0.1 Cours

Comme les chainse de caractères, les listes sont des **séquences, c'est à dire des collections ordonnées d'objets**. Ce sont donc des conteneurs dont les élements sont toujours dans le même ordre et on peut y accéder grâce à leur *index*.

0.1.1 Propriétés communes aux séquences (Rappel)

Les séquences ont des propriétés en commun. Si `seq` est une séquence, alors :

1. `seq[k]` désigne l'élément d'indice k de cette séquence (la numérotation commençant à 0) , l'accès à cet élément se fait en coût constant; il n'est pas plus coûteux d'accéder à `seq[1]` ou `seq [10000]` en Python.
2. on peut effectuer des coupes à l'aide de la technique du tranchage (slicing) ;
3. la fonction `len` donne la longueur de la séquence, c'est à dire son nombre d'éléments ;
4. la concaténation et la duplication se notent respectivement `+` et `*` ;

Cette dernière permet de créer des listes vides rapidement:

Les fonctions `str`, `list`, `tuple` permettent de convertir un type de séquence en un autre ; par exemple :

```
>>> list('Commodore')
['C', 'o', 'm', 'm', 'o', 'd', 'o', 'r', 'e']
```

0.1.2 Listes

Leur **type** est `list`. L'appellation *liste* est trompeuse car les listes *Python* ne correspond pas au type de donnée liste usuellement utilisé en informatique : elles ont des propriétés du type de donnée **tableau** et du type de donnée **liste** usuels. Les listes sont des séquences **mutables** : on peut leur ajouter des éléments, les modifier.

Exemple 1

```
>>> L1 = [3, 1, 4, 1, 5, 9, 6 , 5 , 4]
>>> L1[4]
5
>>> L2 = ['linux', 'android', 'unix', 'windows' , 'macos']
>>> L2[0]
'linux'
>>> L2.pop(3)
'windows'
>>> L2
['linux', 'android', 'unix', 'macos']
>>> L = L1+L2
>>> L
[3, 1, 4, 1, 5, 9, 6, 5, 4, 'linux', 'android', 'unix', 'macos']
>>> L3 = [[1, 2, 3, 4], [0, 0, 0, 0], [-1, 7, 8, 2]]
>>> L3[2][1]
7
>>> L3[0][0]=-5
>>> L3
[[-5, 2, 3, 4], [0, 0, 0, 0], [-1, 7, 8, 2]]
>>> len(L3)
3
```

```
>>> len (L3[1])
4
>>> L1.append(10)
>>> L1
[3, 1, 4, 1, 5, 9, 6, 5, 4, 10]
>>> L1=[]
>>> L1
[]
```

Notez la différence entre la modification en place pour les méthodes `pop`, `append` et la concaténation qui nécessite d'utiliser une nouvelle affectation. Cette dernière a été employée avec les chaines de caractère qui sont non modifiables.

0.2 Exercices : Listes de types simples

Exercice 1 Ecrivez une fonction `liste_impair` qui à partir d'une liste d'entiers `L` renvoie la liste des entiers impairs de L.

```
 >>> liste_impair([3, 1, 4, 7,  6, 2])
[3, 1]
```

Exercice 2 Ecrivez une fonction `somme` prenant deux listes L1 et L2 de même taille en paramètre qui renvoie une liste où chaque élément est la somme des éléments de L1 et L2 de même indice.

Exemple:

```
 >>> somme([3, 1, 4], [1,  6, 1])
[4, 7, 5]
```

Exercice 3 : Conversion décimal vers binaire Ecrivez une fonction `dec2bin` renvoyant une liste d'entier représentant le nombre `nbre` passé en paramètre en base 2.

Exemple:

```
 >>> dec2bin(12)
 [1, 1, 0, 0]
```

Exercice 4 : Liste de prénoms Soit `L` une liste de prénoms. Écrire une fonction `prenom_court` ne renvoyant que la liste des prénoms de moins de 5 lettres (strictement) de la liste `L`.

Exemple:

```
>>> L = ['Rick', 'Gabrielle', 'Morty', 'Emma', 'Ulrich', 'Eva']
>>> prenom_court(['Rick', 'Gabrielle', 'Morty', 'Emma', 'Ulrich', 'Eva'])
['Rick',  'Emma', 'Eva']
```

Exerice 5 : Marche aléatoire On effectue des sauts aléatoires sur une droite graduée, la puce se trouve en 0 au début de la partie et on lance une pièce:

Si on obtient `pile`, la puce saute d'une unité à droite, et son abscisse devient donc +1.

Sinon, elle saute à gauche et son abscisse devient -1. Une partie est composée de plusieurs sauts, on voudrait estimer les positions possibles pour la puce.

1. Quels sont les résultats possibles pour la position de la puce si une partie se joue en trois sauts ?

2. Ecrire une fonction `marche_alea` prenant en paramètre `nb_sauts` et renvoyant le résultat de la partie.
3. Ecrire une fonction `partie` prenant en paramètre le nombre de parties jouées et renvoyant les résultats obtenus:
   (a) dans une liste
   (b) dans un dictionnaire.

Exercice 6 : Opérations logiques

1. Programmer une fonction `non` qui correspond à la négation pour une liste donnée.

```
>>> non([1, 1, 0, 1])
[0, 0, 1, 0]
```

2. Mêmes questions avec une fonction `ou_eg` et une fonction `et_eg` prenant en paramètre des listes de longueurs égales.
3. Ecrire une fonction `ou` puis une fonction `et` qui prend en paramètre deux listes de longueurs éventuellement différentes. Il faudra écrire une fonction intermédiaire `ajoute_zeros` pour gérer ce cas.

Exercice 7 : Parcours séquentiel d'une liste Ces algorithmes font partie des algorithmes type BAC. Une liste L est donnée:

1. Écrire une fonction `moyenne_liste` renvoyant la moyenne des valeurs de la liste `L` passée en paramètre. Quelle est la complexité de cet algorithme ?
2. Écrire une fonction `maxi_liste` renvoyant le maximum des valeurs de la liste `L`. Quelle est la complexité de cet algorithme ?
3. Écrire une fonction `positions_minimum` qui prend en paramètre une liste de nombres et renvoie une liste contenant les positions de la valeur minimale de ces nombres (la liste pourra ne contenir qu'un seul élément si le minimum n'apparaît qu'une fois).
4. Écrire une fonction `recherche` renvoyant `True` si la valeur `val` est présente dans la liste `L` et `False` sinon. Quelle est la complexité de cet algorithme ?

Exercice 8

1. Écrire une fonction `tableau_aleatoire` prenant en paramètre `n`, `a` et `b` qui renvoie une liste de taille n contenant des entiers tirés au hasard entre a et b inclus.
2. Écrire une fonction `loto` simulant le tirage du loto: 5 boules parmi les numéros 1 à 49 et un complémentaire entre 1 et 10. Créer une liste de 49 boules, elle sera passée en paramètre à la fonction `loto`.
3. Écrire une fonction `echange` prenant en paramètre `tab`, `i`, `j` et qui échange dans le tableau `tab` les éléments d'indice `i` et `j`. Écrivez deux versions: une avec et sans **effet de bord**.
4. Reprendre la fonction `maximum` pour qu'elle renvoie un *tuple* constitué du maximum et de tous les indices dans la liste. Gérer avec une assertion le cas où la liste est vide (programmation défensive).

0.3 Complément sur les listes

0.3.1 Liste en compréhension

Un ensemble peut se définir:

- en donnant ses éléments E = {2, 4, 6, 8, 10},
- en se servant de certaines de ses propriétés : E = $\{x \in \mathbb{N} \mid x \leqslant 10$ et x est pair$\}$. Cette dernière permet de définir une liste en compréhension.

Voici une liste des nombres pairs, définie à partir de la liste des 10 premiers entiers non nuls en compréhension:

```
>>> L = [x for  x in range (1, 11) if x%2 == 0]
>>> L
[2, 4, 6, 8, 10]
```

En voici la syntaxe:
`L = [item for item in liste if condition]`.

1. Ecrire une fonction `multiplier`(liste, k) prenant en paramètre une liste et un entier. Elle retourne une nouvelle liste dont chaque élément est celui de la liste passée en paramètre multiplié par cet entier.
2. Ecrire une fonction `puissance`(liste, k) prenant en paramètre une liste et un entier. Elle retourne une nouvelle liste dont chaque élément est celui de la liste passée en paramètre élevé à la puissance de cet entier.
3. Ecrire une fonction `non_zero`(liste). Elle retourne une nouvelle liste ne contenant aucun zéro.
4. Ecrire une fonction `liste_alea` prenant un paramètre `long` renvoyant une liste de taille `long` formée d'entiers pris aléatoirement dans $[\![0, 100]\!]$.
   (a) Avec une boucle.
   (b) Avec une compréhension de liste.

   Exemple

   ```
    >>> liste_alea(5)
   [44, 56, 80, 13, 62]
   ```

5. Ecrire une fonction `liste_pair` prenant un paramètre `liste` renvoyant la liste des entiers pairs contenu dans `liste` en utilisant la compréhension de liste.

   Exemple:

   ```
   >>> liste_pair([1, 12, 5, 4, 8, 123])
   [12, 4, 8]
   ```

0.3.2 Liste de listes

Une liste peut contenir d'autres listes:

```
>>> liste = [["Rick", "Morty", "Bender"], [100, 102], [True, 1==2]]
```

Les tableaux contiennent des listes d'entiers.

```
>>> tab = [[3, 1, 4], [2, 7, 12] , [100, 102, 101]]
>>> tab[0]
[3, 1, 4]
>>> tab[1][2]
12
```

On peut les écrire par compréhension:

```
>>> liste = [[x, y] for x in range(3) for y in range(3)]
```

Les variables suivantes sont des listes de listes:

- tableau1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
- tableau2 = [[2, 14, 5], [3, 5, 7], [15, 19, 4], [8, 6, 5]]

`tableau[i]` renvoie la sous-liste de rang i, alors que `tableau[i][j]` renvoie l'entier situé au rang j dans la sous-liste de rang i. Par exemple :

```
>>> tableau2 = [ [2,14,5], [3,5,7], [15,19,4], [8,6,5] ]
>>> tableau2[0]
[2, 14, 5]
>>> tableau2[2][1]
19
```

Les listes de liste peuvent être vues comme des matrices à i lignes et j colonnes. Le tableau1 est représenté par la matrice 1 et d'une façon plus générale avec les index i pour les lignes et j pour les colonnes par la matrice M.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Matrice 1

$$\begin{pmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{pmatrix}$$

Matrice M

Ce point de vue est par exemple très pratique pour représenter des images.

Figure 1: [Illustration David Revoy CC-BY]

0.3.3 Exercices

Exercice 1

1. Ecrire une fonction `somme_diagonale` prenant un tableau `tab` de taille $n \times n$ en paramètre et renvoyant la somme des éléments de sa première diagonale.
2. Ecrire une fonction `somme_anti_diag` prenant un tableau `tab` $n \times n$ en paramètre et renvoyant la somme des éléments de son anti-diagonale.
3. Ecrire une fonction `somme` prenant un tableau `tab` $n \times n$ en paramètre et renvoyant la somme de ses éléments.
4. Ecrire une fonction `initialisation` renvoyant une liste de liste de taille $m \times n$ où tous les éléments valent zéro:
   (a) avec un boucle.
   (b) par compréhension.

Exercice 3 : Loi de De Morgan

1. Ecrire une fonction `tous_les_binaires` prenant en paramètre un entier `taille` et renvoyant la liste de tous les nombres binaire de 0 à `taille-1`, écrits sous forme de liste. Ajouter des zéros si besoin.

```
>>>tous_les_binaires(2)
[[0, 0], [0, 1], [1, 0], [1, 1]]
```

2. Version récursive (Terminale NSI):
   (a) Si `taille`=1: renvoyer `[[0], [1]]`.
   (b) Si `taille`$\geqslant$2:
   - ajouter 0 en début de la liste renvoyée par `tous_les_binaires(taille-1)`
   - ajouter 1 en début de la liste renvoyée par `tous_les_binaires(taille-1)`
3. Vérifier les lois de De Morgan pour `taille`=8 bits.

Exercice 4 : Vecteurs creux. *Cet exercice utilise le type de données dictionnaire (`dict`).*

Une des techniques pour compresser un matrice ou un tableau *creux* (*sparse matrices*), c'est à dire avec une forte proportion de zéro est de n'indiquer que la liste des valeurs non nulles avec leur index.

1. Ecrire une fonction qui prend un tableau et le représente comme un vecteur creux.

   Exemple

```
>>> v = [0, 0, 4, 0, 0, 0, 5, 6]
>>> vecteur_creux(tab)
[(2, 4), ( 6, 5), (7, 6)]
```

2. Coder une fonction semblable pour une liste de liste mais en renvoyant un dictionnaire.

   Exemple

```
>>> v = [[0, 0], [4, 0], [0, 0], [5, 6]]
>>> vecteur_creux2(tab)
{(1, 0):4, (3, 0):5, (3, 1):6}
```

Exercice 5 : Run length encoding (RLE) Une des techniques pour compresser un matrice ou un tableau est le codage par plages. il s'agit d'indiquer pour chaque suite de valeurs identiques le nombre de fois (au moins une) que cette valeur est répétée. Le tableau de paires obtenu est généralement pus court que le tableau initial. Ecrire une Fonction `rle` prenant un paramètre `tab` renvoyant le codage par plage d'un vecteur.

Exemple

```
>>>rle(['aa', 'aa', 'b', 'c', 'c', 'c', 'aa'])
[(2, 'aa'), (1, 'b'), (3, 'c'), (1, 'aa')]
```

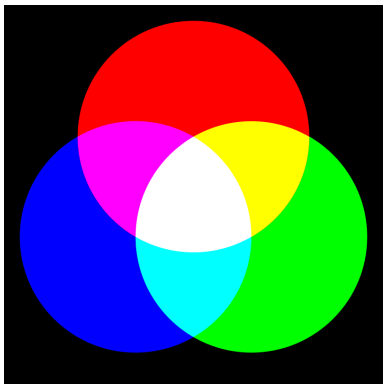
Cet algorithme est surtout utilisé pour compresser des images en regroupant les pixels voisins de valeurs similaires. Il est notamment utilisé avec le format JPEG en combinaison avec d'autres techniques.

0.4 TP : images et couleur RGB

0.4.1 Partie A : Création d'une image couleur pixel par pixel

On rappelle qu'une image couleur est composée de pixels comportant trois informations : les intensités de rouge, de vert et de bleu. Ces intensités sont codées par des valeurs entières comprises entre 0 et 255.

Le schéma ci-contre donne le principe du système additif de la couleur qui permet de visualiser comment la combinaison de plusieurs sources colorées lumineuses (du rouge, du vert et du bleu) produit les couleurs d'autres couleurs (ici le jaune, le magenta et le cyan).



Compléter le tableau ci-dessous avec le triplet de valeurs correspondant à la couleur.

Couleur	Noir	Rouge	Vert	Bleu	Cyan	Jaune	Magenta	Blanc
Triplet	$(0, 0, 0)$	$(255, 0, 0)$					$(255, 0, 255)$	

Figure 2: Compléter le tableau

1. On donne le script Python ci-dessous. Colorier le quadrillage représentant les pixels de l'image construite de la couleur qui convient.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0							
1							
2							
3							
4							

```
def dessine_image() :
  img = [[0 for x in range(7)] for y in range(5)]
  for x in range(7) :
    img[0][x] = [255, 0, 0]
  for y in range(5) :
    img[y][6] = [0, 0, 255]
  for x in range(6) :
    for y in range(1, 5) :
      if x == y :
        img[y][x] = [0, 255, 0]
      else :
        img[y][x] = [255, 255, 255]
```

2. Quelles sont les couleurs utilisées dans cette image ?
3. Pourquoi le dernier pixel de la première ligne est-il bleu et pas rouge ?
4. Où sont situés les pixels verts ?
5. Comment modifier le code fourni afin que la deuxième ligne de l'image soit cyan, que la troisième colonne soit magenta et que le pixel à l'intersection de cette deuxième ligne et troisième colonne soit noir ?

0.4.2 Partie B : Filtres couleurs

On donne la fonction `FiltreBleu`, qui, à partir d'une image couleur img codée sous la forme d'une liste de listes, produit une image bleue :

```
def filtre_bleu(img) :
  ''' Produit une image bleue à partir d'une image en
  couleur donnée sous forme matricielle (liste de lignes)
  en mettant les intensités de rouge et de vert à 0.
  '''
  hauteur = len(img)
  largeur = len(img[0])
  new = [[0 for x in range(largeur)] for y in range(hauteur)]
  for y in range(hauteur) :
    for x in range(largeur) :
      new[y][x] = [0, 0, img[y][x][2]]
  return new
```

1. Comment modifier la fonction pour produire un filtre rouge ?
2. Même question pour un filtre vert.
3. Modifier le script afin d'échanger les quantités de rouge et de bleu.

Figure 3: Le projet Voronoï utilise des listes et le système de couleur RVB

0.5 Sources

- Liste et forum NSI.
- TP RGB : *M. Boehm, C. Poulmaire, P. Remy ⋆ Académie de Versailles.*