V - TP 1 Certains s'empilent, d'autres se défilent...

1 Des chapeaux et des bonhommes de neige

Cinq chapeaux empilés sont distribués, en commençant en haut et allant vers le bas, à cinq bons hommes de neige en commençant à gauche et finissant à droite. À la fin, chaque bonhomme de neige devrait recevoir un chapeau à sa taille.

Question 1. Quelle pile de chapeaux correspond à quelle rangée de bonshommes de neige ?

1.1 Une pile de chapeaux

On représente un chapeau par un objet de type `Chapeau` qui possède un attribut `taille` qui représente sa taille : ainsi le plus petit chapeau a une taille de 1, et le plus grand chapeau a une taille de 5. On modélise une **pile** de chapeaux par un objet de type `Pile`.
Les interfaces des classes `Chapeau` et `Pile` sont les suivantes :

- classe `Chapeau`

Fonction	Signature	Description
`Chapeau(c)`	`int -> Chapeau`	Renvoie un chapeau de taille `c`
`c.affiche()`	`() -> None`	Affiche le chapeau `c`

- classe `Pile`

Fonction	Signature	Description
`Pile()`	`() -> Pile`	Créer une pile vide
`p.est_vide()`	`() -> Bool`	Détermine si la pile `p` est vide.
`p.empiler(c)`	`Chapeau -> None`	Déposer le chapeau `c` au sommet de la pile `p`
`p.depiler()`	`() -> Chapeau`	Renvoie le chapeau au sommet de la pile `p`, lorsque cela est possible.
`p.affiche()`	`() -> None`	Affiche la pile `p` sans la modifier.

Code python

```python
from ds import Pile, Chapeau

c = Chapeau(1)
p = Pile()
p.empiler(c)
p.empiler(Chapeau(2))
p.affiche()
print(p.depiler())
```

Résultat

```
[Chap. 1, Chap. 2] (Sommet pile)
Chap. 2
```

Question 2. Écrire le code python d'une fonction `init_piles` qui renvoie les 4 piles `p1`, `p2`, `p3` et `p4` représentant respectivement les piles de chapeaux 1, 2, 3, et 4.

Code python

```python
def init_piles():
    p1 = Pile()
    # p1.empiler(...)
    # ...
    # À compléter
    return p1, p2, p3, p4
```

Code python

```python
for p in init_piles():
    p.affiche()
```

Résultat

```
[Chap. 1, Chap. 2, Chap. 3, Chap. 4, Chap. 5] (Sommet pile)
[Chap. 2, Chap. 1, Chap. 5, Chap. 3, Chap. 4] (Sommet pile)
[Chap. 4, Chap. 5, Chap. 1, Chap. 2, Chap. 3] (Sommet pile)
[Chap. 5, Chap. 4, Chap. 3, Chap. 2, Chap. 1] (Sommet pile)
```

1.2 Une file de bonshommes de neige

On représente un bonhomme de neige par un objet de type `Bonhomme` muni d'un attribut `taille` qui représente sa taille : ainsi le plus petit bonhomme de neige a une taille de 1, et le plus grand bonhomme de neige a une taille de 5. Si `b` est un objet de type `Bonhomme` et `c` un objet de type `Chapeau`, alors `b.content(c)` renvoie `True` si et seulement si le chapeau `c` est celui du bonhomme `b`.
On modélise une **file** de bonshommes de neige par un objet de type `File`, qui contient les bonshommes de neige.
Les interfaces des classes `Bonhomme` et `File` sont les suivantes :

- classe `Bonhomme`

Fonction	Signature	Description
`Bonhomme(b)`	`int -> Bonhomme`	Renvoie un bonhomme de taille `b`
`b.content(c)`	`Chapeau -> Bool`	Renvoie `True` si le chapeau `c` convient au bonhomme `b`
`b.affiche()`	`() -> None`	Affiche le bonhomme `b`

- classe `File`

Fonction	Signature	Description
`File()`	`() -> File`	Créer une file vide
`f.est_vide()`	`() -> Bool`	Détermine si la file `f` est vide.
`f.enfiler(c)`	`Bonhomme -> None`	Déposer le bonhomme `b` à la fin de la file `f`
`f.defiler()`	`() -> Bonhomme`	Renvoie le premier bonhomme de la file `f`, lorque cela est possible.
`f.affiche()`	`() -> None`	Affiche la file `f` sans la modifier.

Code python

```python
from ds import File, Bonhomme

b = Bonhomme(1)
f = File()
f.enfiler(b)
f.enfiler(Bonhomme(2))
f.affiche()
print(f.defiler())
```

Résultat

```
(Début file) [Bonh. 1, Bonh. 2]
Bonh. 1
```

Question 3. Écrire le code python d'une fonction `init_files` qui renvoie les 4 files `f1`, `f2`, `f3` et `f4` représentant respectivement les files de bonshommes de neige 1, 2, 3, et 4.

Code python

```python
def init_files():
    f1 = File()
    # f1.enfiler(...)
    # ...
    # À compléter.
    return f1, f2, f3, f4
```

Code python

```python
for f in init_files():
    f.affiche()
```

Résultat

```
(Début file) [Bonh. 1, Bonh. 2, Bonh. 3, Bonh. 4, Bonh. 5]
(Début file) [Bonh. 5, Bonh. 4, Bonh. 3, Bonh. 2, Bonh. 1]
(Début file) [Bonh. 4, Bonh. 3, Bonh. 5, Bonh. 1, Bonh. 2]
(Début file) [Bonh. 3, Bonh. 2, Bonh. 1, Bonh. 5, Bonh. 4]
```

1.3 Compatiblité entre pile de chapeaux et file de bonshommes

On dit qu'une pile de chapeaux est **compatible** avec une file de bonshommes de neige si et seulement si tous les bonhommes de neige sont content lorsqu'ils prennent le premier chapeau de la pile dans l'ordre imposé par la file.
On se propose de résoudre le problème de l'association d'une pile de chapeaux avec une file de bonshommes de neige. On souhaite écrire une fonction `compatibles` qui étant donné une pile de chapeaux `p` et une file de bonshommes de neige `f`, renvoie `True` si et seulement si les deux structures sont compatibles.

Code python

```
def compatibles(p, f):
    """ Pile, File -> Bool
    Détermine si les chapeaux sont associés aux bon bonshommes """
    pass
```

Code python

```
p1, p2, p3, p4 = init_piles()
f1, f2, f3, f4 = init_files()
print(compatibles(p1, f1))
print(compatibles(p1, f2))
```

Résultat

```
False
False
```

1.4 Le problème de la mutabilité et solution du problème

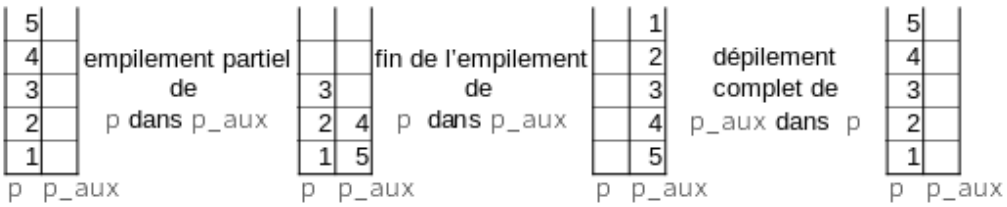
Question 4. **1.** Afficher le contenu des piles `p1` et `f1` **après** exécution de la fonction `compatibles`.

2. Que constate-t-on ? À quoi cela est-il dû ?

3. Quel problème cela va-t-il éventuellement poser ?

1.4.1 Modification de la fonction `compatibles`

On souhaite modifier le code de la fonction `compatibles` afin qu'à la fin de l'exécution de l'instruction `compatibles(p, f)`, les piles `p` et `f` soient dans le même état qu'au début.
Pour cela, lors de l'exécution de la fonction `compatibles`, à chaque dépilement (resp. défilement) de la pile `p` (resp. `f`) on empilera le chapeau `c` (resp. enfilera le bonhomme `b`) obtenu dans une pile `pile_aux` (resp. une file `file_aux`), et on s'assurera de restaurer l'état initial avant toute instruction `return`.
On écrira pour cela une fonction `restaure_pile(p, p_aux)` (resp. `restaure_file(f, f_aux)`) restaurant l'état initial de `p` à l'aide de `p_aux` (resp. `f` à l'aide de `f_aux`). On donne ci-dessous un schéma explicatif de l'algorithme de la fonction `restaure_pile`. Une idée similaire est à adapter pour la fonction `restaure_file`.



Code python

```
def restaure_pile(p, p_aux):
    """ Pile, Pile -> None
    Restaure p à son état initial lorsque les dépilements successifs
    on été empilés dans p_aux. """
    while not p.est_vide():
        c = p.depiler()
        p_aux.empiler(c)
    while not p_aux.est_vide():
        # À compléter
```

Code python

```python
def restaure_file(f, f_aux):
    """ File, File -> None
    Restaure f à son état initial lorsque les défilement successifs
    on été enfilés dans f_aux. """
    pass

def compatibles(p, f):
    """ Pile, File -> Bool
    Détermine si les chapeaux sont associés aux bons bonshommes
    L'état de la pile p et la file f sera le même avant et après exécution
    """
    pass
```

Code python

```python
p1, p2, p3, p4 = init_piles()
f1, f2, f3, f4 = init_files()
p1.affiche()
f1.affiche()
print(compatibles(p1, f1))
p1.affiche()
f1.affiche()
```

Résultat

```
[Chap. 1, Chap. 2, Chap. 3, Chap. 4, Chap. 5] (Sommet pile)
(Début file) [Bonh. 1, Bonh. 2, Bonh. 3, Bonh. 4, Bonh. 5]
False
[Chap. 1, Chap. 2, Chap. 3, Chap. 4, Chap. 5] (Sommet pile)
(Début file) [Bonh. 1, Bonh. 2, Bonh. 3, Bonh. 4, Bonh. 5]
```

1.4.2 Calcul de la solution du problème

Écrire une fonction `solution` qui renvoie la liste des couples solution au problème.
Par exemple, si la pile de chapeau numéro 1 correspond à la file de bonhomme de neige 2, alors on ajoutera l'entrée `(1, 2)` à la liste des couples solution.

Code python

```python
def solution():
    """ () -> [(int, int)]
    Renvoie la liste des associations pile i <-> file j """
    piles = init_piles()
    files = init_files()
    sol = []
    # À compléter
```

Code python

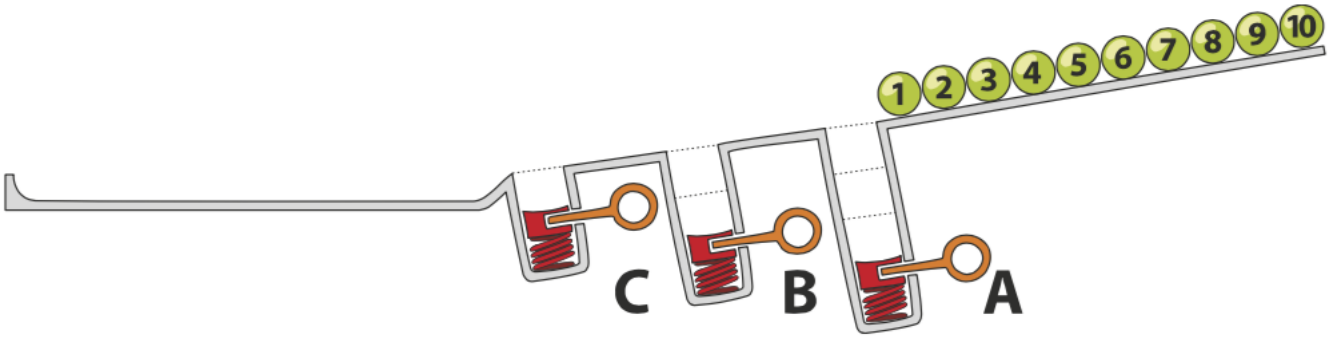
```python
print(solution())
```

Résultat

```
[(1, 2), (2, 3), (3, 4), (4, 1)]
```

2 Une rampe de billes

Sur une rampe, il y a 10 billes numérotées dans l'ordre. Le long de la rampe, il y a trois trous A, B et C : le trou A peut contenir trois billes au maximum, le trou B deux billes et le trou C une seule bille au maximum.
Quand les billes roulent sur la rampe, elles tombent successivement dans les trous jusqu'à ce qu'elles les remplissent (les billes 1, 2 et 3 tombent dans le trou A, les billes 4 et 5 tombent dans le trou B et la bille 6 tombe dans le trou C). Les autres billes passent par-dessus et continuent leur chemin jusqu'à la fin de la rampe.
Quand toutes les billes ont parcouru la rampe, les ressorts, placés dans les trous A à C, éjectent les billes qu'ils contenaient : d'abord, les trois billes du trou A, ensuite, celles du trou B et finalement, celle du trou C. Les billes sont ainsi poussées sur la rampe. On attend que toutes les autres billes aient passé avant qu'un ressort ne soit relâché.



Question 5. **1.** Dans quel ordre les billes de la séquence 1 à 10 seront-elles alignées à la fin ?

2. Sur une installation du même type, dix billes initialement ordonnées de 1 à 10 sur la rampe arrivent dans l'ordre : 8, 9, 10, 4, 3, 2, 1, 5, 7, 6.

Combien de trous y avait-il sur le parcours, et quelle est leur profondeur ? On justifiera la réponse.

3. Compléter la fonction `arrivee_rampe(profondeurs)` qui prend en paramètre une liste d'entiers donnant la profondeur de chacun des trous sur une installation du même type que celle vue précédemment, dans l'ordre dans lequel ils sont positionnés sur le parcours, et qui renvoie la liste des entiers 1 à 10, dans l'ordre dans lequel les billes 1 à 10 arriveraient sur la rampe.

On utilisera pour cela les classes `Pile` et `File` définies précédemment. Ces classes ont toutes les deux été enrichies d'un attribut `max_elt` et d'une méthode `est_pleine` qui renvoie `True` si et seulement si la pile (resp. file) contient `max_elt` éléments. Lors d'un empilement (resp. enfilement), on vérifie d'abord que la pile (resp. file) n'est pas pleine. Si la pile (resp. file) est pleine, alors on soulève une exception. On appelle ce genre de pile (resp. files) des piles (resp. files) **bornées**.

Fonction	Signature	Description
`Pile(max_elt = n)`	`int -> Pile`	Renvoie une pile bornée de capacité maximale `n`
`p.est_pleine()`	`() -> Bool`	Détermine si la pile a atteint sa capacité maximale.
`File(max_elt = n)`	`int -> File`	Renvoie une file bornée de capacité maximale `n`
`f.est_pleine()`	`() -> Bool`	Détermine si la file a atteint sa capacité maximale.

Code python

```python
def arrivee_rampe(profondeur):
    """ [int] -> [int]
    Détermine l'ordre d'arrivée des boules numérotées de 1 à 10 """
    rampe_lancement = File()
    for b in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
        ... # à compléter
    piles = [Pile(max_elt = p) for p in profondeur] # liste de piles bornées, dans l'ordre du parcours.
    rampe_finale = File() # billes dans l'ordre d'arrivée
    numero_pile = 0 # indice de la pile à remplir

    # on remplit les piles dans l'ordre dans lequel elles sont
    # positionnées sur le parcours tant que cela est possible
    # (c'est à dire tant que la dernière pile du parcours n'est pas vide)
    # Si la pile que l'on cherche à remplir est pleine,
    # alors il faut commencer par changer le numéro de la pile à remplir.
    # On empile une bille prise depuis la rampe de lancement sur la pile à remplir.
    while ...:
        ... # à compléter

    # Toutes les billes restantes roulent directement dans la rampe d'arrivée.
    while ...:
        ... # à compléter

    # On retire les ressorts dans l'ordre dans lesquels ils sont
    # positionnés sur le parcours. Les billes ressortent des trous et
    # viennent s'accumuler sur la rampe de lancement.
    for p in piles:
        while ...:
            ... # à compléter

    return rampe_finale
```

Code python

```python
print(arrivee_rampe([3, 2, 1]))
print(arrivee_rampe([4, 1, 2]))
print(arrivee_rampe([5, 5]))
```

Résultat

```
(Début file) [7, 8, 9, 10, 3, 2, 1, 5, 4, 6]
(Début file) [8, 9, 10, 4, 3, 2, 1, 5, 7, 6]
(Début file) [5, 4, 3, 2, 1, 10, 9, 8, 7, 6]
```

3 Conclusion

Question 6. En anglais, l'acronyme `FILO` signifie `Firt In Last Out` (premier entré dernier sorti), et l'acronyme `FIFO` signifie `Fist In First Out` (premier entré premier sorti).
Quel acronyme peut s'appliquer à une pile ? À une file ?