Sources : `https://e-nsi.forge.aeif.fr/ecrit` *et dépôt de l'AEIF pour les sources Latex.*

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

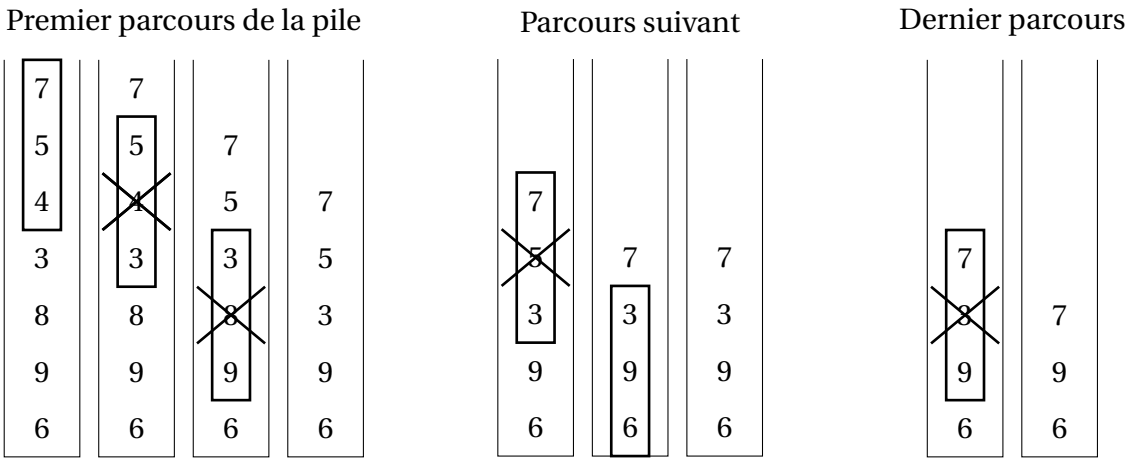
On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

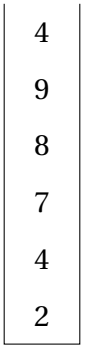
- Si la pile contient du haut vers le bas, le triplet 1 0 3, on supprime le 0.
- Si la pile contient du haut vers le bas, le triplet 1 0 8, la pile reste inchangée.

On parcourt la pile ainsi de haut en bas et on procède aux réductions. Arrivé en bas de la pile, on recommence la réduction en repartant du sommet de la pile jusqu'à ce que la pile ne soit plus réductible. Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

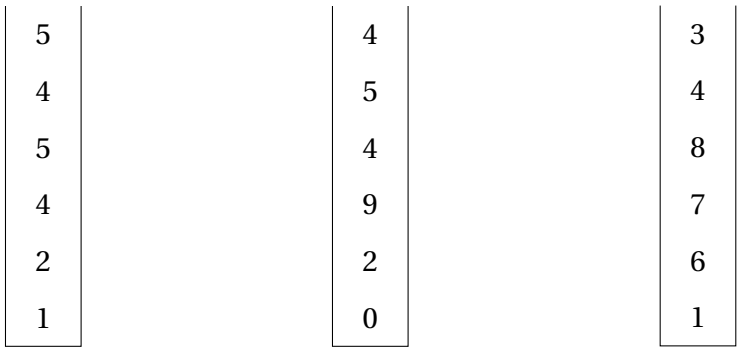
Voici un exemple détaillé de déroulement d'une partie.



1. **a.** Donner les différentes étapes de réduction de la pile suivante :



(b) Parmi les piles proposées ci-dessous, donner celle qui est gagnante.



Pile A Pile B Pile C

L'interface d'une pile est proposée ci-dessous. On utilisera uniquement les fonctions figurant dans le tableau suivant :

Structure de données abstraite : Pile
• `creer_pile_vide()` renvoie une pile vide • `est_vide(p)` renvoie `True` si `p` est vide, `False` sinon • `empiler(p, element)` ajoute `element` au sommet de `p` • `depiler(p)` retire l'élément au sommet de `p` et le renvoie • `sommet(p)` renvoie l'élément au sommet de `p` sans le retirer de `p` • `taille(p)` renvoie le nombre d'éléments de `p`

2. La fonction `reduire_triplet_au_sommet` permet de supprimer l'élément central des trois premiers éléments en partant du haut de la pile, si l'élément du bas et du haut sont de même parité. Les éléments dépilés et non supprimés sont replacés dans le bon ordre dans la pile.
Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et qui la modifie en place. Cette fonction renvoie le booléen `est_reduit` indiquant si le triplet du sommet a été réduit ou non.

```
def reduit_triplet_au_sommet(p):
    haut = depile(p)
    milieu = depile(p)
    bas = sommet(p)
    est_reduit = ...
    if haut % 2 != ...:
        empile(p, ...)
        ...
    empile(p, ...)
    return ...
```

3. On se propose maintenant d'écrire une fonction `parcourt_pile_en_reduisant` qui parcourt la pile du haut vers le bas en procédant aux réductions pour chaque triplet rencontré quand cela est possible.

La pile est toujours modifiée en place.

La fonction `parcourt_pile_en_reduisant` renvoie un booléen indiquant si la pile a été réduite à au moins une reprise lors du parcours.

a. Donner la taille minimale que doit avoir une pile pour être réductible.

b. Recopier et compléter le code suivant sur la copie :

```
def parcourt_pile_en_reduisant(p):
    q = pile_vide()
    reduction_pendant_parcours = False
    while taille(p) >= 3:
        if ...:
            reduction_pendant_parcours = ...
        e = depile(p)
        empile(q, e)
    while not est_vide(q):
        ...
        ...
    return ...
```

4. Partant d'une pile d'entiers `p`, on propose ici d'implémenter une fonction récursive `jouer` renvoyant la pile `p` entièrement simplifiée. On effectue donc autant de parcours que nécessaire.

Une fois la pile parcourue de haut en bas, on effectue un nouveau parcours à condition que le parcours précédent ait modifié la pile. Si à l'inverse, la pile n'a pas été modifiée, on ne fait rien, car la partie est terminée.

Recopier et compléter sur la copie le code ci-dessous :

```
def joue(p):
    if parcourt_pile_en_reduisant(...):
        ...(...)
```