Master SVS **Examen – UE 34** décembre 2018

Tout équipement de calcul, programmation, communication est interdit. Le sujet comporte 2 parties notées séparément, chacune d'elles étant composée de plusieurs exercices indépendants. Lire d'abord la totalité de l'énoncé, et commencer par ce qui vous semble le plus facile. ***Si besoin, on pourra supposer certaines questions résolues.***

PARTIE 1

Exercice 1 : (**Qui suis-je ?**)

1. En considérant la fonction `qui_suis_je` ci-dessous, donnez le résultat des instructions suivantes :

(a) `qui_suis_je({},"toto")`

(b) `qui_suis_je("a":1, "b":2,10)`

(c) `qui_suis_je({1:"a", 2:"b"},"z")`

(d) `qui_suis_je([1,2,3,1],1)`

```
def qui_suis_je(dic,a):
    for e in dic:
        dic[e] = dic[e]+a
    return(dic)
```

```
def qui_suis_je_2(L):
    LL=[]
    i=0
    while i<=len(L):
        LL = LL+[L[:i]]
        i=i+1
    return(LL)
```

2. Que fait la fonction `qui_suis_je_2` ?

Exercice 2 : (**Palindrome**)

Une séquence palindromique est une séquence d'acides nucléiques ADN ou ARN identique lorsqu'elle est lue dans le sens 5' → 3' sur un brin ou dans le sens 5' → 3' sur le brin complémentaire. C'est par exemple le cas de la séquence ci-contre, qui est reconnue par l'enzyme appelée EcoRI, une enzyme de restriction d'E. coli.

```
5'  -GAATTC-  3'
3'  -CTTAAG-  5'
```

Écrire une fonction qui prend en paramètre une chaîne de caractères représentant une séquence d'ARN ou d'ADN et qui dit si elle est un palindrome.

Exercice 3 : (**Un réseau génétique**)

Nous proposons de faire une simulation de réseaux de régulation génétiques. Dans tout ce problème, on considère un réseaux de régulation ne comportant que 3 gènes : A, B et C. Les interactions sont les suivantes :

- si A est présent, il active B,
- si A est présent, il inhibe C, autrement dit, lorsque A est absent, il active C
- si B est présent, il active A,
- si B est présent, il active C,
- si C est présent, il active A.

On se propose faire évoluer ce système en partant de l'état initial où seul C est présent.

1. On répresente l'état du système par une liste contenant 3 booléens, où le premier représentera la présence ou l'absence de A, et de manière similaire pour B et C.
   Quelle sera la représentation en python de l'état initial du système ?
2. On considère ici que toutes les interactions possibles ont lieu en même temps. Écrire la fonction `etatsuivant(...)` qui prend en argument une liste représentant l'état du système et qui renvoie une nouvelle liste représentant l'état du système après que toutes les interactions possibles aient eu lieu.
3. Écrire la fonction `simu(...)` qui prend en argument un état initial et un entier n et qui calcule les n états successifs de l'état initial. Le résultat sera rendu sous forme de liste d'états.

Exercice 4 : (**Jeu des erreurs**) Pour chacune des fonctions suivantes, trouvez l'erreur et corrigez-la.

```
def erreur1(liste,a):
  res = []
  i   = 0
  while i < len(liste):
    res = res + [liste[i]+a]
  return(res)
```

```
def erreur2(liste_de_str):
    res = ""
    for e in liste_de_str:
        res = res + [e]
    return(res)
```

```
def erreur3(dictionnaire):
    liste=[]
    for e in dictionnaire:
        if dictionnaire[e] != 0:
            liste == liste + [e,dictionnaire[e]]
    return(liste)
```

PARTIE 2

Exercice 5 : Écrivez une fonction `isSequence` qui prend en entrée une chaîne de caractères et retourne un booléen qui dit si cette chaîne ne contient que des caractères `A T G C` ou `N`. Ici on autorise des séquences issues d'un séquençage où la lecture de certaines bases est douteuse, et dans ce cas on écrit un "`N`" à la place des bases douteuses. Par exemple, `isSequence("AAATTNGNC")` et `isSequence("ACCGC")` retournent 1 (c'est-à-dire vrai) alors que `isSequence("AVVTTNGNC")` retourne 0 (faux) à cause de la présence de "V".

Exercice 6 : (**Un modèle simplifié de morphogénèse**) Les modèles de morphogénèse sont utiles pour expliquer le développement des formes biologiques : colonies d'unicellulaires, arborescences végétales, formes de coquillages, etc. Certains des modèles permettent de mieux comprendre comment les motifs si caractéristiques du pelage de certains mammifères (zèbre, girafe, léopard, etc) peuvent se mettre en place.

On considère ici un modèle simple du développement de tels motifs. L'épiderme est assimilé à une couche unique (2D) de cellules dont chacune est soit pigmentée soit non pigmentée. De plus, chaque cellule aura 4 voisins : au nord, au sud, à l'est et à l'ouest (on ne considèrera pas les voisins aux nord-est, nord-ouest, sud-est et sud-ouest). L'état initial de chaque cellule est aléatoire. Le système évolue de la façon suivante : entre 2 pas de temps considérés, chaque cellule passe dans l'état le plus représenté dans son voisinage immédiat (ses 4 voisins et lui-même).

1. Quelle structure de données peut-on utiliser pour représenter une portion d'épiderme ? On notera *configuration* l'état de cette portion d'épiderme modélisée.
2. Écrire une fonction python `init(m,n)` qui initialise la configuration de l'épiderme composée de m lignes et de n colonnes (chaque cellule a une probabilité identique égale à 0.5 d'être pigmentée). Pour cela, on considérera que la fonction `random.randint(a,b)` du module `random` renvoie un nombre entier compris entre a et b selon une loi uniforme.
3. Écrire une fonction `etatmajoritaire(...)` qui prend en argument une configuration et les indices i et j et qui calcule pour la cellule caractérisée par sa position (i, j), l'état le plus représenté dans son voisinage (5 cellules à regarder dans le cas général). Attention, une cellule n'a pas toujours 4 voisins :
   — lorsque la cellule est dans un angle, il n'y a que 3 cases à regarder (ses 2 voisins et elle-même),
   — lorsque la cellule est sur un bord (mais pas dans un angle), il n'y a que 4 cellules à regarder (ses 3 voisins et elle-même), dans ce cas là, lorsqu'il y a 2 cellules pimentées et deux cellules non-pigmentées, la fonction renverra son propre état,
   — lorsque la cellule est au centre de la grille, il y a 5 cellules à regarder (ses 4 voisins et elle-même).
4. Écrire une fonction `nextconfig` qui prend en argument une configuration et qui renvoie la nouvelle configuration représentant l'état de la partie d'épiderme modélisée à l'étape suivante.
5. Écrire une fonction `simulation` qui prend en argument une configuration initiale et qui simule l'évolution de l'épiderme (cellules pigmentées ou non pigmentées) pendant T pas de temps. Cette fonction renvoie la liste des différentes configurations par lesquelles la simulation est passée.
6. La stabilité est atteinte lorsque aucune cellule ne change d'état entre 2 pas de temps consécutifs. Modifiez la fonction précédente pour que la simulation s'arrête dès que la stabilité est atteinte et qui s'arrête au bout de T pas de temps sinon. On pourra d'abord écrire une fonction qui prend en paramètre deux configurations de l'épiderme modélisé et qui dit si les deux configurations sont égales ou non.