TD numéro 3 : let

À préparer avant la séance

• Donner la valeur retournée par les expressions ci-dessous :

- `(list 'a (cons '(b c) '(d)))`
- `(append '(a) '(b c) '(d))`
- `(append (list 'a '(b)) '(c))`

• On définit les listes suivantes :

```
(define L1 '(a b c))
(define L2 '(d e))
```

Donner les expressions utilisant L1 et L2 et permettant d'obtenir les résultats suivants :

```
… → ((b c) (d e))
… → (c e)
… → (b c d)
```

• Écrire en utilisant le let une fonction qui, étant donnés les coefficients d'un trinôme, rend les racines sous forme d'une liste. La liste sera vide si Δ<0, n'aura qu'un élément si Δ=0, et deux éléments sinon.

```
(racines 1 2 -3) → (-3 1)
```

À faire pendant la séance

• Nous souhaitons définir la fonction `som-prod` qui retourne la somme et le produit d'une liste non vide de nombres.

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

1. Écrire une première version récursive de la fonction sans utiliser le let. Dérouler ensuite le fonctionnement de cette fonction sur l'exemple ci-dessus.

2. Écrire une seconde version de cette fonction en y intégrant le let, c'est-à-dire en utilisant le résultat de l'appel récursif pour effectuer les calculs. Les calculs seront effectués de manière habituelle, c'est-à-dire en remontant.

3. Écrire ensuite une troisième version qui, bien qu'étant récursive, s'inspire de la programmation itérative, en utilisant un paramètre supplémentaire pour effectuer les calculs en descendant.

• Écrire une fonction qui, étant donnée une liste d'entiers, construit une liste de deux sous-listes : celle contenant les éléments impairs et celle contenant les éléments pairs.

```
(parite '(1 2 6 5 7)) → ((1 5 7) (2 6))
```