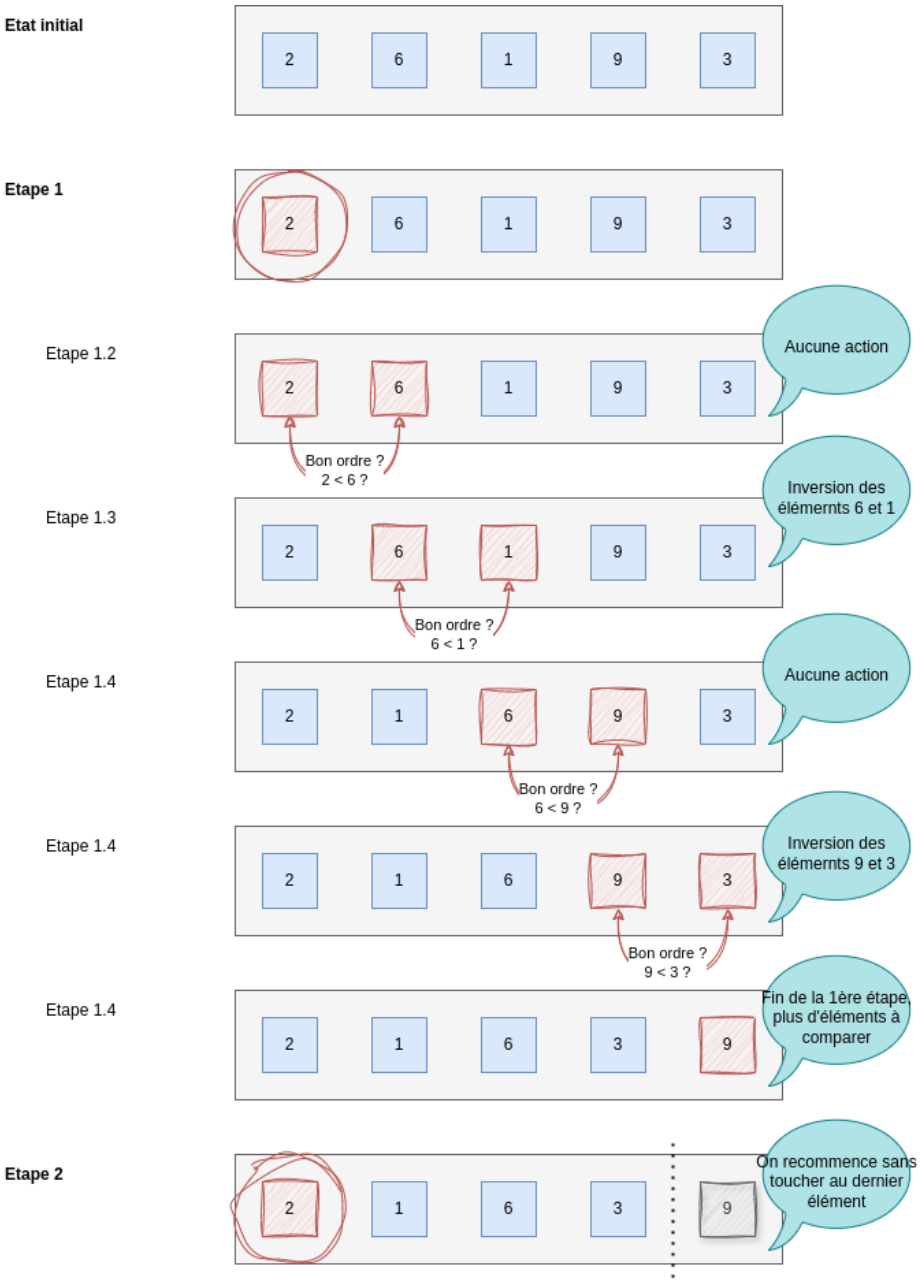
Bonus 2 : Fonction sort (3 points bonus)

Votre fonction doit être déclarée en suivant le prototype suivant.

```
def my_sort(my_list: [int]) -> [int]:
    pass
```

Vous ne devez donc pas trier la liste passée en paramètre mais renvoyer une nouvelle liste triée par ordre croissant.

Vous devez implémenter un algorithme de tri basique mais très connu. C'est un algorithme de tri qui consiste à parcourir une liste et à inverser si nécessaire l'élément considéré avec le prochain élément. Dans le visuel ci-après nous illustrons comment se déroule la 1ère étape de cet algorithme.



Attention, l'utilisation des méthodes `sort`, `sorted` ou toute autre fonction issue d'une bibliothèque tierce sera sanctionnée par un 0 à l'exercice.

Vous devez retrouver le nom de cet algorithme de tri.