

Implémentation

~~I'm a Programmar~~
~~I'm a Programer~~
~~I'm a Programmor~~
I write code.

04 mars 2019

[Damien Morard](#)

Au cours de cette séance d'exercices, vous implémenterez des règles récur-sives en [LogicKit](#).

Exercice 1 : Implémenter la récursivité

La dernière séance nous avons regardé plus théoriquement comment fonctionne les règles récursives grâce à la *Programmation Logique*. Nous allons reprendre les deux précédentes séances d'exercices et enfin implémenter les différents problèmes!

Pour ce premier exercice, nous reprenons les pseudo-codes de la session d'exercice 3 et nous allons tenter de les mettre sous le bon format. Pour vous aider et vous éviter de devoir reconstruire toute la structure LogicKit, vous avez déjà un projet swift avec toutes les bases nécessaires pour démarrer. Votre but implémenter les règles suivantes dans votre base de connaissance, et tester vos résultats.

1. Implémenter et tester op1.

```
.fact("op1", 0, y, y)
.rule("op1", succ(x), y, res) {
    .fact("op1", x, succ(y), res)
}
```

2. Implémenter et tester op2.

```
.fact("op2", succ(x), 0),  
.rule("op2", succ(x), succ(y)) {  
    .fact("op2", x, y)  
}
```

3. Implémenter et tester op3.

```
.fact("op3", cons(x, y), x),  
.rule("op3", cons(head, tail), x) {  
    .fact("op3", tail, x)  
}  
list = cons(3, cons(1, cons(5, empty)))
```

4. Implémenter et tester op4.

```
.fact("op4", empty, x, cons(x, empty)),  
.rule("op4", cons(head, tail), x, cons(head, res)) {  
    .fact("op4", tail, x, res)  
},  
list = cons(3, cons(1, cons(5, empty)))
```

5. Implémenter et tester op5.

```
.fact("op5", empty, 0),  
.rule("op5", cons(head, tail), succ(res)) {  
    .fact("op5", tail, res)  
}
```

6. Implémenter et tester op6.

```
.fact("op6", empty, x, cons(x, empty)),  
.rule("op6", cons(head, tail), x, cons(head, res)) {  
    .fact("op6", tail, x, res)  
}
```

7. Implémenter et tester op7.

```
.fact("op7", cons(x, empty), x),  
.rule("op7", cons(head, tail), x) {  
    .fact("op7", tail, x) &&  
    x > head  
},  
.rule("op7", cons(head, tail), head) {  
    .fact("op7", tail, x) &&  
    head >= x  
}
```

Exercise 2 : Palindrome



Ce second exercice va mettre en pratique toutes vos connaissances sur la programmation logique ! Écrivez la règle **palindrome** prenant en paramètre une liste de caractères. Cette règle n'est évaluée à vrai que si cette liste forme un palindrome, c'est-à-dire qu'elle peut être lue de gauche à droite comme de droite à gauche, en formant le même mot. Exemple de palindrome : "abba", "2002", "Tu l'as trop écrasé César ce Port-Salut"... Dans notre cas les mots seront exprimés sous forme de liste.

```
.rule("palindrome", mot) :- ...
```

1. Pour cela, écrivez une règle **renverser** qui renverse l'ordre des éléments d'une liste. Vous pourrez vous aider de la règle **concat** qui existe déjà sous LogicKit.

```
.rule("renverser", list, res) ...
```

La fonction **concat** en LogicKit permet de concaténer deux listes. Les deux arguments **DOIVENT** être des listes de termes, et non un simple terme. N'oubliez pas non plus d'importer la base de connaissance des listes ! (L'opérateur **+** est utilisé pour la fusion de deux bases de connaissances).

```
// Two ways to write a list of terms
let list1: Term = List.from([1,2].map(Nat.from))
let list2: Term = List.cons(Nat.from(4), List.empty)
let res: Term = .var("res")
let query = kb.ask(List.concat(list1, list2, res))

for binding in query {
  print(binding["res"])
}
```

2. Écrivez maintenant la règle **palindrome**.
3. Pouvez-vous obtenir tous les palindromes possibles ? Si oui, comment ? Si non, pourquoi ?