Énoncé du problème

Vous devez écrire un programme nommé **analyser_clients** qui reçoit deux listes [list] contenant des dictionnaires [dict] dont les clés sont des numéros de billets [int]. Les valeurs de la première liste sont les noms de passagers d'un autobus [str], et la deuxième leur âge [int].

Voici un exemple de données d'entrée :

```
noms = [
    {1: 'Alice'},
    {2: 'Bob'}
]

ages = [
    {1: 10},
    {2: 70}
]
```

Étape 1 : Filtrer les listes en retirant les passagers non valides

Vous devez retirer les passagers qui ne sont pas valides des deux listes.

Un passager est valide si les conditions suivantes sont respectées :

- Son nom est une chaîne de caractères non vide.
- Son âge est situé entre 0 et 120 inclusivement.
- Le numéro de son billet se trouve dans les deux listes (c.-à-d. les noms et les âges).
- Le numéro de son billet est un entier positif incluant zéro.

Par exemple, considérez les listes suivantes:

```
noms = [
    {-1: 'Alice'},
    {2: 'Bob'},
    {3: 'Eva'},
    {4: ''},
    {9: 'Jean'}
]

ages = [
    {-1: 10},
    {2: 170},
```

```
    {3: 0},
    {4: 40},
    {5: 45},
    {9: 50}
]
```

Les passagers invalides sont :

- Le premier passager (Alice), car son numéro de billet (-1) est négatif.
- Le deuxième passager (Bob), car son âge est supérieur à 120.
- Le troisième passager (Eva), car son âge est égal à zéro.
- Le quatrième passager, car son nom est une chaîne de caractères vide.
- Le cinquième passager, car son numéro de billet n'est pas dans les deux listes et nous n'avons pas son nom.

Les passagers valides sont :

- Le sixième passager (Jean) dont le numéro de billet est 9, car il respecte toutes les conditions.

Les listes filtrées sont donc :

```
noms = [
    {9: 'Jean'}
]

ages = [
    {9: 50}
]
```

Étape 2: Valider que les listes filtrées ne sont pas vides

Si les listes filtrées sont vides, le programme doit déclencher une erreur de la forme suivante :

```
AssertionError: Les listes filtrées sont vides.
```

Cela met fin à l'exécution du programme.

Étape 3 : Trier les passagers valides par catégorie d'âge

Les passagers sont ensuite triés dans des listes [list] selon leur âge basé selon la grille suivante :

- **Bambin (0-2 ans) :** Gratuit
- **Enfant (3-5 ans) :** 0.50 $
- **Junior (6-12 ans) :** 1.00 $
- **Étudiant (13-25 ans) :** 2.00 $
- **Adulte (26-50 ans) :** 3.50 $
- **Aîné (51-65 ans) :** 2.50 $
- **Âge d'or (66 ans et plus) :** 1.50 $

Il y a donc autant de listes qu'il y a de catégories.

Étape 4 : Analyse

Le programme retourne une structure de donnée imbriquée de la forme suivante :

```
{
  'bambin' : {'noms': [noms bambins], 'profit': total pour les bambins},
  'enfant' : {'noms': [noms des d'enfants], 'profit': total pour les enfants},
  ...
  'age_or' : {'noms': [noms des âges d'or], 'profit': total pour l'âge d'or},
  'profit_total' : total des profits
}
```

Remarquez que la dernière clé (profit_total) correspond à la somme de tous les profits. Les totaux sont calculés à partir de la grille de l'étape précédente.

Par exemple, considérez les listes suivantes :

```
noms = [
    {1: 'Alice'},
    {2: 'Bob'}
]

ages = [
    {1: 10},
```

```
            {2: 70}
        ]
```

Le dictionnaire retourné est le suivant :

```
    {
        'bambin': {'noms': [], 'profit': 0.00},
        'enfant': {'noms': [], 'profit': 0.00},
        'junior': {'noms': ['Alice], 'profit': 1.00,
        'etudiant': {'noms': [], 'profit': 0.00},
        'adulte': {'noms': [], 'profit': 0.00},
        'aine': {'noms': [], 'profit': 0.00},
        'age_or': {'noms': ['Bob'], 'profit': 1.50},
        'profit_total': 2.50
    }
```

Notez que les noms des catégories doivent être en minuscules (bambin, enfant, etc.).