Tuples et listes

Étienne Lozes – Université Nice Sophia Antipolis

Formation ISN Python 2018

Rappel : somme vectorielle

La somme de deux vecteurs du plan, de coordonnées (x_1, x_2) et (y_1, y_2) est le vecteur de coordonnées $(x_1 + y_1, x_2 + y_2)$. Plus généralement, si $\mathbf{x} = (x_1, \ldots, x_n)$ et $\mathbf{y} = (y_1, \ldots, y_n)$ sont des vecteurs d'un espace de dimension n, leur somme est le vecteur $(x_1 + y_1, \ldots, x_n + y_n)$.

Exercice 1 (Somme vectorielle, $\star$)

1. *Que vaut* `(1,2) + (3,4)` *?*
2. *Écrivez une fonction* `vect_sum2(x,y)` *qui prend en arguments deux vecteurs du plan* $\mathbf{x}$ *et* $\mathbf{y}$*, chacun représenté par un couple, et qui renvoie leur somme vectorielle. Par exemple,* `vect_sum2( (1,2) , (3,4) )` *renvoie* `(4,6)`*.*
3. *Écrivez une fonction* `vect_dim(x)` *qui prend en argument un vecteur* $\mathbf{x}$ *et qui renvoie sa dimension. Par exemple,* `vect_dim( (1,2) )` *renvoie* 2 *et* `vect_dim( (2,3,3,7) )` *renvoie* 4*.*
4. *Écrivez une fonction* `vect_sum(x,y)` *qui prend en argument deux vecteurs de même dimension et qui renvoie leur somme.*

□

Exercice 2 (Lettre et le néant, $\star$)
Écrivez une fonction `find_alpha(s)` *qui prend en argument une chaîne de caractères* `s` *et qui retourne dans un couple l'indice du premier caractère de* s *qui est une lettre ainsi que ce caractère. Par exemple,* `find_alphanum('12ab')` *renvoie* `(2,'a')`*. Si la chaîne* s *ne contient aucune lettre, la fonction renvoie* `None`*. Vous pourrez utiliser la méthode* `isalpha` *sur les chaînes de caractères.* □

Exercice 3 (Moyenne, $\star$)
Écrivez une fonction `moyenne(L)` *prenant en argument une liste non vide* L *d'entiers ou de flottants et renvoyant leur moyenne.* □

Exercice 4 (Jouons avec un dictionnaire orthographique, $\star$)

1. *Récupérez les fichiers* `fr_dic.py` *et* `fr_dic_short.py` *sur la page du cours.*

   http://deptinfo.unice.fr/~elozes/AlgoPython/fr_dic.py
   http://deptinfo.unice.fr/~elozes/AlgoPython/fr_dic_short.py

   puis placez-les dans votre répertoire de travail, ils serviront pour cet exercice et le suivant. Si cela prend trop de temps de récupérer `fr_dic.py`*, prenez seulement* `fr_dic_short.py`*.*
2. *Créez un fichier* `tp6.py` *contenant les deux lignes suivantes :*

```
1 from fr_dic import fr_dic
2 print(fr_dic[:500])
```

   et sauvez-le dans votre répertoire de travail. Exécutez-le : vous devriez voir s'afficher une liste de mots.

3. *Faites afficher les vingt premiers mots de* `fr_dic`*, un par ligne, en donnant leur indice dans* `fr_dic` *et leur longueur.*

```
0 abaissable 10
1 abaissante 10
2 abaissée 8
...
19 abarticulation 14
```

avec `fr_dic.py`

```
0 abat 4
1 abeillage 9
2 abjurer 7
...
19 alpiste 7
```

avec `fr_dic_short.py`

4. *Calculez sans l'afficher une liste L qui contient la longueur des mots du dictionnaire. Affichez* `L[:3]` *: vous devez trouver* `[10,10,8]` *si vous travaillez avec* `fr_dic.py`*, et* `[4,9,7]` *si vous travaillez avec* `fr_dic_short.py`
5. *À l'aide d'une boucle for, calculez* `max_L` *le plus grand élément de* `L`*. Vérifiez en comparant* `max_L` *à* `max(L)`*.*
6. *Affichez le ou les mots les plus longs du dictionnaire. Si vous travaillez avec* `fr_dic`*, il y en a trois, si vous travaillez avec* `fr_dic_short`*, il y en a un seul.*

□

Règles du jeu du pendu

Le jeu du pendu est un jeu entre deux joueurs, appellés ci-dessous le *concepteur* et le *chercheur*. Au début, le concepteur choisit un mot secret m et publie m en ne divulgant que sa première et dernière lettre, les autres étant remplacées par `_`.

A chaque tour de jeu, le chercheur demande une lettre : si cette lettre apparait dans le mot secret, et si elle n'a pas été déjà divulguée, le concepteur divulgue toutes les positions auxquelles elle apparait, sinon le chercheur perd une vie. Si le chercheur trouve toutes les lettres avant de perdre sa dernière vie, il a gagné.

Exemple de partie Une partie où le concepteur est l'ordinateur et le chercheur est l'humain ; l'humain gagne la partie.

```
il te reste 5 vies
a_______t
quelle lettre demandes-tu, humain? e
bien joue!
a_è_e_e_t
quelle lettre demandes-tu, humain? l
raté...
il te reste 4 vies
a_è_e_e_t
quelle lettre demandes-tu, humain? n
bien joue!
a_è_e_ent
quelle lettre demandes-tu, humain? g
raté...
il te reste 3 vies
a_è_e_ent
quelle lettre demandes-tu, humain? t
raté...
il te reste 2 vies
a_è_e_ent
quelle lettre demandes-tu, humain? c
raté...
il te reste 1 vie
a_è_e_ent
quelle lettre demandes-tu, humain? r
```

```
bien joue!
a_ère_ent
quelle lettre demandes-tu, humain? m
bien joue!
amèrement
GAGNÉ! On rejoue?
```

Exercice 5 (Le jeu du pendu, ⋆⋆)

Dans votre répertoire, créez un fichier `pendu.py` *qui commence par importer la variable* `fr_dic` *du module* `fr_dic` *(ou du module* `fr_dict_short`*. – si vous ne pouvez pas travailler avec* `fr_dic.py`*.*

1. *Écrivez une fonction* `mot_au_hasard()` *sans arguments qui renvoie un mot tiré au hasard. Vous devrez utiliser la variable globale* `fr_dic` *et la fonction* `randint` *du module* `random` *(si vous ne vous souvenez plus comment fonctionne* `randint`*, faites* `help(randint)`*. Testez votre fonction en affichant dix mots tirés au hasard*
2. *Écrivez une fonction* `mot_vers_liste(m)` *qui prend en argument une chaîne de caractères* `m` *et qui renvoie une nouvelle liste contenant les caractères divulgués par le* concepteur *en début de partie.*
   Par exemple, `mot_vers_liste('avion')` *renvoie* `['a','_','_','_','n']`
3. *Il est temps de voir comment nous allons utiliser ces fonctions. La fonction principale sera la suivante :*

```
# fonction principale
def nouvelle_partie(vies) :
    m = mot_au_hasard() # le mot a deviner
    L = mot_vers_liste(m) # le mot partiellement decouvert
    print('il te reste' , vies , 'vies')
    while vies > 0 and '_' in L :
        print(''.join(L))
        c = input('quelle lettre demandes-tu, humain? ')
        if met_a_jour(L,m,c) :  # <- la fonction met_a_jour
            print('bien joue!')
        else :
            print('rate...')
            vies = vies - 1
            print('il te reste' , vies , 'vie' + ('s' if vies>1
   else '') )
    if vies == 0 :
        print('PENDU! Le mot secret etait' , m ,'. On rejoue?')
    else :
        print(m)
        print('GAGNE! On rejoue?')
```

   On suppose pour le moment que le dictionnaire ne contient pas de mots accentués.
   Écrivez la fonction `met_a_jour(L,m,c)` *qui prend en arguments une liste* L *de caractères divulgués, une chaîne de caractères* m *de même longueur que* L *et contenant le mot secret, et un caractère* c*. La fonction renvoie* `True` *si* c *peut être divulgué dans* L*, et change le contenu de* L *au passage. Sinon, la fonction ne change pas* L *et renvoie* `False`*.*
   Par exemple, on aura

```
>>> m = 'aviation'
>>> L = ['a','_','_','_','_','_','_','_','_','n']
>>> met_a_jour(L,m,'a')
True
>>> L
```

```
['a','_','_','a','_','_','_','n']
>>> met_a_jour(L,m,'n')
False
>>> L
['a','_','_','a','_','_','_','n']
```

4. *On va maintenant gérer correctement les caractères accentués. La fonction*

```
from unicodedata import normalize
def asciize(s):
    return normalize('NFKD',s).encode('ascii' , 'ignore').decode('
   ascii')
```

permet de renvoyer une chaîne s' obtenue en otant les cédilles et les accents dans s. Modifiez votre fonction `met_a_jour(L,m,c)` *pour qu'elle gère correctement les accents.*

```
>>> m = 'pépètes'
>>> L = ['p','_','_','_','_','_','_','s']
>>> met_a_jour(L,m,'e')
True
>>> L
['p','é','_','è','_','e','s']
```

□

Exercice 6 (Compactage, ⋆⋆)

Dans cet exercice, vous pouvez utiliser la méthode `append`.

1. *Écrivez une fonction* `grouper(L)` *qui prend en argument une liste L et qui renvoie la liste obtenue en remplaçant toute suite d'éléments consécutifs $x, x, x, \ldots, x$ par un seul élément x. Par exemple,* `grouper([4,4,4,2,2,4])` *renvoie* `[4,2,4]`.
2. *Écrivez une fonction* `compacter(L)` *qui prend en argument une liste L et qui renvoie la liste obtenue par groupage en indiquant de plus à l'aide d'un couple la taille de chaque groupe. Par exemple,* `compacter([4,4,4,2,2,4])` *renvoie* `[(3,4),(2,2),(1,4)]`.

□

Tri par insertion

Le tri par insertion est un algorithme de tri qui insère un à un les éléments à trier dans une liste qui contient à la fin le résultat attendu. Par exemple, on aura

```
Etape 0 : tries : []         a trier : [5,8,7,1]
Etape 1 : tries : [5]        a trier : [8,7,1]
Etape 2 : tries : [5,8]      a trier : [7,1]
Etape 3 : tries : [5,7,8]    a trier : [1]
Etape 3 : tries : [1,5,7,8]  a trier : []
```

Exercice 7 (Tri par insertion, ⋆⋆)

1. *Écrivez une fonction* `index_insertion(L,n)` *qui prend en arguments une liste triée d'entiers L et un entier n et qui renvoie l'indice de la position à laquelle insérer n dans L afin de garder la liste triée. Par exemple,* `index_insertion([1,5,6,10],2)` *renvoie 1 car 2 est à l'indice 1 dans la liste* `[1,2,5,6,10]`. *Si n est déjà dans la liste L, on renverra le plus grand indice qui convient. Par exemple,* `index_insertion([1,2,6,10],2)` *renvoie 2.*

2. *Écrivez une fonction* `insertion_sorted(L)` *qui prend en argument une liste d'entiers* L *et qui renvoie une nouvelle liste contenant les entiers de* L *en ordre croissant. Vous pourrez utiliser la méthode* `insert` *sur les listes.*

□

Exercice 8 (Le jeu du pendu suite : l'ordinateur sauve sa peau !, ⋆⋆⋆)
On veut maintenant échanger les rôles, et faire chercher le mot secret à l'ordinateur. Vous allez programmer une intelligence artificielle ! On commence par implémenter une première stratégie utilisant du hasard, puis on étudie une stratégie plus efficace.

1. *Écrivez une fonction* `candidats(deb,fin,l)` *qui prend en argument deux caractères* `deb` *et* `fin` *et un entier* `longueur` *et qui renvoie la liste de tous les mots du dictionnaire qui commencent par* `deb`*, finissent par* `fin`*, et sont de longueur* `l`*. Par exemple,* `candidats('h','s',6)` *renvoie la liste*

```
['habeas', 'hachis', 'haggis', 'hermès', 'herpès', 'hiatus', 'hormis', 'hybris']
```

2. *Écrivez une fonction* `choix_lettre(s,L)` *qui prend en argument un mot partiellement révellé* s *et une liste de mots* L *et, et qui renvoie au hasard une lettre qui apparait dans au moins un mot de* L *à une position où on n'a encore rien révellé. Par exemple, si* `s = 'h___is'` *et* `L = ['hachis', 'haggis', 'hormis', 'hybris']`*, un appel à* `choix_lettre(s,L)` *renverra une lettre au hasard parmi h,a,c,g,o,r,m,y,b,r.*
3. *Écrivez une fonction* `filtre_lettre(c,L)` *qui prend en argument une chaîne de caractères* `s` *et une liste de mots* `L` *et qui renvoie la sous-liste des mots qui ne contiennent pas* `c`*, sauf éventuellement comme première ou dernière lettre. Par exemple,* `filtre_lettre('e',['avion','état','route','été','enquête']) == ['avion','état','route','été']`.
4. *Écrivez une fonction* `est_compatible(s,m)` *qui prend en arguments deux chaînes de caractères* s *et* m *correspondant à un mot incomplet* s *et un mot complet* m*, et qui renvoie* `True` *si* s *peut se compléter en* m*. Par exemple,* `est_compatible('h___is','hachis') == True` *et* `est_compatible('h___is','hermès') == est_compatible('he___s','hermès') == False`.
5. *Écrivez une fonction* `joue_chercheur(vies)` *qui démarre une partie où vous devez faire deviner un mot à l'ordinateur. L'ordinateur donnera des informations sur l'état d'avancement de ses réflexions. On aura par exemple*

```
Quel est ton indice de départ, humain? h____s
J'hésite entre ['habeas', 'hachis', 'haggis', 'hermès', 'herpès', 'hiatus',
   'hormis', 'hybris']
Je demande le a. Nouvel indice? h____s
Il me reste 4 vies.
J'hésite entre ['hermès', 'herpès', 'hormis', 'hybris']
Je demande le i. Nouvel indice? h___is
J'hésite entre ['hormis', 'hybris']
Je demande le y. Nouvel indice? hy__is
J'ai trouvé : hybris.
```

6. *Améliorez la fonction* `choix_lettre(s,L)` *pour accroître les chances de gagner de l'ordinateur.*
   Indication Une possibilité est de s'inspirer de la dichotomie : on peut penser que le choix de la lettre `c` *sera un bon choix si il permet de garder autant de candidats qu'il permet d'en éliminer, autrement dit si le nombre de mots de* L *qui contiennent* `c` *est le plus proche possible de la moitié de la longueur de* L*.*
   Par exemple, si `s = 'h_____s'` *et* `L = ['habeas', 'hachis', 'haggis', 'hermès', 'herpès', 'hiatus', 'hormis', 'hybris']`*, le choix de* `'i'` *permet de garder 5 candidats et d'en éliminer 3, tandis que le choix de* `'y'` *permet de garder 1 candidats et d'en éliminer 7 ; le choix de* `i` *est donc meilleur que le choix de* `y`*. Mais le choix optimal, sur cet exemple, est* `'a'`*, car il permet de garder 4 candidats et d'en éliminer tout autant.*

Vous pouvez cependant réfléchir à d'autres stratégies (maximiser les chances de réduire le nombre de lettres à deviner, stratégie "petit joueur" pour prendre le moins de risque possible, etc), et comparer expérimentalement l'efficacité de ces stratégies.

Vous pouvez enfin vous demander, d'un point de vue mathématique, si il y a une stratégie optimale. La réponse est oui, il y a une stratégie optimale, mais ce n'est pas la même selon que l'on considère l'humain comme un joueur honnête qui tire son mot au hasard avec une loi de probabilité connue de l'ordinateur[1] *et qui ne change plus ce mot après, ou si au contraire l'humain est un tricheur et qu'il n'a pas vraiment fixé de mot secret, et qu'il adapte ses réponses en fonction de celles de l'ordinateur. Ce sont deux modèles de jeux très différents, et une stratégie optimale pour le premier maximise la probabilité de gagner, tandis que pour le deuxième jeu la stratégie optimale consiste à jouer de façon à gagner quelle que soit la stratégie pleine de fourberies de l'humain, lorsque c'est possible.*

□

1. par exemple uniformément, ou selon la fréquence d'apparition des mots dans un grand corpus de textes