# TD : Détection et correction

## I. Générateur aléatoire d'erreurs

Pour commencer, vous allez créer un générateur aléatoire d'erreurs : cet outil vous permettra de simuler les erreurs qui peuvent survenir lors du traitement ou du partage de l'information.

**Remarque :** Dans tout ce TP, on suppose que toutes les listes d'entiers manipulées ne contiennent que des 0 et des 1 (il n'est donc pas nécessaire de vérifier que c'est bien le cas)

**Question 1.** Ecrire une fonction `booleen_aleatoire`, telle que `booleen_aleatoire(n)` a `n`% de chances renvoyer `True`, et sinon renvoie `False`.

**Question 2.** Ecrire une fonction `ajouter_erreurs`, telle que `ajouter_erreurs l n` renvoie une copie de la liste `l` dont chaque élément a `n`% de chances d'avoir été modifié (les 0 sont changés en 1, et les 1 en 0).

**Question 3.** Ecrire une fonction `compter_erreurs`, telle que `compter_erreurs l1 l2` renvoie le nombre de différences entre les listes `l1` et `l2`.

**Remarque :** Si `l1` et `l2` ne sont pas de même taille, utiliser l'instruction `raise Exception ("Message d'erreur")` pour renvoyer un message d'erreur.

## II. Ajout de redondance

**Question 4.** Ecrire une fonction `ajouter_redondance`, telle que `ajouter_redondance l` renvoie une liste où chacun des bits présents dans la liste `l` a été triplé.

**Question 5.** Ecrire une fonction `supprimer_redondance`, telle que `supprimer_redondance l` renvoie une liste où chacun des triplets de bits présents dans la liste `l` a été « détriplé ».

**Remarque :** Utilisez l'instruction `raise Exception ("Message d'erreur")` pour renvoyer un message d'erreur si la liste contient un triplet de bits dont toutes les valeurs ne sont pas identiques.

**Question 6.** Ecrire une fonction `detecter_via_redondance`, telle que `detecter_via_redondance l` renvoie le nombre d'erreurs détectées dans la liste `l` (qu'on suppose redondante).

**Indice :** Le nombre d'erreurs détectées est le nombre de triplets de bits où dont toutes les valeurs ne sont pas identiques.

**Question 7.** Ecrire une fonction `corriger_via_redondance`, telle que `corriger_via_redondance l` renvoie une copie de la liste `l` (qu'on suppose redondante) dans laquelle les erreurs détectées ont été automatiquement corrigées.

**Question 8.** Combiner les fonctions écrites dans les parties I et II pour tester cette méthode de détection et de correction d'erreurs. Cette méthode fonctionne-t-elle dans tous les cas de figure ? Si non, quels sont les cas où des erreurs ne sont pas détectées ?

## III. Bits de contrôle

**Remarque :** Pour des raisons de simplicité, dans la suite de ce TP, on supposera que la longueur des listes à transmettre est toujours un multiple de 10.

**Question 9.** Ecrire une fonction `ajouter_bits_de_controle`, telle que `ajouter_bits_de_controle l` renvoie une copie de la liste `l` dans laquelle un bit de contrôle a été ajouté tous les 10 bits.

**Question 10.** Ecrire une fonction `supprimer_bits_de_controle`, telle que `supprimer_bits_de_controle l` renvoie une liste dont on a supprimé les bits de contrôle.

**Remarque :** Utiliser l'instruction `raise Exception ("Message d'erreur")` pour renvoyer un message d'erreur si la longueur de la liste n'est pas un multiple de 11.

**Question 11.** Ecrire une fonction `detecter_via_bits_de_controle`, telle que `detecter_via_bits_de_controle l` renvoie le nombre d'erreurs détectées dans la liste `l` (dont on suppose qu'elle contient des bits de contrôle).

Indice : Le nombre d'erreurs détectées est le nombre de lots de 11 bits dont le bit de contrôle ne correspond pas aux 10 premiers éléments.

Question 12. Combiner les fonctions écrites dans les parties I et III pour tester cette méthode de détection. Cette méthode fonctionne-t-elle dans tous les cas de figure ? Si non, quels sont les cas où des erreurs ne sont pas détectées ?