Conseils pratiques :

- *pour écrire votre code, le plus simple (pour l'indentation automatique, par exemple) est de l'écrire dans Thonny ou sur repl.it avant de le coller dans la zone bleue.*
- *Lorsque votre document est prêt à être rendu, cliquez sur «Rendre» dans Google Classroom.*

Ce devoir est composé de 5 exercices.

Exercice 1

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

```
Réponse : c
```

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

```
Réponse : b
```

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

```
Réponse : b
```

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

```
Réponse : b
```

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.

```
Réponse : b
```

6. Le type `list` utilisé dans Python correspond le mieux :
    a. Au type abstrait liste chaînée
    b. Au type abstrait tableau dynamique
    c. Au type abstrait file

```
Réponse : b
```

7. La récupération d'un élément d'un type `list` de Python, connaissant son indice :
    a. nécessite un temps proportionnel au nombre d'éléments de la liste
    b. s'effectue en temps constant
    c. est impossible

```
Réponse : b
```

8. La récupération d'un élément d'une liste chaînée, connaissant son indice :
    a. nécessite un temps proportionnel au nombre d'éléments de la liste
    b. s'effectue en temps constant
    c. est impossible

```
Réponse : a
```

9. Pour gérer les stocks de produits périssables, il est conseillé d'adopter une structure
    a. de pile
    b. de file

```
Réponse : b
```

10. Pour gérer les documents envoyés à l'imprimante, le système d'exploitation utilise une structure
    a. de pile
    b. de file

```
Réponse : b
```

Exercice 2

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

- `p = Pile()` permet de créer une pile vide
- `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.

```
Réponse :
>>> p.empile(a)
>>> p.empile(b)
>>> a = p.depile()
>>> b = p.depile()
```

Exercice 3

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 pas
- `p.empile(n)` empile l'élément n sur la pile `p`
- `p.depile()` renvoie l'élément du haut de la pile et le supprime.

```
Réponse :

class Pile:
    def __init__(self):
        self.data = []

    def est_vide(self):
        return len(self.data) == 0

    def empile(self,x):
        self.data.append(x)

    def depile(self):
        if self.est_vide() == True :
            raise IndexError("Vous avez essayé de dépiler une pile
vide !")
        else :
            return self.data.pop()
```

Exercice 4

On dispose d'une implémentation d'un 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 pas
- `p.empile(n)` empile l'élément n sur la pile `p`
- `p.depile()` renvoie l'élément du haut de la pile et le supprime.

À l'aide de ce type `pile`, implémentez un type `file` qui aura l'interface suivante :

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

Réponse :

```
class File:
    def __init__(self):
        self.entree = Pile()
        self.sortie = Pile()

    def est_vide(self):
        return self.entree.est_vide() and self.sortie.est_vide()

    def enfile(self,x):
        self.entree.empile(x)

    def defile(self):
        if self.est_vide():
            raise IndexError("File vide !")

        if self.sortie.est_vide() == True :
            while self.entree.est_vide() == False :
                self.sortie.empile(self.entree.depile())

        return self.sortie.depile()
```

Exercice 5

On dispose d'une implémentation d'un 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 pas
- `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 aussi d'une implémentation d'un 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 pas
- `f.enfile(n)` enfile l'élément n au bas de la file `f`
- `f.defile()` renvoie l'élément du haut de la file et le supprime.

Vous pouvez retrouver ces deux implémentations dans le repl.it ci-dessous : vous pouvez vous en servir pour répondre aux deux questions suivantes et tester vos résultats.
https://gist.github.com/glassus/32474fc68f5ac26e6da1bdfcf6331a67

Question 1

À l'aide des interfaces ci-dessus, écrire une procédure `inverse(p)` qui prend en paramètre une pile `p` et inverse l'ordre de ses éléments. Cette procédure ne renvoie rien, la pile passée en paramètre est modifiée par l'exécution de cette procédure.

```
Réponse :
# il suffit de vider toute la pile p dans une file f avant de la vider
# celle-ci dans la pile p

def inverse(p):
     f = File()
     while p.est_vide() == False :
          k = p.depile()
          f.enfile(k)
     while f.est_vide() == False :
          m = f.defile()
          p.empile(m)
```

Question 2

À l'aide des interfaces ci-dessus (et sans utiliser la question précédente), écrire une fonction `copie(p)` qui prend en paramètre une pile `p` et en crée une copie. Cette procédure renvoie une nouvelle pile identique à celle passée en paramètre. La pile initiale p n'est pas modifiée.

```
Réponse :
# il suffit de vider toute la pile p dans une autre pile p_temp.
# Puis de vider cette pile p_temp à la fois dans la nouvelle pile
# new_p et dans l'ancienne pile p (qui sinon resterait vide)


def copie(p):
    new_p = Pile()
    p_temp = Pile()
    while p.est_vide() ==  False :
        p_temp.empile(p.depile())
    while p_temp.est_vide() == False :
        k = p_temp.depile()
        new_p.empile(k)
        p.empile(k)
    return new_p
```