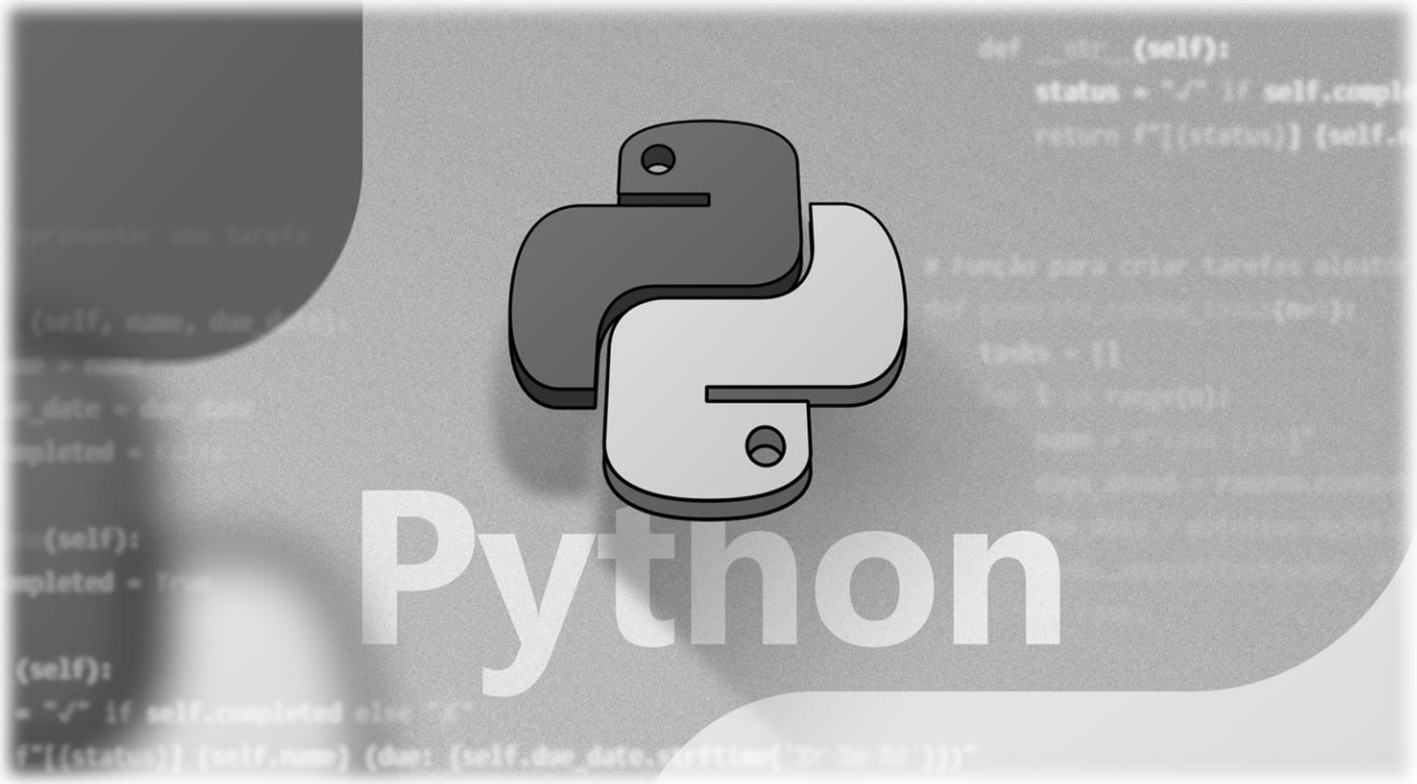
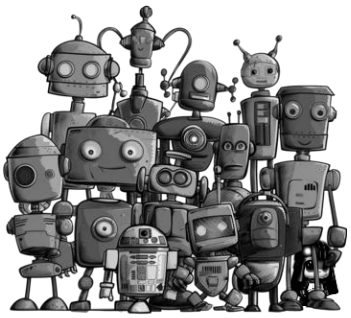
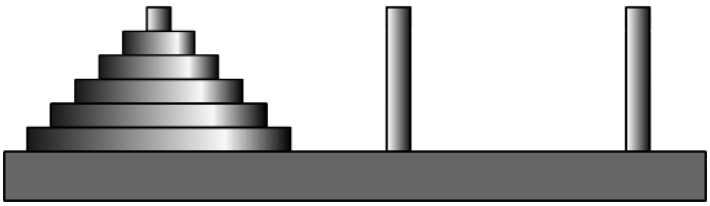
Cahier d'entrainement
les Piles - les Files
les Listes – les Tuples - les Dictionnaires





Où est DarkSATHI Li ?



Exercice 1 : Amérique du nord 2021 sujet 1

Cet exercice porte sur la notion de pile, de file et sur la programmation de base en Python.
Les interfaces des structures de données abstraites Pile et File sont proposées ci-dessous. On utilisera uniquement les fonctions ci-dessous :

Structure de données abstraite : Pile

Opérations :

- `creer_pile_vide : ∅ → Pile`
  `creer_pile_vide()` renvoie une pile vide
- `est_vide : Pile → Booléen`
  `est_vide(pile)` renvoie True si pile est vide, False sinon
- `empiler : Pile, Élément → ∅`
  `empiler(pile, element)` ajoute element à la pile pile
- `depiler : Pile → Élément`
  `depiler(pile)` renvoie l'élément au sommet de la pile en le retirant de la pile

Structure de données abstraite : File

Opérations :

- `creer_file_vide : ∅ → File`
  `creer_file_vide()` renvoie une file vide
- `est_vide : File → Booléen`
  `est_vide(file)` renvoie True si file est vide, False sinon
- `enfiler : File, Élément → ∅`
  `enfiler(file, element)` ajoute element dans la file file
- `defiler : File → Élément`
  `defiler(file)` renvoie l'élément au sommet de la file file en le retirant de la file file

1. On considère la file `F` suivante :

`enfilement → |"rouge"|"vert"|"jaune"|"rouge"|"jaune"|← défilement`

a. Quel sera le contenu de la pile `P` et de la file `F` après l'exécution du programme Python suivant ?

```
P = creer_pile_vide()
while not(est_vide(F)):
    empiler(P, defiler(F))
```

b. Créer une fonction `taille_file` qui prend en paramètre une file `F` et qui renvoie le nombre d'éléments qu'elle contient. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

2. Écrire une fonction `former_pile` qui prend en paramètre une file `F` et qui renvoie une pile `P` contenant les mêmes éléments que la file. Le premier élément sorti de la file devra se trouver au sommet de la pile; le deuxième élément sorti de la file devra se trouver juste en dessous du sommet, etc.

Exemple :
si `F = |"rouge"|"vert"|"jaune"|"rouge"|"jaune"|` alors l'appel `former_pile(F)` va renvoyer la pile `P` ci-dessous :

```
P = | "jaune" |
    | "rouge" |
    | "jaune" |
    | "vert"  |
    | "rouge" |
```

3. Écrire une fonction `nb_elements` qui prend en paramètres une file `F` et un élément `elt` et qui renvoie le nombre de fois où `elt` est présent dans la file `F`. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

4. Écrire une fonction `verifier_contenu` qui prend en paramètres une file `F` et trois entiers : `nb_rouge`, `nb_vert` et `nb_jaune`. Cette fonction renvoie le booléen `True` si "rouge" apparaît au plus `nb_rouge` fois dans la file `F`, "vert" apparaît au plus `nb_vert` fois dans la file `F` et "jaune" apparaît au plus `nb_jaune` fois dans la file `F`. Elle renvoie `False` sinon. On pourra utiliser les fonctions précédentes.

Exercice 2 : 2021 sujet 0

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe dernier arrivé, premier sorti.
Une pile est munie de quatre fonctions primitives définies dans le tableau ci-dessous :

Structure de données abstraite : File

Opérations :

- `creer_file_vide : ∅ → File`
  `creer_file_vide()` renvoie une file vide
- `est_vide : File → Booléen`
  `est_vide(file)` renvoie `True` si file est vide, `False` sinon
- `enfiler : File, Élément → ∅`
  `enfiler(file, element)` ajoute element dans la file file
- `defiler : File → Élément`
  `defiler(file)` renvoie l'élément au sommet de la file file en le retirant de la file file

1. On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut) :

```
| 4 |
| 2 |
| 5 |
| 8 |
```

Quel sera le contenu de la pile `Q` après exécution de la suite d'instructions suivante ?

```
Q = creer_pile_vide()
while not est_vide(P):
    empiler(Q, depiler(P))
```

2. On appelle hauteur d'une pile le nombre d'éléments qu'elle contient. La fonction `hauteur_pile` prend en paramètre une pile `P` et renvoie sa hauteur. Après appel de cette fonction, la pile `P` doit avoir retrouvé son état d'origine.

   Exemple : si `P` est la pile de la question 1 : `hauteur_pile(P) = 4`.

   Compléter sur votre copie le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les `...` par les bonnes instructions.

```
def hauteur_pile(P):
    Q = creer_pile_vide()
    n = 0
    while not(est_vide(P)):
        ...
        x = depiler(P)
        empiler(Q, x)
    while not(est_vide(Q)):
        ...
        empiler(P, x)
    return ...
```

3. Créer une fonction `max_pile` ayant pour paramètres une pile `P` et un entier `i`. Cette fonction renvoie la position `j` de l'élément maximum parmi les `i` derniers éléments empilés de la pile `P`. Après appel de cette fonction, la pile `P` devra avoir retrouvé son état d'origine. La position du sommet de la pile est `1`.

   Exemple : si `P` est la pile de la question 1 : `max_pile(P, 2) = 1`

4. Créer une fonction retourner ayant pour paramètres une pile P et un entier j. Cette fonction inverse l'ordre des j derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.
   Exemple : si `P` est la pile de la question 1, après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

```
| 5 |
| 2 |
| 4 |
| 8 |
```

5. L'objectif de cette question est de trier une pile de crêpes.

   On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

   On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

   Créer la fonction `tri_crepes` ayant pour paramètre une pile `P`. Cette fonction trie la pile `P` selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

   Exemple : Si la pile `P` est

   ```
   | 7  |
   | 8  |
   | 12 |
   | 14 |
   ```

   après l'appel de `tri_crepes(P)`, la pile `P` devient

   ```
   | 5  |
   | 7  |
   | 8  |
   | 12 |
   | 14 |
   ```

Exercice 3 : Métropole candidat libre 2021 sujet 1

On crée une classe Pile qui modélise la structure d'une pile d'entiers. Le constructeur de la classe initialise une pile vide. La définition de cette classe sans l'implémentation de ses méthodes est donnée ci-dessous.

```
class Pile:
    def __init__(self):
        """Initialise la pile comme une pile vide."""
    def est_vide(self):
        """Renvoie True si la liste est vide, False sinon."""
    def empiler(self, e):
        """Ajoute l'élément e sur le sommet de la pile, ne renvoie rien."""
    def depiler(self):
        """Retire l'élément au sommet de la pile et le renvoie."""
    def nb_elements(self):
        """Renvoie le nombre d'éléments de la pile."""
    def afficher(self):
        """Affiche de gauche à droite les éléments de la pile, du fond de la pile
vers son sommet."""
```

Seules les méthodes de la classe ci-dessus doivent être utilisées pour manipuler les objets Pile.

1.
   a. Écrire une suite d'instructions permettant de créer une instance de la classe `Pile` affectée à une variable `pile1` contenant les éléments `7`, `5` et `2` insérés dans cet ordre. Ainsi, à l'issue de ces instructions, l'instruction `pile1.afficher()` produit l'affichage : `7, 5, 2`.

   b. Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler()
pile1.empiler(5)
pile1.empiler(element1)
pile1.afficher()
```

2. On donne la fonction `mystere` suivante :

```
def mystere(pile, element):
    pile2 = Pile()
    nb_elements = pile.nb_elements()
    for i in range(nb_elements):
        elem = pile.depiler()
        pile2.empiler(elem)
        if elem == element:
            return pile2
    return pile2
```

   a. Dans chacun des quatre cas suivants, quel est l'affichage obtenu dans la console ?
   - Cas n°1
```
>>> pile.afficher()
7, 5, 2, 3
>>> mystere(pile, 2).afficher()
```
   - Cas n°2
```
>>> pile.afficher()
7, 5, 2, 3
>>> mystere(pile, 9).afficher()
```
   - Cas n°3
```
>>> pile.afficher()
7, 5, 2, 3
>>> mystere(pile, 3).afficher()
```
   - Cas n°4
```
>>> pile.est_vide()
True
>>> mystere(pile, 3).afficher()
```

   b. Expliquer ce que permet d'obtenir la fonction `mystere`.

3. Écrire une fonction `etendre(pile1, pile2)` qui prend en arguments deux objets Pile appelés `pile1` et `pile2` et qui modifie `pile1` en lui ajoutant les éléments de `pile2` rangés dans l'ordre inverse. Cette fonction ne renvoie rien.

Exemple :

```
>>> pile1.afficher()
7, 5, 2, 3
>>> pile2.afficher()
1, 3, 4
>>> etendre(pile1, pile2)
>>> pile1.afficher()
7, 5, 2, 3, 4, 3, 1
>>> pile2.est_vide()
True
```

4. Écrire une fonction `supprime_toutes_occurences(pile, element)` qui prend en arguments un objet `Pile` appelé `pile` et un élément `element` et supprime tous les éléments `element` de `pile`.

   Exemple :

```
>>> pile.afficher()
7, 5, 2, 3, 5
>>> supprime_toutes_occurences(pile, 5)
>>> pile.afficher()
7, 2, 3
```

Exercice 4 : Métropole 2021 sujet 2

Une méthode simple pour gérer l'ordonnancement des processus est d'exécuter les processus en une seule fois et dans leur ordre d'arrivée.

1. Parmi les propositions suivantes, quelle est la structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out) ?
   - Liste
   - Dictionnaire
   - Pile
   - File

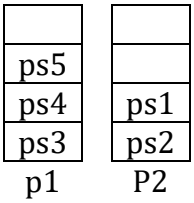
2. On choisit de stocker les données des processus en attente à l'aide d'une liste Python `lst`. On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`. Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.

On choisit maintenant d'implémenter une file `file` à l'aide d'un `couple(p1, p2)` où `p1` et `p2` sont des piles. Ainsi `file[0]` et `file[1]` sont respectivement les piles `p1` et `p2`.

Pour enfiler un nouvel élément `elt` dans `file`, on l'empile dans `p1`.
Pour défiler `file`, deux cas se présentent :

- La pile `p2` n'est pas vide : on dépile `p2`.
- La pile `p2` est vide : on dépile les éléments de `p1` en les empilant dans `p2` jusqu'à ce que `p1` soit vide, puis on dépile `p2`.

3. Réaliser une illustration du fonctionnement décrit précédemment.
4. On considère la situation représentée ci-dessous :



On exécute la séquence d'instructions suivante :

```
enfiler(file, ps6)
defiler(file)
defiler(file)
defiler(file)
enfiler(file, ps7)
```

Représenter le contenu final des deux piles à la suite de ces instructions.

5. On dispose des fonctions :

- `empiler(p, elt)` qui empile l'élément `elt` dans la pile `p`,
- `depiler(p)` qui renvoie le sommet de la pile `p` si `p` n'est pas vide et le supprime,
- `pile_vide(p)` qui renvoie `True` si la pile `p` est vide, `False` si la pile `p` n'est pas vide.

a. Écrire en Python une fonction `est_vide(f)` qui prend en argument un couple de piles `f` et qui renvoie `True` si la file représentée par `f` est vide, `False` sinon.
b. Écrire en Python une fonction `enfiler(f, elt)` qui prend en arguments un couple de piles `f` et un élément `elt` et qui ajoute `elt` en queue de la file représentée par `f`.
c. Écrire en Python une fonction `defiler(f)` qui prend en argument un couple de piles `f` et qui renvoie l'élément en tête de la file représentée par `f` en le retirant.

Exercice 5 : Centres Etrangers 2023 sujet 2

Simon est un jeu de société électronique de forme circulaire comportant quatre grosses touches de couleurs différentes : rouge, vert, bleu et jaune. Le jeu joue une séquence de couleurs que le joueur doit mémoriser et répéter ensuite. S'il réussit, une couleur parmi les 4 est ajoutée à la fin de la séquence. La nouvelle séquence est jouée depuis le début et le jeu continue. Dès que le joueur se trompe, la séquence est vidée et réinitialisée avec une couleur et une nouvelle partie commence.
Dans cet exercice, nous essaierons de reproduire ce jeu.
Les quatre couleurs sont stockées dans un tuple nommé couleurs :

```
couleurs = ("bleu", "rouge", "jaune", "vert")
```

Pour stocker la séquence à afficher, nous utiliserons une structure de file que l'on nommera `sequence` tout au long de l'exercice.

La file est une structure linéaire de type FIFO (First In First Out). Nous utiliserons durant cet exercice les fonctions suivantes :

Structure de données abstraite : File

- `creer_file_vide() : renvoie une file vide`
- `est_vide(f) : renvoie True si f est vide, False sinon`
- `enfiler(f, element) : ajoute element en queue de f`
- `defiler(f) : retire l'élément en tête de f et le renvoie`
- `taille(f) : renvoie le nombre d'éléments de f`

En fin de chaque séquence, le Simon tire au hasard une couleur parmi les 4 proposées. On utilisera la fonction `randint(a, b)` de la bibliothèque random qui permet d'obtenir un nombre entier compris entre a inclus et b inclus pour le tirage aléatoire.

Exemple : `randint(1, 5)` peut renvoyer `1`, `2`, `3`, `4` ou `5`.

1. Recopier et compléter, sur votre copie, les ... des lignes 3 et 4 de la fonction `ajout(f)` qui permet de tirer au hasard une couleur et de l'ajouter à une séquence. La fonction `ajout` prend en paramètre la séquence `f`, elle renvoie la séquence `f` modifiée (qui intègre la couleur ajoutée au format chaîne de caractères).

```
def ajout(f):
    couleurs = ("bleu", "rouge", "jaune", "vert")
    indice = randint(..., ...)
    enfiler(..., ...)
    return f
```

En cas d'erreur du joueur durant sa réponse, la partie reprend au début, il faut donc vider la file `sequence` pour recommencer à zéro en appelant `vider(sequence)` qui permet de rendre la file `sequence` vide sans la renvoyer.

2. Écrire la fonction `vider` qui prend en paramètre une séquence `f` et la vide sans la renvoyer.

Le Simon doit afficher successivement les différentes couleurs de la séquence. Ce rôle est confié à la fonction affich_seq(sequence), qui prend en paramètre la file de couleurs sequence, définie par l'algorithme suivant :

- on ajoute une nouvelle couleur à sequence ;
- on affiche les couleurs de la séquence, une par une, avec une pause de 0,5 s entre chaque affichage.

Une fonction `affichage(couleur)` (dont la rédaction n'est pas demandée dans cet exercice) permettra l'affichage de la couleur souhaitée avec couleur de type chaîne de caractères correspondant à une des 4 couleurs. La temporisation de 0,5 s sera effectuée avec la commande `time.sleep(0.5)`. Après l'exécution de la fonction `affich_seq`, la file `sequence` ne devra pas être vidée de ses éléments.

3. Recopier et compléter, sur la copie, les `...` des lignes 4 à 10 de la fonction `affich_seq(sequence)` ci-dessous :

```
def affich_seq(sequence):
    stock = creer_file_vide()
    ajout(sequence)
    while not est_vide(sequence):
        c = ...
        ...
        time.sleep(0.5)
        ...
    while ...:
        ...
```

4. Cette question est indépendante des précédentes : bien qu'elle fasse appel aux fonctions construites précédemment, elle peut être résolue même si le candidat n'a pas réussi toutes les questions précédentes.

   Nous allons ici créer une fonction `tour_de_jeu(sequence)` qui gère le déroulement d'un tour quelconque de jeu côté joueur. La fonction `tour_de_jeu` prend en paramètre la file de couleurs `sequence`, qui contient un certain nombre de couleurs.

   - Le jeu électronique Simon commence par ajouter une couleur à la séquence et affiche l'intégralité de la séquence.
   - Le joueur doit reproduire la séquence dans le même ordre. Il choisit une couleur via la fonction `saisie_joueur()`.
   - On vérifie si cette couleur est conforme à celle de la séquence.
   - S'il s'agit de la bonne couleur, on poursuit sinon on vide `sequence`.
   - Si le joueur arrive au bout de la séquence, il valide le tour de jeu et le jeu se poursuit avec un nouveau tour de jeu, sinon le joueur a perdu et le jeu s'arrête.

   La fonction `tour_de_jeu` s'arrête donc si le joueur a trouvé toutes les bonnes couleurs de `sequence` dans l'ordre, ou bien dès que le joueur se trompe.
   Après l'exécution de la fonction `tour_de_jeu`, la file `sequence` ne devra pas être vidée de ses éléments en cas de victoire.

   a. Afin d'obtenir la fonction `tour_de_jeu(sequence)` correspondant au comportement décrit ci-dessus, recopier le script ci-dessous et :
      - Compléter le `...`
      - Choisir parmi les propositions de syntaxes suivantes lesquelles correspondent aux `ZONES A, B, C, D, E` et `F` figurant dans le script et les y remplacer (il ne faut donc en choisir que six parmi les onze) :

      ```
      vider(sequence)
      defiler(sequence)
      enfiler(sequence, c_joueur)
      enfiler(stock, c_seq)
      enfiler(sequence, defiler(stock))
      enfiler(stock, defiler(sequence))
      affich_seq(sequence)
      while not est_vide(sequence):
      while not est_vide(stock):
      if not est_vide(sequence):
      if not est_vide(stock):
      ```

```
def tour_de_jeu(sequence):
    ZONE A
    stock = creer_file_vide()
    while not est_vide(sequence):
        c_joueur = saisie_joueur()
        c_seq = ZONE B
        if c_joueur ... c_seq:
            ZONE C
        else:
            ZONE D
    ZONE E
    ZONE F
```

b. Proposer une modification pour que la fonction se répète si le joueur trouve toutes les couleurs de la séquence (dans ce cas, une nouvelle couleur est ajoutée) ou s'il se trompe (dans ce cas, la séquence est vidée et se voit ajouter une nouvelle couleur). On pourra ajouter des instructions qui ne sont pas proposées dans la question a.

Exercice 6 : 2024 sujet 0

Pour son évaluation de fin d'année, l'institut d'Enseignement Néo-moderne (EN) a décidé d'adopter le principe du QCM. Chaque évaluation prend la forme d'une liste de questions numérotées de 0 à 19. Pour chaque question, 5 réponses sont proposées. Les réponses sont numérotées de 1 à 5. Exactement une réponse est correcte par question et chaque candidat coche exactement une réponse par question. Pour chaque évaluation, on dispose de la correction sous forme d'une liste `corr` contenant pour chaque question, la bonne réponse ; c'est-à-dire telle que `corr[i]` est la bonne réponse à la question `i`. Par exemple, on présente ci-dessous la correction de l'épreuve 0 :

```
corr0 = [4, 2, 1, 4, 3, 5, 3, 3, 2, 1, 1, 3, 3, 5, 4, 4, 5, 1, 3, 3]
```

Cette liste indique que pour l'épreuve 0, la bonne réponse à la question 0 est 4, et que la bonne réponse à la question 19 est 3.
Avant de mettre une note, on souhaite corriger les copies question par question ; c'est-à-dire associer à chaque copie, une liste de booléens de longueur 20, indiquant pour chaque question, si la réponse donnée est la bonne. Le candidat Tom Matt a rendu la copie suivante pour l'épreuve 0 :

```
copTM = [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3, 3, 3]
```

La liste de booléens correspondante est alors :

```
corrTM = [True, False, False, True, True, False, False, False, False,
False, False, False, False, True, False, False, False, False, True, True]
```

1. Écrire en Python une fonction `corrige` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la liste des booléens associée à la copie `cop` selon la correction `corr`.
   Par exemple, `corrige(copTM, corr0)` renvoie `corrTM`.
2. Écrire en Python une fonction `note` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la note attribuée à la copie `cop` selon la correction `corr`, sans construire de liste auxiliaire.

Par exemple, `note(copTM, corr0)` renvoie `6`.

L'institut EN souhaite automatiser totalement la correction de ses copies. Pour cela, il a besoin d'une fonction pour corriger des paquets de plusieurs copies. Un paquet de copies est donné sous la forme d'un dictionnaire dont les clés sont les noms des candidats et les valeurs sont les listes représentant les copies de ces candidats. On peut considérer un paquet `p1` de copies où l'on retrouve la copie de Tom Matt :

```
p1 = {
    ('Tom', 'Matt'): [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3,
3, 3, 3],
    ('Lambert', 'Ginne'): [2, 4, 2, 2, 1, 2, 4, 2, 2, 5, 1, 2, 5, 5, 3, 1,
1, 1, 4, 4],
    ('Carl', 'Roth'): [5, 4, 4, 2, 1, 4, 5, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2,
2, 3, 4],
    ('Kurt', 'Jett'): [2, 5, 5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 4, 1, 3, 1, 3,
2, 4, 4],
    ('Ayet', 'Finzerb'): [4, 3, 5, 3, 2, 1, 2, 1, 2, 4, 5, 5, 1, 4, 1, 5,
4, 2, 3, 4]
}
```

3. Écrire en Python une fonction `notes_paquet` qui prend en paramètre un paquet de copies `p` et une correction `corr` et qui renvoie un dictionnaire dont les clés sont les noms des candidats du paquet `p` et les valeurs sont leurs notes selon la correction `corr`.

   Par exemple, `notes_paquet(p1, corr0)` renvoie `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

La fonction `notes_paquet` peut faire appel à la fonction `note` demandée en question 2, même si cette fonction n'a pas été écrite.

4. Expliquer si on peut utiliser des listes de noms plutôt qu'un couple comme clés du dictionnaire.
5. Proposer une autre solution pour éviter les problèmes d'identification des candidats portant les mêmes prénoms et noms. Cette proposition devra prendre en compte la sensibilité des données et être argumentée succinctement.

Un ingénieur de l'institut EN a démissionné en laissant une fonction Python énigmatique sur son poste. Le directeur est convaincu qu'elle sera très utile, mais encore faut-il comprendre à quoi elle sert. Voici la fonction en question :

```
def enigme(notes):
    a = None
    b = None
    c = None
    d = {}
    for nom in notes:
        tmp = c
        if a == None or notes[nom] > a[1]:
            c = b
            b = a
            a = (nom, notes[nom])
        elif b == None or notes[nom] > b[1]:
            c = b
            b = (nom, notes[nom])
        elif c == None or notes[nom] > c[1]:
            c = (nom, notes[nom])
        else:
            d[nom] = notes[nom]
        if tmp != c and tmp != None:
            d[tmp[0]] = tmp[1]
    return (a, b, c, d)
```

6. Calculer ce que renvoie la fonction `enigme` pour le dictionnaire `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.
7. En déduire ce que calcule la fonction `enigme` lorsqu'on l'applique à un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes.
8. Expliquer ce que la fonction `enigme` renvoie s'il y a strictement moins de 3 entrées dans le dictionnaire passé en paramètre.
9. Écrire en Python une fonction `classement` prenant en paramètre un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes et qui, en utilisant la fonction `enigme`, renvoie la liste des couples `((prénom, nom), note)` des candidats classés par notes décroissantes.

   Par exemple, `classement({('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3})` renvoie `[(('Tom', 'Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4), (('Ayet', 'Finzerb'), 3), (('Carl', 'Roth'), 2)]`.

Le professeur Paul Tager a élaboré une évaluation particulièrement innovante de son côté. Toutes les questions dépendent des précédentes. Il est donc assuré que dès qu'un candidat s'est trompé à une question, alors toutes les réponses suivantes sont également fausses. M. Tager a malheureusement égaré ses notes, mais il a gardé les listes de booléens associées. Grâce à la forme particulière de son évaluation, on sait que ces listes sont de la forme `[True, True, ..., True, False, False, ..., False]`.
Pour recalculer ses notes, il a écrit les deux fonctions Python suivantes (dont la seconde est incomplète) :

```
def renote_express(copcorr):
    c = 0
    while copcorr[c]:
        c = c + 1
    return c

def renote_express2(copcorr):
    gauche = 0
    droite = len(copcorr)
    while droite - gauche > 1:
        milieu = (gauche + droite)//2
        if copcorr[milieu]:
            ...
        else:
            ...
    if copcorr[gauche]:
        return ...
    else:
        return ...
```

10. Compléter le code de la fonction Python `renote_express2` pour qu'elle calcule la même chose que `renote_express`.
11. Déterminer les coûts en temps de `renote_express` et `renote_express2` en fonction de la longueur `n` de la liste de booléens passée en paramètre.
12. Expliquer comment adapter `renote_express2` pour obtenir une fonction qui corrige très rapidement une copie pour les futures évaluations de M. Tager s'il garde la même spécificité pour ses énoncés. Cette fonction ne devra pas construire la liste de booléens correspondant à la copie corrigée, mais directement calculer la note.

Où est Chewbacca ?