Lycée Les 3 Sources — **Bourg-Lès-Valence**
NSI T^le — Année **2023-24**

Test n°2

Nom et prénom :

Exercice 1 : (3pt) On dispose des fonctions suivantes pour manipuler des piles.

Fonction	Description
`cree_pile()`	Renvoie une nouvelle pile vide.
`empile(element, pile)`	Rajoute `element` au sommet de `pile`.
`depile(pile)`	Renvoie l'élément se trouvant au sommet de `pile` et l'enlève. La pile ne doit pas être vide.
`est_vide(pile)`	Renvoie un booléen indiquant si `pile` est vide ou non.

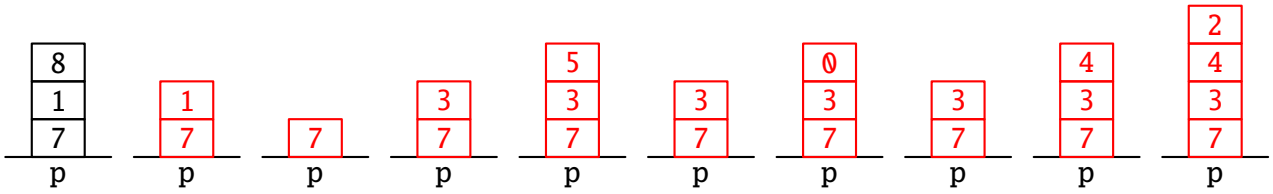
On part avec une pile représentée ci-contre et les instructions suivantes :

```
1 >>> v = depile(p)
2 >>> v = depile(p)
3 >>> empile(3, p)
```

```
4 >>> empile(5, p)
5 >>> v = depile(p)
6 >>> empile(0, p)
```

```
7 >>> v = depile(p)
8 >>> empile(4, p)
9 >>> empile(2, p)
```

1) Représenter l'état de la pile après chaque instruction.
2) Indiquer, dans l'ordre, les différentes valeurs prises par la variable `v` tout au long des instructions. **Solution :** : 8, 1, 5, 0



Exercice 2 : (3pt) On dispose des fonctions suivantes pour manipuler des files.

Fonction	Description
`cree_file()`	Renvoie une nouvelle file vide.
`enfile(element, file)`	Rajoute `element` à la fin de la **`file`**.
`defile(file)`	Renvoie l'élément se trouvant au début de **`file`** et l'enlève. La file ne doit pas être vide.
`est_vide(file)`	Renvoie un booléen indiquant si **`file`** est vide ou non.

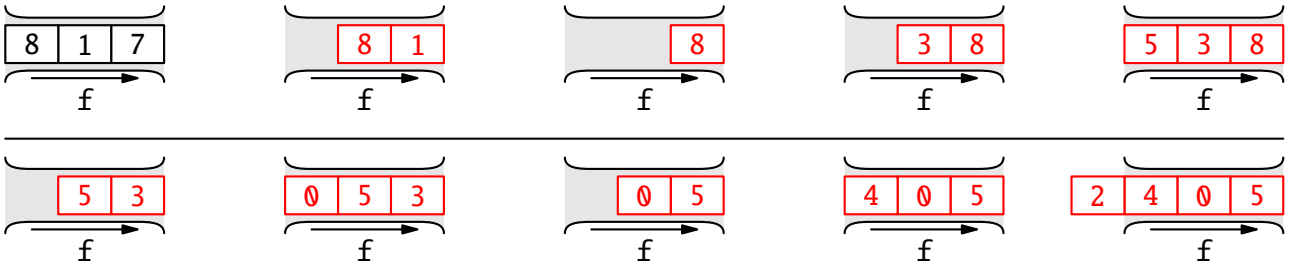
On part avec une file représentée ci-dessous et les instructions suivantes :

```
1 >>> v = defile(f)
2 >>> v = defile(f)
3 >>> enfile(3, f)
```

```
4 >>> enfile(5, f)
5 >>> v = defile(f)
6 >>> enfile(0, f)
```

```
7 >>> v = defile(f)
8 >>> enfile(4, f)
9 >>> enfile(2, f)
```

1) Représenter ci-dessous les étapes prises par la file après chaque instruction. L'entrée est à gauche et la sortie est à droite de chaque file.
2) Indiquer, dans l'ordre, les différentes valeurs prises par la variable `v` tout au long des instructions. **Solution :** : 7, 1, 8, 3



Exercice 3 : (11pt) Pour un jeu vidéo, on souhaite créer une classe pour le sac-à-dos du héro. Pour cela, on a préalablement défini une classe `Objet` qui possède l'interface suivante :

Méthode	Description
`__init__(self, nom, poids, prix)`	Constructeur.
`nom(self)`	Renvoie un texte correspondant au nom de l'objet.
`poids(self)`	Renvoie un réel correspondant au poids, en kg, de l'objet.
`prix(self)`	Renvoie un entier correspondant au prix, en pièces d'or, de l'objet.

Voici quelques exemples d'objets :

```
epee_hero = Objet("épée héro", 5, 100)
bouclier_hero = Objet("bouclier hero", 8, 80)
lingot_or = Objet("lingot d'or", 1, 400)
```

Un sac-à-dos est défini par les attributs suivants :

Attribut	Description
`liste`	Une liste Python contenant les objets qui sont dans le sac.
`poids_total`	Un réel qui correspond au poids total des objets du sac.
`capacite_max`	Un réel qui correspond à la capacité maximale que peut contenir le sac.

Voici les méthodes de la classe `Sac_a_dos` :

Méthode	Description
`__init__(self, capacite_max)`	Constructeur. Au début, le sac est vide.
`ajoute(self, objet)`	Rajoute `objet` dans le sac, s'il y a encore de la place, sinon lève une exception.
`enleve(self, objet)`	Enlève `objet` du sac, s'il y est. Sinon, une exception est levée.
`prix_total(self)`	Renvoie un entier correspondant à la valeur totale des objets contenus dans le sac.
`donne(self, other, objet)`	Enlève `objet` de `self` et l'ajoute au sac `other`.

1) Quelle instruction permet de créer l'objet `potion` qui s'appelle `"Potion"`, vaut 10 pièces d'or et pèse 0,1 kg ? **Solution :** `potion = Objet("Potion", 0.1, 10)`
2) On considère la suite d'instructions :

```
sac = Sac_a_dos(50)
sac.ajoute(lingot_or)
sac.ajoute(bouclier_hero)
```

Quelle est la valeur des expressions suivantes ?

1) `sac.poids_total` 2) `sac.prix_total()` 3) `sac.liste[0].nom()`

Solution :

1) `9` 2) `480` 3) `"lingot d'or"`

3) Quelle instruction permet d'enlever le lingot d'or de `sac` ?
Solution : `sac.enleve(lingot_or)`
4) Quelle instruction permet de donner le bouclier au héro dont le sac s'appelle `sac2` ?
Solution : `sac.donner(sac2, bouclier_hero)`

5) Écrire le code du constructeur **__init__**. Au début, le sac est vide.

```
def __init__(self, capacite_max):
    self.capacite_max = capacite_max
    self.poids_total = 0
    self.liste = []
```

6) Écrire le code de la méthode `ajoute` qui rajoute `objet` dans le sac, s'il en a encore de la place et met à jour le poids total. Sinon, une exception `ValueError` est levée avec le message `'sac plein'` avec l'instruction **raise** `ValueError('sac plein')`.

```
def ajoute(self, objet):
    if self.poids_total + objet.poids() <= self.capacite_max:
        self.liste.append(objet)
        self.poids_total += objet.poids()
    else:
        raise ValueError('sac plein')
```

7) Écrire le code de la méthode `enleve` qui enlève `objet` du sac, s'il y est, et met à jour le poids total. Si ce n'est pas le cas, une exception `KeyError` est levée.
Avec une liste Python, `liste.remove(val)` permet d'enlever la valeur `val` de `liste`.

```
def enleve(self, objet):
    if objet in self.liste:
        self.liste.remove(objet)
        self.poids_total -= objet.poids()
    else:
        raise KeyError('objet non trouvé')
```

8) Écrire le code de la méthode `prix_total` qui renvoie un entier correspondant à la valeur totale des objets contenus dans le sac.

```
def prix_total(self):
    total = 0
    for objet in self.liste:
        total += objet.prix()
    return total
```

9) Écrire le code de la méthode `donne` qui enlève `objet` de **self** et l'ajoute au sac `other`. **On suppose que le transfert peut avoir lieu**. Vous pouvez (et devez) utiliser des méthodes définies précédemment.

```
def donne(self, other, objet):
    self.enleve(objet)
    other.ajoute(objet)
```