EXERCICE 2 (4 points)

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

1.

a. Expliquer en quelques mots ce 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
2
3
4
5
6
7
8
	`def somme_entiers_rec(n):`
	`    """ Permet de calculer la somme des entiers,`
	`    de 0 à l'entier naturel n """`
	`    if n == 0:`
	`        return 0`
	`    else:`
	`        print(n) #pour vérification`
	`        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.

EXERCICE 4 (4 points)

Cet exercice traite du thème « structures de données », et principalement des piles.

La classe `Pile` utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparaît à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

```
>>> mapile = Pile()
>>> mapile.empiler(2)
>>> mapile
[2]
>>> mapile.empiler(3)
>>> mapile.empiler(50)
>>> mapile
[2, 3, 50]
>>> mapile.depiler()
50
>>> mapile
[2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

```
1 def est_triee(self):
2   if not self.est_vide() :
3      e1 = self.depiler()
4      while not self.est_vide():
5          e2 = self.depiler()
6          if e1 ... e2 :
7              return False
8          e1 = ...
9     return True
```

1. Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

On créé dans la console la pile `A` représentée par `[1, 2, 3, 4]`.

2. a. Donner la valeur renvoyée par l'appel `A.est_triee()`.
   b. Donner le contenu de la pile `A` après l'exécution de cette instruction.

On souhaite maintenant écrire le code d'une méthode `depileMax` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.
Après l'exécution de `p.depileMax()`, le nombre d'éléments de la pile `p` diminue donc de 1.

```
1   def depileMax(self):
2       assert not self.est_vide(), "Pile vide"
3       q = Pile()
4       maxi = self.depiler()
5       while not self.est_vide() :
6          elt = self.depiler()
7          if maxi < elt :
8             q.empiler(maxi)
9             maxi = ...
10          else :
11             ...
12       while not q.est_vide():
13          self.empiler(q.depiler())
14       return maxi
```

3. Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

On créé la pile `B` représentée par [9, -7, 8, 12, 4] et on effectue l'appel `B.depileMax()`.

4. a. Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle `while` de la ligne 5.
   b. Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.
   c. Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depileMax`.

On donne le code de la méthode `traiter()` :

```
1   def traiter(self):
2       q = Pile()
3       while not self.est_vide():
4           q.empiler(self.depileMax())
5       while not q.est_vide():
6           self.empiler(q.depiler())
```

5. a. Donner les contenus successifs des piles `B` et `q`
   - avant la ligne 3,
   - avant la ligne 5,
   - à la fin de l'exécution de la fonction `traiter`

   lorsque la fonction `traiter` est appliquée sur la pile `B` contenant [1, 6, 4, 3, 7, 2].

   b. Expliquer le traitement effectué par cette méthode.