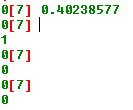
***Inteligencia Artificial: Práctica 1***

***Ejercicio 1:***

En el apartado **1.1**, al ejecutar el código recursivo nos ha quedado lo siguiente:

1. (cosine-distance-rec ’(1 2) ’(1 2 3)) = 0.40238577

2. (cosine-distance-rec nil ’(1 2 3)) = 1

3. (cosine-distance-rec ’() ’()) = 0

4. (cosine-distance-rec ’(0 0) ’(0 0)) = 0

Al ejecutar el mismo código con las funciones para mapcar nos dan los mismos resultados. Hemos decidido que cuando dos listas estén vacías el resultado sea 0 pues la distancia entre dos vectores vacíos debe ser nula, así como para dos vectores nulos. En caso de que solamente una de las dos listas esté vacía devuelve un 1.

En el apartado **1.2** nos sale lo siguiente:

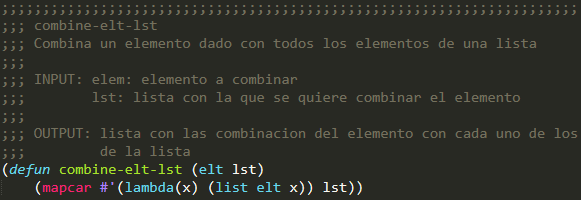
1. (order-vectors-cosine-distance ’(1 2 3) ’()) =

2. (order-vectors-cosine-distance ’() ’((4 3 2) (1 2 3))) =

***Ejercicio 2:***

***Ejercicio 3:***

En el apartado **3.1** teníamos que combinar un elemento con una lista, dando lugar a los distintos pares con el siguiente código:



Podemos observar que nos dan los siguientes resultados:

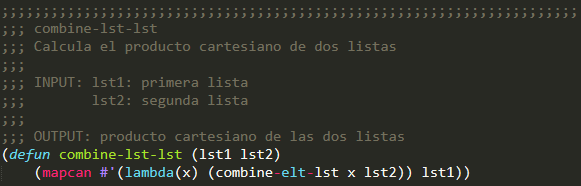
(combine-elt-lst 'a '(1 2 3)) -> ((A 1) (A 2) (A 3))

(combine-elt-lst 'a nil) -> nil

(combine-elt-lst nil nil) -> nil

(combine-elt-lst nil '(a b) -> ((nil a) (nil b))

A continuación, en el apartado **3.2 s**e pedía combinar dos listas, para ello hemos utilizado la función combine-elt-lst anterior recursivamente:



Dándonos los siguientes resultados:

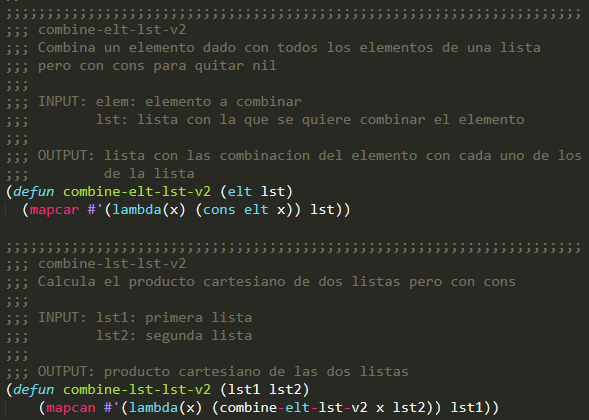
(combine-lst-lst '(a b c) '(1 2)) -> ((A 1) (A 2) (B 1) (B 2) (C 1) (C 2))

(combine-lst-lst nil nil) -> nil

(combine-lst-lst '(a b c) nil) -> nil

(combine-lst-lst nil '(a b c)) -> nil

Finalmente, en el apartado **3.3** había que combinar n listas. Puesto que *list* no excluye el caso *nil*, hemos creado dos funciones nuevas que utilizan  *cons* para evitar que *nil* aparezca en las listas:



Son básicamente iguales que las iniciales y con ellas hemos creado la siguiente: