Cette situation d'évaluation comporte ce document ainsi que des fichiers Python présents sur l'ordinateur à la disposition du candidat.
Le candidat doit restituer ce document avant de sortir de la salle d'examen.
Le candidat doit agir en autonomie et faire preuve d'initiative tout au long de l'épreuve.

En cas de difficulté, le candidat peut solliciter l'examinateur afin de lui permettre de continuer la tâche.
Des moments privilégiés pour solliciter l'examinateur sont indiqués dans le document sous la forme d'appels professeur.
L'examinateur peut intervenir à tout moment, s'il le juge utile.

Contexte

La société **RailPlan** souhaite concevoir un outil pour **vérifier la cohérence d'un emploi du temps de trains** sur une même voie ferrée.

Chaque train est représenté par un tuple composé des valeurs suivante :

- un **nom** une chaîne de caractères (ex. "TGV 8437"),
- une **heure de départ** est une chaîne de caractères toujours au format "hh:mm". A une heure deux du matin l'heure sera représentée "01h02",
- une **heure d'arrivée**, même format que l'heure d'arrivée "hh:mm".

On suppose que deux trains ne peuvent pas se trouver sur la même voie au même moment.

1. Conversion d'un horaire en minutes

Il peut être utile de convertir une représentation d'une durée au format "hh:mm" en nombre total de minutes pour faciliter les comparaisons temporelles.
Écrire une fonction `vers_minutes(horaire)` qui transforme un horaire "hh:mm" en nombre total de minutes depuis minuit.

Exemples :

- "00:00" → 0
- "01:30" → 90
- "23:59" → 1439

2. Vérification d'un chevauchement

Pour construire l'emploi du temps, il est nécessaire de vérifier si deux trains ont des horaires qui se chevauchent par exemple pour la gestion de quais.
Si deux trains doivent accéder à un quai, il faut de manière logique que les deux trains ne se trouvent pas au même quai pour ne pas en empêcher un autre de partir.

Écrire une fonction `chevauchement(train1, train2)` qui renvoie `True` si deux trains se chevauchent dans le temps, `False` sinon.

Rappel : chaque train est représenté par un tuple `(nom, heure_depart, heure_arrivee)`.

Appel 1 : Appeler le professeur pour présenter la fonction.

3. Tests unitaires de non-conflit

On doit tester la fonction `chevauchement` avec des cas particuliers pour s'assurer qu'elle fonctionne correctement.
Identifier les différents scénarios de test que l'on doit couvrir. Ces tests doivent être écrits sous forme d'assertions.
Compléter la fonction `test_chevauchement` en ajoutant **trois tests pertinents**.

4. Génération du planning quotidien

On souhaite produire un **planning texte** contenant les horaires des trains triés par ordre de départ.

Chaque ligne du fichier doit être de la forme :

```
Nom — Départ : hh:mm — Arrivée : hh:mm
```

Expliquer quel est l'intêret d'utiliser la fonction `sorted` de Python et pas utiliser la fonction `sort` qui trie une liste en place.
Appeler la fonction `exporter_planning` 2 fois d'affilée quel est le résultat dans le fichier de sortie ?
Remplacer le paramètre `'a'` par un paramètre `'w'` et appeler 2 fois la fonction. Quelle est la différence sur le résultat ? Quel paramètre est le plus approprié dans le cas ?

Appel 2 : Appeler le professeur pour présenter la sortie finale.

Description du dossier

Le dossier fourni au candidat contient :

- `trains.py` : code source à compléter ou à exécuter ;
- `tests_trains.py` : tests unitaires.

Préparation de l'environnement

Aucune bibliothèque externe n'est nécessaire. L'épreuve se déroule avec l'interpréteur Python standard.