Modélisation et robotique

Exercices 6

1 Révisions du cours

Question A. Que fait ce code :

```
>>> print('3,2,1,0,-1,-2,-3'.split('-'))
>>> print('3,2,1,0,-1,-2,-3'.split(','))
>>> s = 'bonjour toi'
>>> print('m'.join(s.split('t')))
```

Question B. Politesse. Écrivez une fonction `politesse(s)` qui prenne en paramètre une chaine de caractère. La fonction devra insérer dans cette chaine le mot 'Monsieur' à la position du premier espace, et renvoyer le résultat. Exemple d'utilisation :

```
>>> print(politesse('bonjour Meunier'))
bonjour Monsieur Meunier
```

Question C. Pluriel. Vous disposez d'une liste de couples de mots mal orthographiés associés à leur orthographe correcte. Par exemple

```
ORTHO = [('chous', 'choux'), ('pin',
'pain'), ('dinner', 'diner'), ('ve',
'veux')]
```

Écrivez une fonction `corrige` qui prenne en paramètre cette liste et une phrase composée de mots séparés par des espaces, et qui renvoie la phrase avec l'orthographe corrigé. Par exemple :

```
>>> corrige('je ve des chous et du pin
pour le dinner', ORTHO)
'je veux des choux et du pain pour le
diner'
```

Question D. Split avec plusieurs séparateurs possibles. On ne peut pas utiliser la fonction `split` sur une chaine dont les mots ne sont pas séparés par le même caractère.

Par exemple, impossible d'appliquer `split` sur la chaine ~'A, B C'~ pour obtenir la liste `['A','B','C']`. En effet, voici ce que `split` donne sur cette chaine avec différents séparateurs :

```
>>> 'A,B C'.split(',')
['A', 'B C']
>>> 'A,B C'.split(' ')
['A,B', 'C']
>>> 'A,B C'.split(', ')
['A,B C']
```

Écrivez une fonction `split(s,liste_sep)` qui pallie ce problème. Votre fonction permettra de faire cela :

```
>>> split('hello,jo.tu vas bien',
[',','.',' '])
['hello', 'jo', 'tu', 'vas', 'bien']
```

1.0.1 Correction CORRECTION

C'est la question la plus difficile du TD.

Il faut pouvoir enchaîner plusieurs split chacun avec un séparateur différent, par composition de fonction. Le problème est que split est de type $A \to B$ avec $A \neq B$, ce qui empêche la composition. Plus précisément une fois le séparateur choisi, split prend en entrée une chaîne (A est string) et retourne une liste de chaîne (B est list of string). Join fait l'opération inverse en terme de types. Il y a deux possibilités pour répondre au problème.

La première est de définir une fonction split qui travaille avec un seul séparateur mais qui prend en entrée une liste de chaînes sur lesquelles elle applique split puis regroupe les résultats dans une même liste.

```
def flatsplit(phrases, sep):# la bind (flatmap) de la monade ;)
    res = []
    for phrase in phrases:
        res += phrase.split(sep)
    return res

def split(phrase, separateurs):
```

```
    phrases = [phrase] # applique la return de la monade ;)
    for sep in separateurs:
        phrases = flatsplit(phrases, sep)
    return phrases

phrase = 'hello,jo.tu vas bien'
separateurs = [',','.',' ']
print split(phrase, separateurs)
```

L'autre solution consiste en rester le plus possible dans le type chaines à l'aide join. Cela revient ici à recherche/remplacer chaque sorte de séparateur par un seul et même séparateur, puis à applique le split final. Ce spérateur commun doit impérativement être choisi dans la liste (ou ne pas apparaître dans la phrase).

```
def rechremp(phrase, cible, substitut):
    return substitut.join(phrase.split(cible))

def split(phrase, separateurs):
    if separateurs == []:
        return [phrase]
    sep_commun = separateurs[0]
    for s in separateurs[1:]:
        phrase = rechremp(phrase, s, sep_commun)
    return phrase.split(sep_commun)

phrase = 'hello,jo.tu vas bien'
separateurs = [',','.',' ']
print split(phrase, separateurs)
```

2 Ensembles

Question E. Que fait ce code :

```
>>> s = set([1, 2, 3, 3])
>>> print s
>>> s2= {'a', 'b', 'B', 'b', 3, 2}
>>> print(s & s2)
>>> print(s | s2)
>>> print( list(s - s2) )
```

Question F. Qu'affiche ce code ?

```
c = 0
s = set( 'bas haut-haut-bas-gauche-haut-droite-bas'.split('-') )
for i in s:
    print(i)
    c += 1
print(c)
```

Question G. Couples. Écrire une fonction `AllCouples` qui prend un ensemble en paramètre en renvoie l'ensemble de tous les couples. Exemple :

```
>>> AllCouples({'a', 'b', 'c'})
{('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', a'), ('b', 'b'), ('b', 'c'), ('c', 'a'),
('c', 'b'), ('c', 'c')}
```

Question H. Paires. Écrivez une fonction `AllPairs`, qui renvoie tous les couples `(i, j)` tel que `(j, i)` n'est pas renvoyé. Exemple :

```
>>> AllPairs({'a', 'b', 'c'})
{('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', 'b'), ('b', 'c'), ('c', 'c')}
```

2.0.1 Correction CORRECTION

```
def AllPairs(ens):
    res = set([])
    for g in ens:
        for d in ens:
            if (d, g) not in res:
                res.add((g,d))
    return res
print AllPairs({'a', 'b', 'c'})
```

Question I. Paires différentes. Écrivez une fonction `AllDiffPairs`, qui renvoie tous les couples `(i,j)` tels que $i \neq j$ et `(j, i)` n'est pas renvoyé. Exemple :

```
>>> AllDiffPairs({'a', 'b', 'c'})
{('a', 'b'), ('a', 'c'), ('b', 'c')}
```

3 Tableaux Numpy

Question J. Triangle inclus. On suppose qu'un triangle est représenté par une liste de trois listes de coordonnées.

par exemple : `T = [[10, 10], [40, 10], [20, 0]]`

Écrire une fonction `TriangleInterieur(T)` qui utilise la somme de tableaux Numpy pour renvoyer le nouveau triangle dont les sommets sont exactement au milieu des cotés de `T`.

Par exemple :

```
>>> TriangleInterieur( [[10, 10], [40, 20], [20, 0]] )
[[25, 15], [30, 10], [15, 5]]
```

3.0.1 Correction CORRECTION

```
def TriangleInterieur(triangle):
    a = numpy.array(triangle[0])
    b = numpy.array(triangle[1])
    c = numpy.array(triangle[2])
    return [(a + b)/2, (a + c)/2, (b + c)/2]
```

4 Trajectoires

Question K. Aller à. Définir une fonction `aller_a` qui reçoit en entrée une paire de coordonnées `xbut`, `ybut` et déplace l'héroïne de sa postion actuelle à la position `xbut`, `ybut`. Vous utiliserez `avxy()`, `tgo()` et `tdo()` de façon à maintenir dans des variables globales les coordonnées de l'héroïne (variables `xcorr`, `ycorr`) et son orientation sous la forme d'un point cardinal (variable `orientation` accessible par la fonction `quelle$_{\text{orientation}}$()` qui retourne un point cardinal). On suppose qu'il n'y a pas d'obstacle sur la carte mais la fonction pourra être utilisée dans différentes situations et il faut vérifier que votre héroïne progresse bien pour vous assurer que le programme ne boucle pas éternellement.

Question L. Avec des obstacles de taille 1x1. Modifier votre fonction de telle sorte que l'héroïne sache contourner de petits obstacles occupant une seule case et disposés sur la carte de façon non adjacente horizontalement, verticalement ou en diagonale et non collés aux bords de la carte. Vérifiez toujours la progression de l'héroïne pour éviter les boucles infinies.

Question M. Avec des obstacles de taille 1xn ou nx1. Même question mais désormais les obstacles sont des barres horizontales ou verticales, non adjacentes sur la carte.

Question N. Impasse. Finalement, il semblerait que les obstacles, bien que non-adjacents, touchent parfois le bord de la carte.