Première Représentation des données	Compléments sur les tableaux indexés et dictionnaires	

Objectifs de la séance :

- *Fabriquer un tableau par compréhension*
- *Parcourir un tableau indexé à double entrées (matrice)*
- *Continuer le travail sur l'écriture modulaire et la spécification des fonctions*
- *Écrire des tests unitaires*
- *Découvrir les dictionnaires*

Matériel et Logiciel nécessaire :PC

- *un environnement de programmation Python (Annaconda et Spyder)*

Durée estimée : 2 h

A - Fabriquer des listes par compréhension

Les compréhensions de listes fournissent un moyen de construire des listes de manière très concise. Une application classique est la construction de nouvelles listes où chaque élément est le résultat d'une opération appliquée à chaque élément d'une autre séquence.

Par exemple, si l'on veut une liste contenant la suite des carrés des entiers inférieurs à 20 :

```
# Méthode classique
carres = []
for i in range(20):
    carres.append(i**2)
```

```
# Méthode par compréhension
carres = [i**2 for i in range(20)]
```

On peut également ajouter des conditions, si par exemple nous ne voulons que les carrés pairs :

```
# Carrés pairs
carres = [i**2 for i in range(20) if i%2 == 0]
```

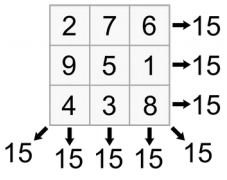
Exercice 1

1. Écrire le code permettant de générer une liste contenant 15 fois le même objet (le nombre 0, la lettre 'a').
2. Écrire le code générant la liste des lettres contenus dans une chaîne de caractère, par exemple partant de la chaîne `'Python'` on veut générer la liste `['P', 'y', 't', 'h', 'o', 'n']`
3. Que fait le code suivant ?

```
[chr(i) for i in range(65,123) if i not in [k for k in range(91, 97)]]
```

B - Carré magique

Le carré magique est une bonne illustration simple de ce qu'est une matrice et comment l'implémenter à l'aide de listes en Python. Le seul carré magique « normal » d'ordre 3 est le carré de Luo Shu (tous les autres sont obtenus par réflexion ou rotation). Voici, une implémentation en Python et comment accéder à ses éléments :




```
carre_magique = [[2, 7, 6], [9, 5, 1], [4, 3, 8]]
# Pour accéder l'élément de la ligne 2, colonne 3
carre_magique[1][2]
```

Exercice 2

1. Écrire une fonction `somme_ligne()` prenant pour paramètres un carré magique et un numéro de ligne et retournant la somme de valeurs de la ligne.
2. Même travail pour la somme des colonnes et des deux diagonales (il faudra trois autres fonctions).
3. Écrire une fonction retournant le nombre magique (somme commune) d'un carré magique après validation à l'aide des fonctions précédentes et `-1` si ne carré n'est pas magique.
4. Que dire du carré suivant ?

```
[[11, 24, 7, 20,3], [4, 12, 25, 8, 16], [17, 5, 13, 21, 9], [10, 18, 1, 14, 22], [23, 6, 19, 2, 15]]
```

5. Expliquer le script suivant :

```
from random import choice

carre = [[0 for i in range(n)] for j in range(n)]
valeurs = [i for i in range(1, n**2+1)]
for i in range(n):
    for j in range(n):
        nombre = choice(valeurs)
        carre[i][j] = nombre
        valeurs.remove(nombre)
```

C - Une nouvelle structure de donnée : les dictionnaires

Un autre type de donnée très utile, natif dans Python, est le *dictionnaire.* Ces dictionnaires sont parfois présents dans d'autres langages sous le nom de "mémoires associatives" ou de "tableaux associatifs".

À la différence des séquences, qui sont indexées par des nombres, les dictionnaires sont indexés par des *clés*, qui peuvent être de n'importe quel type immuable ; les chaînes de caractères et les nombres peuvent toujours être des clés. Des tuples peuvent être utilisés comme clés s'ils ne contiennent que des chaînes, des nombres ou des tuples.

Le plus simple est de considérer les dictionnaires comme des ensembles de paires *clé: valeur*, les clés devant être uniques (au sein d'un dictionnaire). Une paire d'accolades crée un dictionnaire vide : `{}`. Placer une liste de paires `clé:valeur` séparées par des virgules à l'intérieur des accolades ajoute les valeurs correspondantes au dictionnaire ; c'est également de cette façon que les dictionnaires sont affichés.

Quelques exemples dans une console Python :

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

D - Sudoku

Le **sudoku** est un jeu en forme de grille défini en 1979 par l'Américain Howard Garns.

Le but du jeu est de remplir la grille avec une série de chiffres (ou de lettres ou de symboles) tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans un même « bloc » . La plupart du temps, les symboles sont des chiffres allant de 1 à 9, les régions étant alors des carrés de 3 × 3.

Quelques symboles sont déjà disposés dans la grille, ce qui autorise une résolution progressive du problème complet.

				2		7		8
5	2						6	
1		4			9		5	
			1	7	8			6
				4		8		
			9	5	6			
6					5	2		7
	9				3		8	
	3	2	7		4	5	9	

Voici une implémentation de cette grille à l'aide d'une liste :

```
grille = [[0, 0, 0, 0, 2, 0, 7, 0, 8], [5, 2, 0, 0, 0, 0, 0, 6, 0], [1, 0, 4, 0, ...
```

Exercice 3 (mini-projet)

L'objectif est de résoudre par « induction » une grille de sudoku « facile ».

1. Poursuivre l'implémentation de cette grille par une liste.
2. Écrire la fonction liste_cases_vide() prenant pour paramètre un Sudoku et retournant une liste de tuples des « coordonnées » des cases vides de la grille.
   La liste retournée pour cette grille commence ainsi :

```
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 5), (0, 7), (1, 2) ...
```

3. Écrire les fonctions `est_dans_ligne()`, `est_dans_colonne()` et `est_dans_bloc()` prenant pour paramètres une grille de Sudoku, un entier correspondant au numéro de ligne, de colonne ou de bloc, ainsi qu'une `valeur` et retournant le booléen True si cette valeur est dans la zone choisie.
   Écrire quelques tests unitaires pour chacune de ces fonctions.
4. Écrire la fonction `liste_candidats_case()` prenant pour paramètres une grille et un tuple correspondant aux coordonnées d'une case vide et retournant la liste des valeurs possibles ou « candidats » pour cette case. Par exemples :
   a) `liste_candidats_case(grille, (4, 0))` retournera `[2, 3, 7, 9]` ;
   b) `liste_candidats_case(grille, (4, 5))` retournera `[2]`.
5. Écrire la fonction `liste_candidats()` prenant pour paramètre une grille et retournant le dictionnaire des candidats de l'ensemble des cases vides la grille.
6. En déduire une méthode pour remplir la grille par « induction », c'est-à-dire en utilisant ce dictionnaire des candidats.

Quelques grilles supplémentaires :

	8					9	2	
4	1				8		7	
	5					3	8	
	9		4	8	1	2	6	
7		2	3			1		
				2				
5		6			2	8	9	
		8	7				1	6
	3					7	5	

1						3		
	2			4	3	5	7	
	3				8	9		2
	5	6	8		1		3	4
7				6	5	2		
		1						
			6	2	4			3
				3				
		9						6

	2	9		6		8	7	5
4				7		6		3
3					5	1		4
					1		8	
8			2			9	5	
5	9				8		6	1
		6	3	5	7	2		
	1							6
	8		6		2		4	