

# Le langage Racket

---

- Un langage de programmation fonctionnelle
- 20/11/2021

# 1. Exercices

## Exo 1:

Définir un prédictat **isin?** permettant de savoir si un élément appartient à une liste.

Appeler **(isin? '(1 2 3 4 5) 6)** retourne faux.

Appeler **(isin? '(1 2 3 4 5) 3)** retourne vrai.

*(fonctions utiles : empty?, first, rest)*

# 1. Exercices

correction 1:

```
1 (define (isin? L e)
2   (if (empty? L)
3     #f
4     (if (= e (first L))
5       #t
6       (isin? (rest L) e)
7     )))
8 (define L '(1 2 3 4 5))
9 (isin? L 6)
10 (isin? L 3)
```

```
#f
#t
```

## 1. Exercices

Exo 2:

Définir une fonction L->NL qui démultiplie N fois une liste passée en paramètre.  
Appeler (L->NL '(1 2 3 4 5) 4) retourne la liste (1 1 1 1 2 2 2 2 3 3 3  
3 4 4 4 4 5 5 5 5).

(*fonctions utiles : empty?, cons, first, rest*)

# 1. Exercices

correction 2 :

```
1 (define (L->NL L n)
2   (define (f L i)
3     (if (empty? L) empty
4         (if (= i 0)
5             (f (rest L) n)
6             (cons (first L) (f L (- i 1)))))))
7   (f L n))
8 (L->NL '(1 2 3 4 5) 4)
```

```
'(1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5)
```

# 1. Exercices

## Exo 3:

Définir une fonction **nieme** retournant le N-ième élément d'une liste d'entiers positifs ; si N est négatif, alors la fonction retourne -1 ; si N va au delà de la liste, alors la fonction retourne -1 ; ne pas utiliser **list-ref**.

Appeler (**nieme** '(1 2 3 4 5) -1) retourne -1.

Appeler (**nieme** '(1 2 3 4 5) 0) retourne 1.

Appeler (**nieme** '(1 2 3 4 5) 2) retourne 3.

Appeler (**nieme** '(1 2 3 4 5) 10) retourne -1.

(*fonctions utiles : length, first, rest*)

# 1. Exercices

correction 3:

```
1 (define (nieme L i)
2   (define (f L i)
3     (if (= i 0)
4       (first L)
5       (f (rest L) (- i 1)))
6     ))
7   (if (or (>= i (length L)) (< i 0))
8     -1
9     (f L i)
10   )))
11 (nieme '(5 4 3 2 1) 0)
12 (nieme '(5 4 3 2 1) 2)
```

5

3

# 1. Exercices

## Exo 4:

Définir une fonction `plat` retournant une liste aplatie.  
Appeler `(plat '(3 (2 (1 ()))))` retourne la liste `(3 2 1)`.  
*(fonctions utiles : `empty?`, `list?`, `append`, `first`, `rest`)*

# 1. Exercices

correction 4:

Première solution :

```
1 (define (plat L)
2   (if (empty? L) empty
3     (if (list? (first L))
4       (append (plat (first L)) (plat (rest L)))
5       (cons (first L) (plat (rest L))))
6     )))
7 (plat '(3 (2 (1 ()))))  
'(3 2 1)
```

# 1. Exercices

correction 4:

Deuxième solution :

```
1 (define (plat L)
2   (if (list? L)
3     (if (empty? L) empty
4       (append (plat (first L)) (plat (rest L)))))
5     (list L)
6   )))
7 (plat '(3 (2 (1 ()))))  
  
'(3 2 1)
```

# 1. Exercices

## correction 4:

Troisième solution (avec fonction auxiliaire, accumulateur et variable locale) :

```
1 (define (plat L)
2   (define (g L1 L2)
3     (if (empty? L1) L2
4       (let ([i (first L1)])
5         (if (list? i)
6           (g (rest L1) (g i L2))
7           (g (rest L1) (cons i L2))
8         )))
9      )))
10 (g L '())
11 )
12 (plat '(3 (2 (1 ()))))  

  

  ,(3 2 1)
```

## 1. Exercices

### Exo 5:

Définir une fonction **fusion** retournant une liste fusionnant deux listes passées en paramètre ; le comportement de **fusion** est similaire à la fonction **append** sans obligatoirement respecter l'ordre des éléments.  
*(fonctions utiles : **empty?**, **cons**, **first**, **rest**)*

# 1. Exercices

correction 5:

```
1 (define (fusion A B)
2   (if (empty? A) B
3       (fusion (rest A) (cons (first A) B)))
4   ))
5 (define L '(1 (2 (3))))
6 (define P '( 1 2 3))
7 (fusion L P)
8 (append L P)
```

```
'((2 (3)) 1 1 2 3)
'(1 (2 (3)) 1 2 3)
```

Remarque : remplacer `(if (empty? A) B` par `(if (empty? A) B (if (empty? B)`) A aura dans certains cas pour effet de ralentir l'exécution de la fonction, pour une terminaison plus courte en nombre d'appels au détriment d'une augmentation du nombre de tests par appel.

# 1. Exercices

## Exo 6:

Définir une fonction **sublist** retournant une demi-liste d'une liste ; la liste est passé en argument ; un deuxième argument définit si la liste retournée est la première demi-liste (celle des valeurs d'indices pairs) ou la deuxième demi-liste (celle des valeurs d'indices impairs).

Appeler (**sublist** '(1 2 3 4 5 6) 0) retourne la liste (1 3 5).

Appeler (**sublist** '(1 2 3 4 5 6) 1) retourne la liste (2 4 6).

Appeler (**sublist** '(1 2 3 4 5) 1) retourne la liste (2 4).

(*fonctions utiles : empty?, cons, first, rest, reverse*)