TP numéro 2

1 Fonctions sur les listes

• Écrivez une fonction qui ajoute 10 à tous les éléments d'une liste de nombres.

```
(ajoute10 '(1 3 2 4)) -> (11 13 12 14)
```

• Écrivez une fonction qui renvoie la sous-liste formée de tous les symboles d'une liste.

```
(symboles '(a 2 b (6 z) "toto" 7 f)) -> (a b f)
```

• Écrivez une fonction qui renvoie la sous-liste formée des n premiers éléments d'une liste.

```
(premiers 3 '(a b c d e)) -> (a b c)
```

• Écrivez une fonction qui remplace toutes les occurrences d'un élément e_1 dans une liste L par un autre élément e_2.

```
(remplace 'o 'i '(b o n j o u r)) -> (b i n j i u r)
```

2 Mémorisation

• Écrire une fonction qui calcule $\frac{n!+100}{n!+4}$ avec un seul appel à (factorielle n).

• Écrire une fonction qui rend la liste des n+1 premiers nombres de la suite de Fibonacci (de u_0 à u_n) 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 ?

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

```
(trie '(tor 1 tue la 2 3 pin 4)) → ((tor tue la pin) (1 2 3 4))
```

• Écrire une fonction qui étant donnée une liste, construit une liste où les éléments de la liste initial ont été permutés deux à deux.

```
(permute2a2 '(a b c d e f) → (b a d c f e))
```

• Écrire la fonction `som-prod` qui, étant donné une liste de nombres, renvoie une liste de deux nombres correspondant à la somme et au produit des éléments de la liste.

```
(som-prod '(1 2 3 4)) → (10 24)
```

• Écrire la fonction `tous-egaux` qui, étant donné une liste, teste si tous les éléments de la liste sont égaux.

• Écrire la fonction `parite` qui, étant donné une liste de nombres, renvoie une liste de deux sous-listes contenant les éléments pairs et impairs de la liste initiale.

```
(parite '(1 6 7 10 5)) → ((1 7 5) (6 10))
```