TP numéro 2

1. Fonctions sur les listes

• Écrire une fonction qui supprime tous les éléments d'une liste qui sont égaux à un élément `e` passé en argument.

• Écrire une fonction qui vérifie que tous les éléments d'une liste sont égaux.

• Écrire une fonction qui, étant donnée une liste de longueur paire, regroupe deux éléments consécutifs dans une liste.

```
(regroupe '(a b c d e f)) => ((a b) (c d) (e f))
```

• Écrire une fonction qui rend la liste de tous les éléments de rang impair d'une liste.

• Écrire une fonction qui substitue un symbole par un autre dans une liste.

```
(substitue 'a 'b '(a b c a e)) => (b b c b e)
```

2. Mémorisation

• Écrire une fonction qui rend la liste des n+1 premiers nombres de la suite de Fibonacci (de u_n à u_0) sans faire plusieurs fois les mêmes calculs.

```
(fibo-liste 5) → (8 5 3 2 1 1)
```

• Utiliser la fonction `fibo-liste` pour écrire une nouvelle version de la fonction écrite en TD qui calcule le $n^{ième}$ terme de la suite de Fibonacci. Comparez le nombre de calculs effectués pour n=4. Testez les deux fonctions pour n=30. Etes-vous maintenant convaincu(e) de l'intérêt de calculer la complexité d'un algorithme ?

La fonction prédéfinie `random` permet d'engendrer des nombres entiers au hasard : `(random x)` retourne un entier dans l'intervalle [0, x[.

• Écrire une fonction qui retourne une liste de nombres entiers positifs pris au hasard. Les deux paramètres de cette fonction sont la longueur de la liste à construire et la valeur maximale des nombres à engendrer.

```
(liste_random 5 10) → (7 9 6 7 4)
```

• Écrire une fonction qui retourne une liste composée d'un nombre entier pris au hasard entre 0 et N, et d'un booléen indiquant si ce nombre est un multiple de trois ou de sept.

```
(nb_test 10) → (5 #f)
(nb_test 10) → (6 #t)
```

• Écrire une fonction identique à la fonction liste_random, mais qui ne retourne que des nombres pairs.

```
(liste_random_pairs 5 10) → (4 10 6 4 6)
```

• Écrire une fonction qui étant donnée une liste, construit une liste de deux sous-listes : celle contenant les atomes et celle contenant les listes.

```
(trie '(a 5 (r 2) 8 toto "az e" ("r" 7 b) t))
              → ((a 5 8 toto "az e" t) ((r 2) ("r" 7 b)))
```