Ejercicio 21 - Clojure

Definir una función para obtener la matriz triangular superior (incluyendo la diagonal principal) de una matriz cuadrada que está representada como una lista de listas.

Iterativo

En principio se dividió el problema en dos partes.

El primero consistió en que partiendo de una matriz(en formato lista de listas) logremos transformar la lista de listas, en una lista de tuplas donde cada posición corresponde a una tupla <**posición**, **lista**>, donde:

posición: es la posición de la lista que acompaña.
lista: lista de números.

La función propuesta es:

```
(defn indexar_matriz [l]
    (map-indexed list l)
)
```

Por otro lado, partiendo de la lista de tuplas, teniendo en cuenta la posición que tenemos de cada tupla, utilizamos una transformación(***map***) aplicando la función ***nthrest,*** que recibirá la lista actual en la que está iterando y la posición, recortando la lista y devolviendola recortada.
A si mismo dada la lista recortada, la concatenamos a una nueva lista generada, con tantos ceros como posiciones aparezcan en cada iteración. Para ello se utiliza la función ***repeat***
El resultado final será una lista de listas. A continuación las funciones asociadas:

```
(defn _triangular_superior [l]
  (map (fn [[a b]] (concat (repeat a 0)  (nthrest b a))) l)
)

(defn matriz_triangular_superior [l]
  (-> l
    (indexar_matriz)
    (_triangular_superior)
  )
)
```

Ejemplo de aplicación:

```
(-> '((1 2 3) (4 5 6) (7 8 9))
  (matriz_triangular_superior)
  (println)
  )
```

Al aplicar matriz_triangular_superior, primero se transforma la lista de listas recibida al aplicar ***indexar_matriz*** que devuelve la siguiente lista:

```
Resultado: ((0 (1 2 3)) (1 (4 5 6)) (2 (7 8 9)))
```

Luego se aplica seguidamente a esa nueva lista, la función ***_triangular_superior***, donde aplica map a una función anónima definida previamente y en cada paso transforma el elemento recibido:

1) Recibe `(0 (1 2 3))` en la primera iteración, con `a = 0`; `b = (1 2 3)`.
   Genera una lista de ceros con: `(repeat 0 0)`. En este caso será una lista vacía.
   Genera una lista recortada con: `(nthrest '(1 2 3) 0)` . En este caso será la lista sin recortar, quedando: `(1 2 3)`
   Concatena la lista con: `(concat '() (1 2 3)) , Resultando: (1 2 3)`

2) Recibe `(1 (4 5 6))`, con `a = 1`; `b = (4 5 6)`.
   `Aplica: (repeat 1 0). Resulta: (0)` .
   `Aplica: (nthrest '(4 5 6) 1) .Resulta: (4 5)`.
   `Concatena: (concat '(0) (4 5)) , Resultando: (0 4 5)`.

3) Recibe `(2 (7 8 9))`, con `a = 2`; `b = (7 8 9)`.
   `Aplica: (repeat 2 0). Resulta: (0 0)` .
   `Aplica: (nthrest '(7 8 9) 2) . Resulta: (9)`.
   `Concatena: (concat '(0 0) '(4 5)) , Resultando: (0 0 9)`.

```
Resultado: ((1 2 3) (0 5 6) (0 0 9))
```

Ejercicio 22 - Clojure

Definir una función para obtener la diagonal principal de una matriz cuadrada que está representada como una lista de listas.

Iterativo

En principio se dividió el problema en dos partes análogamente al 21.

Reutilizamos la función indexar_matriz definida previamente, para indexar la lista de listas:

```
(defn indexar_matriz [l]
    (map-indexed list l)
)
```

Por otro lado, definimos una función llamada ***_diagonal_principal ,*** que partiendo de la lista de tuplas que nos devuelve el indexado, teniendo en cuenta contamos con la posición que tenemos de cada tupla, utilizamos una transformación(***map***) aplicando la función ***nth,*** que recibirá la lista actual en la que está iterando y la posición, devolviendo en cada paso una el elemento iésimo de la diagonal principal, devolviendo al final una lista de números de la diagonal principal.
A continuación las definiciones de funciones:

```
(defn _diagonal_principal [l]
  (map (fn [[a b]] (nth b a) ) l)
)

(defn matriz_diagonal_principal [l]
  (-> l
    (indexar_matriz)
    (_diagonal_principal)
  )
)
```

Ejemplo de aplicación:

```
(-> '((1 2 3) (4 5 6) (7 8 9))
  (matriz_diagonal_principal)
  (println)
  )
```

Al aplicar ***matriz_diagonal_principal*** , primero se transforma la lista de listas recibida al aplicar ***indexar_matriz ,*** devolviendo la siguiente lista:

`Resultado: ((0 (1 2 3)) (1 (4 5 6)) (2 (7 8 9)))`

Luego se aplica seguidamente a esa nueva lista, la función ***`_diagonal_principal`***, donde aplica map a una función anónima definida previamente y en cada paso transforma el elemento recibido de la siguiente forma:

1. Recibe `(0 (1 2 3))` en la primera iteración, con `a = 0`; `b = (1 2 3)`.
   `Aplica: (nth '(1 2 3) 0) , Resultando: 1`

2. Recibe `(1 (4 5 6))`, con `a = 1`; `b = (4 5 6)`.
   `Aplica: (nth '(4 5 6) 1) , Resultando: 5`

3. Recibe `(2 (7 8 9))`, con `a = 2`; `b = (7 8 9)`.
   `Aplica: (nth '(7 8 9) 2) , Resultando: 9`

`Resultado: (1 5 9)`