Exercice 1 *5 points*

Écrire sur votre copie le numéro de la question et la lettre correspondant à l'unique bonne proposition.

1. Quelle opération ne fait pas partie de l'interface d'une pile ?
   - **a.** Ajouter un élément à la pile
   - **b.** Retirer l'élément le plus récent de la pile
   - **c.** Retirer l'élément le plus ancien de la pile
2. Quelle opération ne fait pas partie de l'interface d'une file ?
   - **a.** Ajouter un élément à la file
   - **b.** Retirer l'élément le plus récent de la file
   - **c.** Retirer l'élément le plus ancien de la file
3. L'opération `defile()` d'une file s'exécute en un temps qui est proportionnel au nombre de valeurs stockées dans la file
   - **a.** Vrai
   - **b.** Faux
4. Pour que deux implémentations d'un même type abstrait (file, pile, ...) soient interchangeables, il faut que :
   - **a.** La complexité en temps soit la même dans les deux cas
   - **b.** L'interface de la structure de données soit les même dans les deux cas
   - **c.** Les deux implémentations soient parfaitement identiques
5. Lors du développement de l'implémentation d'une structure de données, dans quel ordre effectue-t-on généralement les choses ?
   - **a.** On développe d'abord les mécanismes internes, desquels on déduit l'interface proposée par la structure de données.
   - **b.** On définit en premier l'interface de la structure de données, on propose ensuite du code qui l'implémente.

Exercice 2 *3 points*

On dispose d'une implémentation d'un type `pile` (cette implémentation n'est pas à refaire) qui offre l'interface suivante :

- `p = Pile()` qui permet de créer une pile vide `p`.
- `p.empile(n)` empile l'élément `n` sur la pile `p`.
- `p.depile()` renvoie l'élément du haut de la pile et le supprime.

On dispose de deux variables numériques `a` et `b`, ainsi qu'une pile `p` dont l'interface a été donnée plus haut.
Écrire un code qui échange les valeurs de `a` et de `b`, en **quatre** lignes exactement.
Il est interdit de faire intervenir une nouvelle variable (ou d'écrire `a, b = b, a`). Il faut impérativement se servir de la pile.

Exercice 3 *4 points*

À l'aide du type `list` de Python, créer une implémentation d'une structure de pile, qui disposera de l'interface suivante :

- `p = Pile()` permet de créer une pile vide.
- `p.est_vide()` renvoie un booléen disant si la pile est vide ou non.
- `p.empile(n)` empile l'élément `n` sur la pile `p`.
- `p.depile()` renvoie l'élément du haut de la pile et le supprime.

Exercice 4 *8 points*

On dispose d'une implémentation de type `pile` (cette implémentation n'est pas à refaire).

- `p = Pile()` permet de créer une pile vide.
- `p.est_vide()` renvoie un booléen disant si la pile est vide ou non.
- `p.empile(n)` empile l'élément `n` sur la pile `p`.
- `p.depile()` renvoie l'élément du haut de la pile et le supprime.

On dispose d'une implémentation de type `file` (cette implémentation n'est pas à refaire).

- `f = File()` permet de créer une file vide.
- `f.est_vide()` renvoie un booléen disant si la file est vide ou non.
- `f.enfile(n)` empile l'élément `n` au bas de la file `f`.
- `f.defile()` renvoie l'élément du haut de la file et le supprime.

1. À l'aide des interfaces ci-dessus, écrire une fonction `inverse(p)` qui prend en paramètre une pile `p` et inverse l'ordre de ses éléments. Cette fonction ne renvoie rien, la pile `p` passée en paramètre doit être modifiée par l'exécution de cette fonction.
2. À l'aide des interfaces ci-dessus, écrire une fonction `copie(p)` qui prend en paramètre la pile `p` et en crée une copie. (la solution `p2 = p` est interdite...)
   Cette fonction doit renvoyer une pile identique à celle passée en paramètre, et la pile initiale `p` ne doit pas être modifiée.