# PROGRAMACION FUNCIONAL

RODRIGUEZ MORENO MARCO ANTONIO

Aarón Hernández García

Actividad 4.1

11/02/2025

# Realiza un ejemplo de una estructura de Árbol y de una Lista de datos, comparando con dos Lenguajes de Programación Funcional.

## Haskell

Haskell es un lenguaje puramente funcional, con evaluación perezosa y tipado fuerte.

```haskell
data Tree a = Empty | Node a (Tree a) (Tree a) deriving Show

-- Insertar un elemento en un Árbol Binario de Búsqueda (BST)
insert :: (Ord a) => a -> Tree a -> Tree a
insert x Empty = Node x Empty Empty
insert x (Node v left right)
    | x < v     = Node v (insert x left) right
    | otherwise = Node v left (insert x right)

-- Ejemplo de uso
treeExample = foldr insert Empty [5,3,8,1,4]
```

```haskell
-- Definición de una Lista enlazada
data List a = Nil | Cons a (List a) deriving Show

-- Función para agregar un elemento al inicio de la lista
prepend :: a -> List a -> List a
prepend x xs = Cons x xs

-- Ejemplo de uso
listExample = prepend 1 (prepend 2 (prepend 3 Nil)) -- [1,2,3]
```

### Elixir

Elixir es un lenguaje funcional basado en la máquina virtual de Erlang, con un enfoque en concurrencia y programación distribuida.

```
defmodule Tree do
  defstruct value: nil, left: nil, right: nil

  def insert(nil, value), do: %Tree{value: value}
  def insert(%Tree{value: v, left: l, right: r} = node, value) do
    if value < v do
      %Tree{node | left: insert(l, value)}
    else
      %Tree{node | right: insert(r, value)}
    end
  end
end

# Ejemplo de uso
tree_example = Tree.insert(nil, 5)
|> Tree.insert(3)
|> Tree.insert(8)
|> Tree.insert(1)
|> Tree.insert(4)
```

```
# Definición de una Lista enlazada
defmodule List do
  defstruct head: nil, tail: nil

  def prepend(value, nil), do: %List{head: value, tail: nil}
  def prepend(value, list), do: %List{head: value, tail: list}
end

# Ejemplo de uso
list_example = List.prepend(1, List.prepend(2, List.prepend(3, nil)))
```

**Comparación:**

| Característica | Árbol en Haskell | Árbol en Elixir | Lista en Haskell | Lista en Elixir |
|---|---|---|---|---|
| **Definición** | `data` con `deriving Show` | `defstruct` | `data` con `deriving Show` | `defstruct` |
| **Inmutabilidad** | Sí | Sí | Sí | Sí |
| **Inserción** | Recursiva con `insert` | Recursiva con `Tree.insert` | Recursiva con `Cons` | Recursiva con `prepend` |
| **Patrón de acceso** | Evaluación perezosa | Evaluación estricta | Evaluación perezosa | Evaluación estricta |

**Conclusión:**

- **Haskell** tiene evaluación **perezosa**, lo que hace que las estructuras no se evalúen hasta que se necesiten.
- **Elixir** es más **estricto**, lo que puede ser útil en sistemas concurrentes.
- Ambos lenguajes utilizan **recursión** en lugar de iteraciones tradicionales.
- Haskell tiene un **sistema de tipos fuerte** y usa **listas inmutables** de forma nativa.
- Elixir es más **orientado a procesos**, por lo que las estructuras de datos son más explícitas.