Exercice 1

Cet exercice porte sur les structures de données.

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

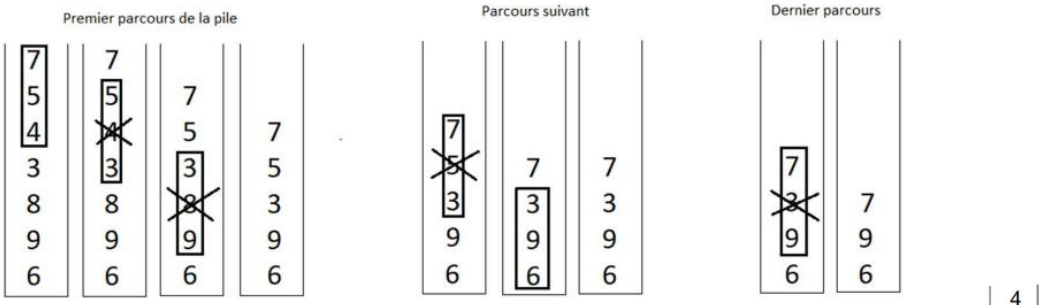
On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

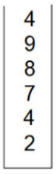
- Si la pile contient du haut vers le bas, le triplet 1 0 3, on supprime le 0.
- Si la pile contient du haut vers le bas, le triplet 1 0 8, la pile reste inchangée.

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.

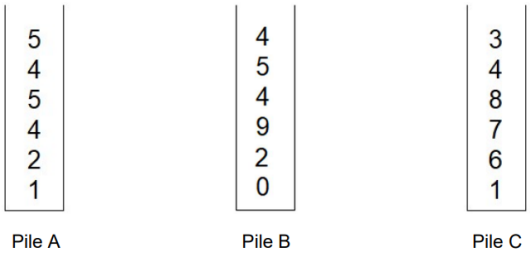
Voici un exemple détaillé de déroulement d'une partie.



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



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



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 ne renvoie donc rien.

```
def reduire_triplet_au_sommet(p):
      a = depiler(p)
      b = depiler(p)
      c = sommet(p)
      if a % 2 != .... :
               empiler(p, ...)
      empiler(p, ...)
```

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

   a. Donner la taille minimale que doit avoir une pile pour être réductible.
   b. Recopier et compléter sur la copie :

```
def parcourir_pile_en_reduisant(p):
       q = creer_pile_vide()
        while taille(p) >= ....:
              reduire_triplet_au_sommet(p)
              e = depiler(p)
              empiler(q, e)
        while not est_vide(q):
                 ..............
                  ..............
         return p
```

3. Partant d'une pile d'entiers p, on propose ici d'implémenter une fonction récursive jouer renvoyant la pile p entièrement simplifiée. Une fois la pile parcourue de haut en bas et réduite, on procède à nouveau à sa réduction à condition que cela soit possible. Ainsi :
   - Si la pile p n'a pas subi de réduction, on la renvoie.

- Sinon on appelle à nouveau la fonction `jouer` , prenant en paramètre la pile réduite.

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

```
def jouer(p):
     q = parcourir_pile_en_reduisant(p)
     if .............. :
            return p
     else:
          return jouer(...)
```

Exercice 2

Notion abordée : structures de données : les piles.

Dans cet exercice, on considère une pile d'entiers positifs. On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

`empiler(P, e)` : ajoute l'élément e sur la pile P ;

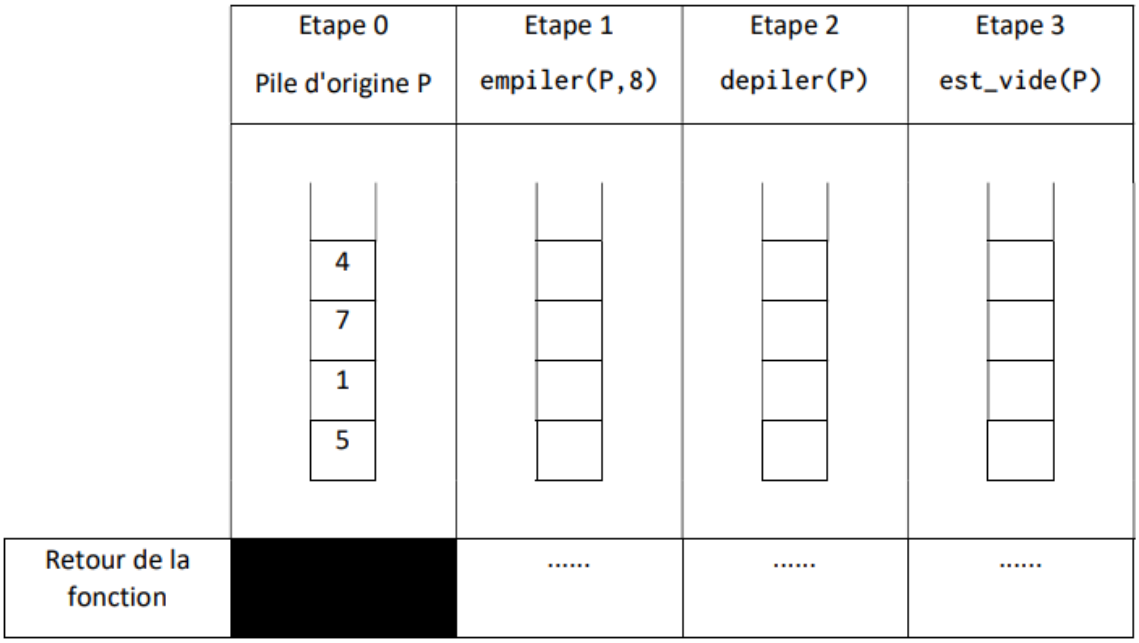
`depiler(P)` : enlève le sommet de la pile P et retourne la valeur de ce sommet ;

`est_vide(P)` : retourne True si la pile est vide et False sinon ;

`creer_pile()` : retourne une pile vide.

Dans cet exercice, seule l'utilisation de ces quatre fonctions sur la structure de données pile est autorisée.

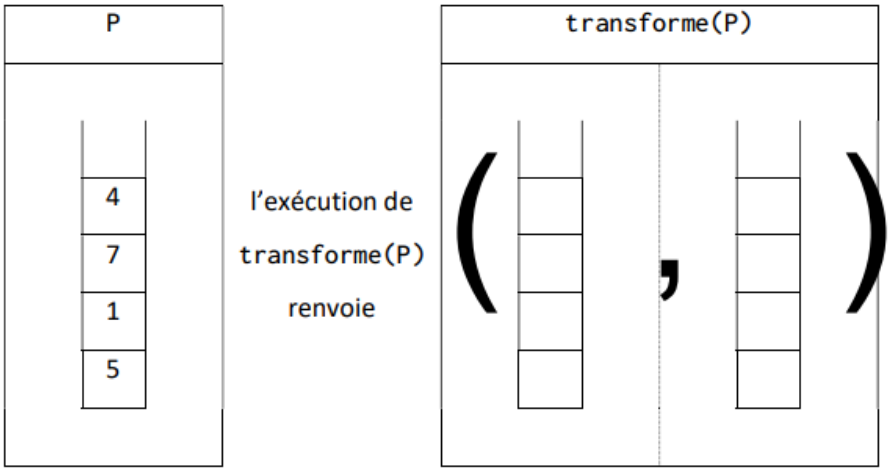
1. Recopier le schéma ci-dessous et le compléter sur votre copie en exécutant les appels de fonctions donnés. On écrira ce que renvoie la fonction utilisée dans chaque cas, et on indiquera None si la fonction ne retourne aucune valeur.



2. On propose la fonction ci-dessous, qui prend en argument une pile P et renvoie un couple de piles :

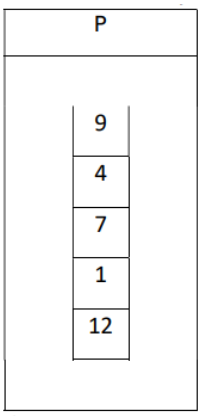
```
def transforme(P) :
   Q = creer_pile()
    while not est_vide(P) :
          v = depile(P)
      empile(Q,v)
return (P,Q)
```

Recopier et compléter sur votre copie le document ci-dessous



3. Ecrire une fonction en langage Python `maximum(P)` recevant une pile P comme argument et qui renvoie la valeur maximale de cette pile. On ne s'interdit pas qu'après exécution de la fonction, la pile soit vide.

On souhaite connaître le nombre d'éléments d'une pile à l'aide de la fonction `taille(P)`



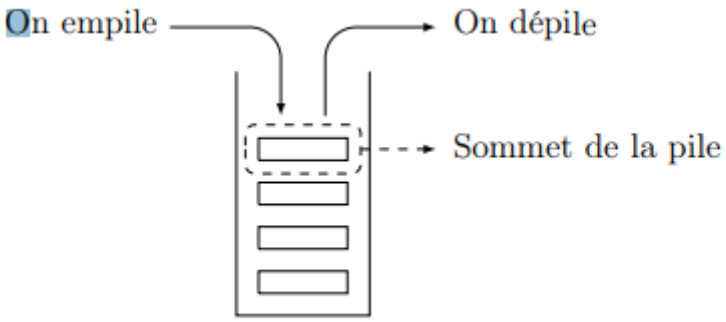
`taille(P)` retournera donc l'entier 5

4. 4.a. Proposer une stratégie écrite en langage naturel et/ou expliquée à l'aide de schémas, qui permette de mettre en place une telle fonction.
   4.b. Donner le code Python de cette fonction taille(P) (on pourra utiliser les cinq fonctions déjà programmées).

EXERCICE 3

Cet exercice porte sur la notion de pile et sur la programmation de base en Python.

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti » :



Une pile

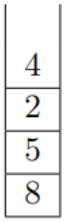
On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

Structure de données abstraite : Pile
Utilise : Éléments, Booléen

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 --> Rien
  empiler(pile, element) ajoute element au sommet de la pile
- depiler : Pile --> Élément
  depiler(pile) renvoie l'élément au sommet de la pile en le retirant de la pile

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



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

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

Question 2

1. 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.`

Recopier et 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 ???
```

2. 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`

Question 3 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(a), après l'appel de `retourner(P, 3)`, l'état de la pile P sera :

5
2
4
8

Question 4 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).

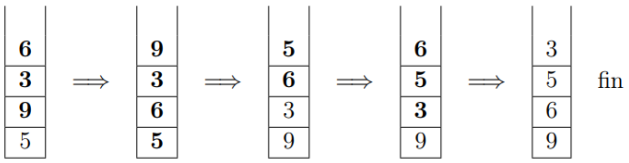
On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus.

Le principe est le suivant :

- On recherche la plus grande crêpe.
- On retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile.

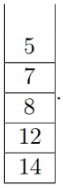
- On retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas.
- La plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

Exemple :



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 

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

.

Exercice 3

Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.

1.
   a. Expliquer en quelques mots se qu'est une fonction récursive.
   b. On considère la fonction Python suivante :

Numéro de lignes	Fonction `compte_rebours`
1	`def compte_rebours(n):`
2	`    """ n est un entier positif ou nul """`
3	`    if n >= 0:`
4	`        print(n)`
5	`        compte_rebours(n - 1)`

L'appel `compte_rebours(3)` affiche successivement les nombres 3, 2, 1 et 0. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 0.

2. En mathématiques, la factorielle d'un entier naturel n est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Par convention, la factorielle de 0 est 1. Par exemple :
   - la factorielle de 1 est 1
   - la factorielle de 2 est 2 x 1 =2
   - la factorielle de 3 est 3 x 2 x 1 = 6
   - la factorielle de 4 est 4 x 3 x 2 x 1 = 24…

Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive `fact` renvoie la factorielle de l'entier passé en paramètre de cette fonction.

Exemple : `fact(4)` renvoie 24

Numéro de lignes	Fonction `fact`
1	`def fact(n):`
2	`    """ Renvoie le produit des nombres entiers`
3	`    strictement positifs inférieurs à n """`
4	`    if n == 0:`
5	`        return` ***à compléter***
6	`    else:`
7	`        return` ***à compléter***

3. La fonction `somme_entiers_rec` ci-dessous permet de calculer la somme des entiers, de 0 à l'entier naturel n passé en paramètre.

Par exemple :

- Pour n = 0, la fonction renvoie la valeur 0.
- Pour n = 1, la fonction renvoie la valeur 0 + 1 = 1.

  …
- Pour n = 4, la fonction renvoie la valeur 0 + 1 + 2 + 3 + 4 = 10

Numéro de lignes	Fonction `somme_entiers_rec`
1	`def somme_entiers_rec(n):`
2	`    """ Permet de calculer la somme des entiers,`
3	`    de 0 à l'entier naturel n """`
4	`    if n == 0:`
5	`        return 0`
6	`    else:`
7	`        print(n) #pour vérification`
8	`        return n + somme_entiers_rec(n - 1)`

L'instruction `print(n)` de la ligne 7 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en œuvre au niveau des appels récursifs.

a. Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante : `res = somme_entiers_rec(3)`

b. Quelle valeur sera alors affectée à la variable `res` ?

4. Écrire en Python une fonction `somme_entiers` non récursive : cette fonction devra prendre en argument un entier naturel n et renvoyer la somme des entiers de 0 à n compris. Elle devra donc renvoyer le même résultat que la fonction `somme_entiers_rec` définie à la question 3.

Exemple : `somme_entiers(4)` renvoie 10.