Colle 07

Cette septième colle vous fera écrire une fonction simple en OCaml sur des listes, et d'autres sur un type définit manuellement. On conclut par une simulation numérique aléatoire.

Une petite fonction sur une liste

1. Écrivez en OCaml une fonction `iteration` qui prend en entrée une liste `li` (de type `'a list`), une fonction `f` de type `'a -> unit` et applique dans l'ordre `f` à chaque élément de la liste. La fonction devra être récursive et ne pas utiliser `List.nth` mais un parcours récursif de la liste avec du *pattern matching*. Elle se comportera comme `List.iter` ;
2. Testez la sur plusieurs listes. Que va-t-il se passer sur une liste vide ? Expliquez.

Un type maison pour des vecteurs 3D

On cherche à écrire des fonctions qui calculent sur des vecteurs de $\overrightarrow{x} \in \mathbb{R}^3$.

1. Plutôt que de travailler avec tableaux de flottants en OCaml de taille n, on travaillera avec un type enregistrement à trois champs, `x`, `y` et `z`. Définir ce type avec des champs non mutables, et au moins deux exemples de vecteurs `vec1` et `vec2` qui seront utilisés pour les exemples suivants ;
2. Écrivez une fonction `norme` qui prenne un vecteur 3D et renvoie sa norme L^2 définie par $|\overrightarrow{x}| = \sqrt{\sum_{i=1}^{3} x_i^2}$. On rappelle que les principales fonctions mathématiques sont disponibles sans avoir besoin de les importer, grâce au module `Stdlib` qui est entièrement importé par défaut (sa documentation est en ligne sur `https://ocaml.org/api/Stdlib.html` si besoin) ;
3. Écrivez une fonction `distance` qui prend deux vecteurs 3D et renvoie leur distance L^2 définie par $d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\sum_{i=1}^{3} (x_i - y_i)^2}$;
4. Écrivez une fonction `somme` qui prend deux vecteurs 3D et renvoie leur somme ;
5. De même, écrivez une fonction `difference` qui prend deux vecteurs 3D et renvoie leur différence ;
6. Enfin, écrivez une fonction `produit_exterieur` qui prend un vecteur 3D et un scalaire et renvoie le produit extérieur du vecteur par ce scalaire, c'est-à-dire $\lambda.\overrightarrow{x} = [\lambda x_1, \lambda x_2, \lambda x_3]$.

Simulation aléatoire

On rappelle que le module `Random` contient des fonctions pour générer des nombres pseudo-aléatoires. On pensera à l'initialiser une fois avec `Random.self_init()` (qui utilise du "vrai aléatoire" venant du monde extérieur). On rappelle que `Random.randint n` génère un entier aléatoire uniforme entre `0` et `n-1` inclus.

1. Écrire une fonction `date_anniversaire_aleatoire ()` qui prend comme argument le `()` (de type unit) et renvoie une date d'anniversaire représentée comme un simple entier entre 1 et 365 (on ignore les années bissextiles) ;

2. Écrire une fonction `collision_classe n` qui prend un entier `n` qui représente la taille d'une classe (par exemple `n=46`), puis simule les jours d'anniversaire de tous les élèves, et renvoie `True` si (au moins) deux élèves ont le même anniversaire (on dit alors qu'il y a *collision*), ou `False` sinon (ie si tous les anniversaires sont différents). On pourra utiliser une `ref` sur une liste des anniversaires déjà générés, et utiliser `List.mem element liste` pour tester l'appartenance de `element` à `liste` ;
3. Écrire une fonction `int_of_bool` qui prend un booléen et renvoie `1` pour `True` et `0` pour `False` ;
4. Écrire une fonction `collisions_classe n nbRepets` qui prend un entier `n` et renvoie la somme du nombre de fois où il y a eu collision en utilisant la fonction de simulation `collision_classe n`, `int_of_bool` sur son résultat, et une somme implémentée à la main avec une référence et une boucle for sur `nbRepets` ;
5. (plus difficile) Écrire une fonction `histogramme nMin nMax nbRepets` qui renvoie une liste contenant le résultat de `nbRepets` répétitions aléatoires de collisions d'anniversaire, pour des classes de tailles `n` allant de `nMin` à `nMax` (inclus). En prenant `nbRepets` assez grand (par exemple `1000`), à partir de quelle taille de classe observe-t-on une probabilité plus grande que 50% d'avoir une collision ?

