

# Mini devoir surveillé

On demande d'écrire en langage Python les fonctions listées ci-dessous. Il n'est pas nécessaire d'indiquer les spécifications lorsque celles-ci sont déjà données dans l'énoncé, ni d'indiquer le rôle d'une variable dans les deux cas suivants :

- Compteur de boucle ;
- Variable qui contiendra le résultat à l'issue du programme.

Dans tous les autres cas, les spécifications des fonctions devront être indiquées entre triple guillemets, et le rôle des variables précisé par un commentaire.

Les deux premières parties contiennent des questions de cours ou des applications directes et suffisent à obtenir une note correcte. La partie 3 est nettement plus difficile. Si vous n'avez pas fini dans le temps imparti, vous pouvez avancer dessus après le DS et rendre la suite dans mon casier d'ici la fin de la semaine auquel cas je mettrai un bonus.

## 1 Cours

1. Une fonction `triplet_pyth` prenant un entier  $n \in \llbracket 5, \infty \llbracket$  et renvoyant un triplet  $(a, b, c) \in \llbracket 1, n \rrbracket^3$  tel que  $a^2 + b^2 = c^2$ .

On suppose que la fonction `randint` permettant de tirer un nombre entier au hasard a été chargée via **from** `numpy.random` **import** `randint`. Elle s'utilise ainsi : pour tout  $(a, b) \in \mathbb{N}^2$ , `randint(a, b)` renvoie un entier de l'intervalle  $\llbracket a, b \llbracket$ .

2. Une fonction factorielle prenant en entrée  $n \in \mathbb{N}$  et renvoyant  $n!$ .
3. Un prédicat `appartient` prenant un élément  $x$  et un tableau  $t$  et indiquant si  $x \in t$ .
4. Une fonction `tableauALenvers` prenant un tableau  $t$  et renvoyant un tableau contenant les mêmes éléments mais dans l'ordre inverse.

## 2 Exercices du TD

1. Une fonction `somme_tab` prenant un tableau d'entiers et renvoyant la somme de ses éléments.
2. Une fonction `max_tab` calculant le maximum d'un tableau supposé non vide, puis une fonction `arg_max` renvoyant l'indice d'une case contenant le maximum d'un tableau.
3. Une fonction `approx_e` prenant un entier  $n \in \mathbb{N}$  et renvoyant  $\sum_{i=0}^{n-1} \frac{1}{i!}$ . Une version efficace rapportera plus de points. Préciser, en fonction de l'argument  $n$ , le nombre de multiplications effectuées par votre fonction.

## 3 Mastermind

On propose d'étudier le jeu du Mastermind. Rappelons les règles du jeu : un des deux joueurs choisit une suite de cinq couleurs secrète, que l'autre joueur devra deviner. Pour ce faire, il propose lui-même une suite de cinq couleurs, et le premier joueur lui indique parmi celles-ci combien figurent dans la combinaison à trouver, et combien figurent dans la combinaison à la bonne position. L'opération est répétée jusqu'à ce que le deuxième joueur propose la bonne combinaison.

On notera  $n$  le nombre de couleurs dans la suite à deviner (donc  $n = 5$ ) et  $k$  le nombre de couleurs différentes possibles. On supposera pour simplifier que les couleurs sont les éléments de  $\llbracket 0, k \llbracket$ , ce qui revient à numérotter les couleurs de 0 à  $k - 1$ .

Une combinaison sera représentée en Python par un tableau de longueur  $n$  rempli d'éléments de  $\llbracket 0, k \llbracket$ .

1. Écrire une fonction `comparaison0` prenant deux combinaisons `à_deviner` et `essai` représentant la combinaison choisie par le joueur 1 et celle proposée par le joueur 2 et renvoyant le nombre de couleurs de `essai` qui sont aussi présentes dans `à_deviner`.
2. Reprenons les combinaisons `à_deviner` et `essai` de la question précédente. Dans le jeu, le joueur 1 doit en réalité indiquer :
  - le nombre de couleurs de `essai` qui sont présentes au même emplacement dans `à_deviner`
  - ainsi que le nombre de couleurs de `essai` qui sont présentes dans `à_deviner` dans une autre place, mais une autre place qui n'a pas déjà été prise en compte ci-dessus.

Programmer ceci.

3. (\*\*\*\*) Écrire une fonction **toutesLesCombinaisons** qui prend en entrée  $n$  et  $p$  et qui renvoie un tableau contenant toutes les combinaisons possibles.

*Indication* : Utiliser  $n$  boucles « for » serait inesthétique et impossible sans connaître  $n$  au moment de programmer. On peut par exemple créer puis utiliser une fonction **prochaineCombinaison** qui prend une combinaison et renvoie la suivante.

*Une méthode basique pour jouer au MasterMind consiste à jouer des combinaisons au hasard parmi celles qui sont compatibles avec les résultats obtenus avec les combinaisons précédemment essayées.*

4. Écrire une fonction prenant en entrée :
- Un tableau **possibilités** contenant plusieurs combinaisons ;
  - Une combinaison **c** ainsi qu'un couple (**bien\_placées**, **mal\_placées**) qui représente le résultat de la comparaison de **c** à la combinaison à deviner,
- et qui renvoie la liste des combinaisons de **possibilités** qui sont compatibles avec le résultat obtenu pour **c**, autrement dit telles que si elles avaient été la combinaison cherchée, le résultat de la comparaison avec **c** aurait été (**bien\_placées**, **mal\_placées**).
5. En déduire une fonction simulant une partie de MasterMind. Les deux joueurs seront simulés par l'ordinateur. Il faudra donc tirer au hasard une combinaison puis générer la liste de toutes les combinaisons possibles. Ensuite, tant que la liste des possibilités contient au moins deux éléments, proposer une nouvelle combinaison et éliminer les combinaisons devenues alors impossibles.
6. Dans le jeu originel, le joueur 2 a droit à 12 tentatives pour trouver la bonne combinaison. Modifier la fonction précédente pour indiquer en combien d'essai la bonne combinaison a été trouvée.