ET_Récursivité_Algorithmique

Exercice 1 : Récursivité

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
2
3
4
5
	`def compte_rebours(n):`
	`    """ n est un entier positif ou nul """`
	`    if n >= 0:`
	`        print(n)`
	`        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
2
3
4
5
6
7
	`def fact(n):`
	`    """ Renvoie le produit des nombres entiers`
	`    strictement positifs inférieurs à n """`
	`    if n == 0:`
	`        return` ***à compléter***
	`    else:`
	`        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 2 : Algorithmique

Cet exercice traite de programmation orientée objet en Python et d'algorithmique.

Un pays est composé de différentes régions. Deux régions sont voisines si elles ont au moins une frontière en commun. L'objectif est d'attribuer une couleur à chaque région sur la carte du pays sans que deux régions voisines aient la même couleur et en utilisant le moins de couleurs possibles.
La figure 1 ci-dessous donne un exemple de résultat de coloration des régions de la France métropolitaine.

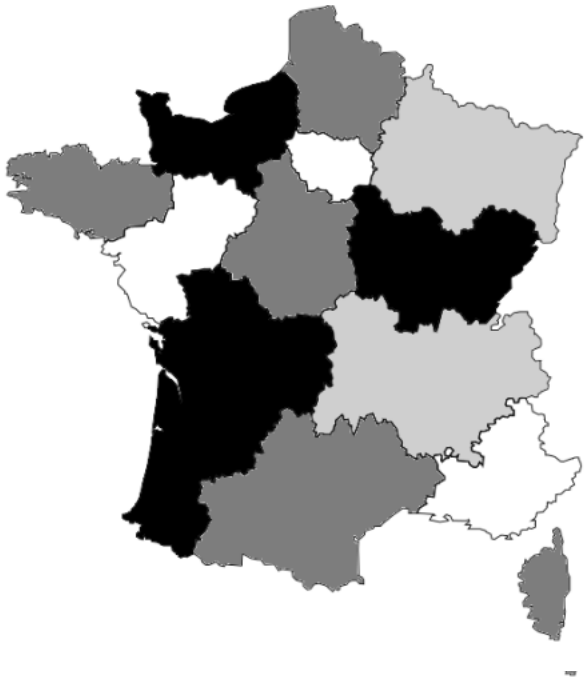

Figure 1 – Carte coloriée des régions de France métropolitaine

On rappelle quelques fonctions et méthodes des tableaux (le type `list` en Python) qui pourront être utilisées dans cet exercice :

- `len(tab)` : renvoie le nombre d'éléments du tableau `tab` ;
- `tab.append(elt)` : ajoute l'élément `elt` en fin de tableau `tab` ;
- `tab.remove(elt)` : enlève la première occurrence de `elt` de `tab` si `elt` est dans `tab`. Provoque une erreur sinon.

Exemple :

- `len([1, 3, 12, 24, 3])` renvoie `5` ;
- avec `tab = [1, 3, 12, 24, 3]`, l'instruction `tab.append(7)` modifie `tab` en `[1, 3, 12, 24, 3, 7]` ;
- avec `tab = [1, 3, 12, 24, 3]`, l'instruction `tab.remove(3)` modifie `tab` en `[1, 12, 24, 3]`.

Les deux parties de cet exercice forment un ensemble. Cependant, il n'est pas nécessaire d'avoir répondu à une question pour aborder la suivante. En particulier, on pourra utiliser les méthodes des questions précédentes même quand elles n'ont pas été codées.

Pour chaque question, toute trace de réflexion sera prise en compte.

Partie 1

On considère la classe `Region` qui modélise une région sur une carte et dont le début de l'implémentation est :

```
1   class Region:
2       '''Modélise une région d'un pays sur une carte.'''
3       def __init__(self, nom_region):
4           '''
5           initialise une région
6           : param nom_region (str) le nom de la région
7           '''
8           self.nom = nom_region
9           # tableau des régions voisines, vide au départ
10          self.tab_voisines = []
11          # tableau des couleurs disponibles pour colorier
    la région
12          self.tab_couleurs_disponibles = ['rouge', 'vert',
    'bleu', 'jaune', 'orange', 'marron']
13          # couleur attribuée à la région et non encore
    choisie au départ
14          self.couleur_attribuee = None
```

1. Associer, en vous appuyant sur l'extrait de code précédent, les noms `nom`, `tab_voisines`, `tab_couleurs_disponibles` et `couleur_attribuee` au terme qui leur correspond parmi : *objet*, *attribut*, *méthode* ou *classe*.

2. Indiquer le type du paramètre `nom_region` de la méthode `__init__` de la classe `Region`.

3. Donner une instruction permettant de créer une instance nommée `ge` de la classe `Region` correspondant à la région dont le nom est « Grand Est ».

4. Recopier et compléter la ligne 6 de la méthode de la classe `Region` ci-dessous :

```
1   def renvoie_premiere_couleur_disponible(self):
2       '''
3       Renvoie la première couleur du tableau des couleurs
    disponibles supposé non vide.
4       : return (str)
5       '''
6       return ...
```

5. Recopier et compléter la ligne 6 de la méthode de la classe `Region` ci-dessous :

```
1  def renvoie_nb_voisines(self) :
2      '''
3      Renvoie le nombre de régions voisines.
4      : return (int)
5      '''
6      return ...
```

6. Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 6 :

```
1  def est_coloriee(self):
2      '''
3      Renvoie True si une couleur a été attribuée à cette
   région et False sinon.
4      : return (bool)
5      '''
6      ...
```

7. Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 8 :

```
1  def retire_couleur(self, couleur):
2      '''
3      Retire couleur du tableau de couleurs disponibles de
   la région si elle est dans ce tableau. Ne fait rien
   sinon.
4      : param couleur (str)
5      : ne renvoie rien
6      : effet de bord sur le tableau des couleurs
   disponibles
7      '''
8      ...
```

8. Compléter la méthode de la classe `Region` ci-dessous, à partir de la ligne 7, <u>en utilisant une boucle</u> :

```
1  def est_voisine(self, region):
2      '''
3      Renvoie True si la region passée en paramètre est une
   voisine et False sinon.
4      : param region (Region)
5      : return (bool)
6      '''
7      ...
```