

Laboratorio 12

Para este laboratorio cree un repositorio de github

- Grabe un video de no más de 10 minutos donde muestre la ejecución de su programa para el ejercicio1. Súbalo a YouTube como video no listado y adjúntelo en el README. de su repositorio.
- Para este ejercicio utilice cualquier otro lenguaje que permita programación funcional que no sea python.
- Para los incisos que no involucren la generación de código, cree una carpeta por separado en su repositorio y coloque las respuestas en un documento con formato PDF

Problema 1: 25%

- Escriba la reducción- β de la operación lógica NOT

$$\begin{aligned} \text{NOT TRUE} \\ = (\lambda b. b \text{ FALSE } \text{ TRUE}) \text{ TRUE} \\ \rightarrow \beta \text{ TRUE } \text{ FALSE } \text{ TRUE} \\ = (\lambda t. \lambda f. t) \text{ FALSE } \text{ TRUE} \\ \rightarrow \beta (\lambda f. \text{ FALSE}) \text{ TRUE} \\ \rightarrow \beta \text{ FALSE} \end{aligned}$$

$$\begin{aligned} \text{NOT FALSE} \\ = (\lambda b. b \text{ FALSE } \text{ TRUE}) \text{ FALSE} \\ \rightarrow \beta \text{ FALSE } \text{ FALSE } \text{ TRUE} \\ = (\lambda t. \lambda f. f) \text{ FALSE } \text{ TRUE} \\ \rightarrow \beta (\lambda f. f) \text{ TRUE} \\ \rightarrow \beta \text{ TRUE} \end{aligned}$$

- Escriba y explique como se vería la recursión y los ciclos.

La recursión consiste en que una función se llama a sí misma para procesar una estructura de datos o repetir un cálculo, reduciendo el problema hasta llegar a un caso base. Para permitir la recursión en el cálculo- λ , que no admite funciones nombradas, se emplea el combinador de punto fijo, que permite definir funciones que se auto-referencian. Además de la recursión directa, también se simulan “ciclos” mediante funciones de orden superior como map, filter o fold, las cuales aplican funciones a colecciones de valores paso a paso.

- Explique cuando es prudente usar este tipo de programación y cuando no. De un ejemplo para cada caso.

La programación funcional es recomendable cuando se necesita trabajar con operaciones sobre datos sin modificar estados, ya que facilita la claridad, la paralelización y las pruebas. Es ideal para tareas como análisis y transformación de información donde el flujo es predecible y basado en funciones pures.

Ejemplo: procesamiento de grandes volúmenes de datos en Spark o análisis de listas con map y filter.

Por otro lado, no es la mejor opción cuando se requiere manipulación constante del estado o rendimiento muy alto sobre estructuras mutables, como en programas interactivos o de tiempo real. En estos casos, el estilo imperativo suele ser más natural y eficiente.

Ejemplo: un motor de videojuegos que actualiza posiciones, colisiones y físicas en cada frame.

Problema 2: 25%

Escribir un programa que ordene una lista de diccionarios con respecto a un key indicado, usando funciones lambda.

Ejemplo

Si la lista original de diccionarios es:

```
1 [  
2     {'make': 'Nokia', 'model': 216, 'color': 'Black'},  
3     {'make': 'Mi Max', 'model': 2, 'color': 'Gold'},  
4     {'make': 'Samsung', 'model': 7, 'color': 'Blue'}  
5 ]
```

La salida (si ordenamos con respecto a key = 'model') debe ser:

```
1 [  
2     {'make': 'Mi Max', 'model': 2, 'color': 'Gold'},  
3     {'make': 'Samsung', 'model': 7, 'color': 'Blue'},  
4     {'make': 'Nokia', 'model': 216, 'color': 'Black'}  
5 ]
```

```
JS problema2.js > ...  
10 const data = [  
11     { make: 'Nokia', model: 216, color: 'Black' },  
12     { make: 'Mi Max', model: 2, color: 'Gold' },  
13     { make: 'Samsung', model: 7, color: 'Blue' }  
14 ];  
15  
16 const byModel = sortByKey('model');  
17 console.log(byModel(data));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTGRESQL QUERY RESU

```
PS C:\Users\UVG\Desktop\lab12> node problema2.js  
[  
    { make: 'Mi Max', model: 2, color: 'Gold' },  
    { make: 'Samsung', model: 7, color: 'Blue' },  
    { make: 'Nokia', model: 216, color: 'Black' }  
]  
PS C:\Users\UVG\Desktop\lab12>
```

Problema 3: 25%

Se tiene una matriz X (definida como lista de listas). Escribir un programa que calcule la matriz transpuesta X^T , usando funciones lambda. La salida debe darse también como una lista de listas.

Ejemplo

Si la matriz original es:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Entonces la salida es:

$$X^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
JS problema3.js > ...
1  const transpose = matrix =>
2    matrix[0].map(_, i) =>
3      matrix.map(row => row[i])
4    );
5
6
7  const X = [
8    [1, 2, 3],
9    [4, 5, 6],
10   [7, 8, 9]
11 ];
12
13 console.log(transpose(X));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PO
PS C:\Users\UVG\Desktop\lab12> node problema3.js
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
PS C:\Users\UVG\Desktop\lab12>

Problema 4: 25%

Escribir un programa que elimine elementos indicados de una lista, usando funciones lambda.

Los elementos a borrar deben indicarse como una lista, y la salida debe ser la lista de elementos remanentes.

Ejemplo

Si la lista inicial es:

```
1 ['rojo', 'verde', 'azul', 'amarillo', 'gris', 'blanco', 'negro']
```

Y la lista de elementos a borrar es:

```
1 ['amarillo', 'cafe', 'blanco']
```

Entonces la salida debe ser:

```
1 ['rojo', 'verde', 'azul', 'gris', 'negro']
```

Nota: Observe que 'café' no estaba en la lista original, por lo que simplemente se ignora.

```
Js problema4.js > ...
1 const removeItems = toRemove => list =>
2   list.filter(x => !toRemove.includes(x));
3
4 const original = ['rojo', 'verde', 'azul', 'amarillo', 'gris', 'blanco', 'negro'];
5 const borrar = ['amarillo', 'cafe', 'blanco'];
6
7 console.log(removeItems(borrar)(original));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTGRESQL QUERY RESULTS

```
PS C:\Users\UVG\Desktop\lab12> node problema4.js
[ 'rojo', 'verde', 'azul', 'gris', 'negro' ]
```