UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

# Info. - TD : Feuille 5

Rappel : vous savez maintenant utiliser les tableaux numpy à n'importe quel nombre de dimensions avec : la création de tableaux, la notion de shape et la fonction reshape (avec valeur -1 possible), les indices, les tranches, les tranches avec pas (potentiellement négatifs), les indices négatifs, les fonctions d'aggrégation, le broadcasting, les indices par tableau de booléens.

En TP vous avez aussi pratiqué les tracés avec matplotlib.

Ce TD vise à utiliser ces compétences pour résoudre les problèmes donnés. Il vous est souvent demandé des *fonctions*. Si vous ne savez plus définir une fonction, reportez vous au cours ou, si vous n'y avez pas accès, écrivez un programme sans fonction.

### Exercice 1 – Carrés magiques

Un **carré magique** de taille $n$, est un tableau de nombres entiers de taille $n \times n$ et dont la somme de chaque ligne est égale, mais aussi égale à la somme de chaque colonne et de chaque diagonale.

Un carré magique **normal** est un carré magique contenant les entiers de 1 à $n^2$ (une fois chacun).

Un carré est **presque magique** si il vérifie la propriété du carré magique pour chaque ligne, chaque colonne mais pas forcément pour les diagonales.

**Q1)** Écrire une fonction `estPresqueCarréMagique(c)` qui, à partir d'un tableau numpy, renvoie un booléen (True si c est presque magique, False sinon). La fonction doit renvoyer la chaine de caractères `Erreur` si le tableau c n'est pas un tableau à 2 dimensions ou si ses dimensions ne sont pas égales (tableau pas carré).

**Q2)** Écrire une fonction `estNormal(t)` qui, à partir d'un tableau numpy, renvoie True si le tableau t contient exactement une fois chaque nombre de 1 à $N$, où $N$ est le nombre d'éléments dans t.

**Q3)** Écrire une fonction `estCarréMagiqueNormal(c)` qui renvoie True si le tableau c est un carré magique et qu'il est normal (cette fonction *doit* utiliser les autres fonctions).

**Q4)** Écrire une fonction `estCarréMagique(c)`, qui fait comme `estPresqueCarréMagique(c)` mais vérifie aussi les diagonales. **Aide 1** : on pourra utiliser `np.eye(n)` qui crée une matrice identité (tableau $n \times n$ avec que des 0 sauf sur la diagonales (où il y a des 1). **Aide 2** : un tableau numpy (de float ou autre) peut être converti en un autre type avec `a.astype(nouveau_type)`, par exemple `a.astype(bool)` pour changer un tableau en tableau de booléens. **Aide 3** : les indices négatifs peuvent servir à « inverser » un tableau selon un axe.

### Exercice 2 – Artisant Mineur

On suppose un monde en 3d, découpés en blocs cubiques de 1m×1m×1m. L'état du monde est représenté dans un tableau numpy à 3 dimensions. Chaque case du tableau contiendra un entier représentant le type de bloc (0 pour vide, 1 pour terre, 2 pour pierre, 3 pour bois, 4 pour fer, 5 pour eau, 6 pour sable, 7 pour lave, 8 pour diamant, …)

On suppose que les deux premiers axes sont des coordonnées géographiques et que le troisième axe est l'altitude (l'altitude augmente avec les indices). Une position dans le monde est alors composée de 3 indices (coordonnées x, y, z) que l'on stockera dans une liste à 3 éléments ou un n-uplet à 3 éléments. On **suppose** aussi l'existence d'une fonction qui génère un monde de la taille demandée. Par exemple, le code suivant crée un monde et affiche les dimensions du tableau correspondant (ici `(1000, 3000, 800)`).

```
1 monde = creerMonde(1000, 3000, 800)
2 print(w.shape)
```

**Q5)** Quelle expression permet d'extraire la case aux coordonnées 15, 20 et à l'altitude 550 ?
Quelle expression permet d'extraire la colonne de blocs aux coordonnées 15, 20 ?
Quelle expression permet d'extraire la tranche du monde à l'altitude 550 ?

**Q6)** Écrire une fonction `minerDessous(w, p)` qui reçoit un monde et une position en paramètres et remplace la case au dessous de la position p par du vide (valeur 0). La fonction doit renvoyer la valeur qu'il y avait dans la case avant qu'elle n'ai été remplacée par 0.

**Q7)** Écrire une fonction `creuserColonne(w, p)` qui creuse sous la position p jusqu'au fond (altitude 0). La fonction doit renvoyer l'ensemble des valeurs (dans un tableau) remplacées par des 0. **Aide** : pensez à faire une copie avec `np.copy`.

**Q8)** Écrire une fonction `creuserVersLeCiel(w, p)` qui creuse depuis la position p jusqu'au ciel (altitude maximale). La fonction doit à nouveau renvoyer l'ensemble des valeurs remplacées.

**Q9)** Écrire une fonction `tracerRepartitionAltitude(w, t, titre)` qui reçoit un monde, un numéro de type de bloc (par exemple 8 pour diamant) et un titre (à utiliser pour le titre du graphique). Cette fonction doit tracer une courbe avec l'axe des x correspondant à l'altitude et l'axe des y correspondant au nombre total de blocs de type `t` à cette altitude.

**Q10)** Écrire une fonction `tracerPlusieursRepartitionAltitude(w, types, titre)` qui reçoit cette fois une liste de numéros de type de bloc. Cette fonction doit tracer, sur un même graphique, les courbes correspondants à chaque type de la liste, avec en légende `Type n` où n est le numéro du type.

**Q11)** Écrire une fonction `creuserAttaqueCirclaire(w, p, r)` qui creuse horizontalement autour de la position p du joueur, avec un rayon de r. Formulé autrement, tous les blocs à la même altitude que p et à une distance inférieur à r mètres doivent être transformés en vide. Ici encore, renvoyer un tableau avec les valeurs creusées. **Aide 1** : on extraira d'abord la tranche horizontale sur laquelle se trouve le joueur. **Aide 2** : en ayant défini correctement les variables X, Y, et en ayant bien compris le broadcasting, l'expression `(X-p[0])**2 + (Y-p[1])**2 < r**2` nous donne un tableau de booléens qui représente le disque autour du joueur. **Aide 3** : n'oubliez pas l'indiçage par tableau de booléen.

**Q12)** Avec la même idée de broadcasting mais en 3d, écrire une fonction `creuserSphere(w, p, r)` qui creuse sur toute une sphère rayon r autours de la position p.

**Q13)** En adaptant ce broadcasting, écrire une fonction `creuserPuits(w, p, r)` qui creuse un puits (cylindre) de rayon r en dessous et autours de la position p.

**Q14)** Écrire une fonction `tracerBarresRepartitionAltitude` qui fonctionne comme `tracerRepartitionAltitude` mais trace un diagramme y barres, horizontal (c'est à dire que les barres seront horizontales).

### Exercice 3 – Au sujet du sudoku

**Q15)** Écrire le code qui initialise un tableau de sudoku contenant uniquement des valeurs de 0 (pour indéfini).

**Q16)** Écrire une variante qui initialise un tableau de sudoku contenant des entiers tirés au hasards entre 1 et 9.

**Q17)** Écrire une fonction `verifie(doku)` qui reçoit un tableau de sudoku en paramètre et renvoie vrai (True) si et seulement si la grille de sudoku est correcte (respecte les contraintes). On supposera qu'il existe une fonction `est_ok(t)` qui reçoit un tableau de 9 valeurs en paramètre et qui renvoie vrai si et seulement si ces neuf valeurs sont les nombres de 1 à 9 (dans un ordre quelconque).

**Q18)** Écrire maintenant la fonction `est_ok`. Il y a plusieurs approches possibles, trouvez en une qui marche dans tous les cas.

**Q19)** Écrire une fonction `acharnement()` qui renvoie une grille de sudoku complète valide en tirant des grilles aux hasards jusqu'à en trouver une qui est correcte.

**Q20)** Dans `acharnement()`, comptez et affichez le nombre d'essais réalisés avant de trouver une grille.

**Q21)** Écrire une fonction `verifie_partielle(doku)` qui vérifie qu'une grille de sudoku respecte pour l'instant toutes les containtes. La grille n'est que partiellement remplie et peut donc contenir des 0. On pourra supposer l'existence d'une fonction `est_ok_partielle(t)` sur le modèle de `est_ok(t)` mais qui accèpte les ensembles partiels (avec des 0).

**Q22)** Écrire un jeu de sudoku, qui

- initilise une grille de sudoku avec quelques valeurs (de votre choix), et l'affiche au joueur
- demande au joueur de rentrer 3 valeurs (par exemple `2 1 8`) pour remplir une case vide (mettre 8 dans la case d'indice 2, 1)
- remplie cette valeur et réaffiche la grille,
- vérifie qu'il n'y a pas d'incohérence (et affiche un message s'il y a une incohérence),
- redemande à l'utilisateur de remplir une nouvelle valeur, etc.,
- félicite le joueur si la grille est bien remplie.