¿QUÉ ES HASKELL?

Haskell es un lenguaje de programación puramente funcional, que trabaja manejando principalmente funciones lambda

Main.hs

```haskell
-- Declaración de la función sumList
sumList :: [Int] -> Int
sumList []     = 0
sumList (x:xs) = x + sumList xs
-- Declaración de la función square
square :: Int -> Int
square x = x * x
-- Declaración de la función que calcula la longitud de una lista
lengthOfList :: [a] -> Int
lengthOfList []     = 0
lengthOfList (_:xs) = 1 + lengthOfList xs

-- Ejecución
main :: IO ()
main = do
  -- Función sumList - Nuestra función de primer orden
  let numbers = [1, 2, 3, 4, 5]
      total = sumList numbers
  putStrLn ("La suma de la lista es " ++ show total)
  putStrLn ("--------------------------")
  -- Función square - Ejemplo de Transparencia Referencial
  let result1 = square 5
      result2 = square 5
  putStrLn ("El cuadrado de 5 es " ++ show result1)
  putStrLn ("El cuadrado de 5 es " ++ show result2)
  putStrLn ("--------------------------")
  -- Función lengthOfList - Ejemplo de Polimorfismo Paramétrico
  let list1 = [1, 2, 3, 4, 5]
      list2 = ["Haskell", "es", "genial"]
  putStrLn ("La longitud de la lista 1 es " ++ show (lengthOfList list1))
  putStrLn ("La longitud de la lista 2 es " ++ show (lengthOfList list2))
```

Console / Shell — Run

```
La suma de la lista es 15
--------------------------
El cuadrado de 5 es 25
El cuadrado de 5 es 25
--------------------------
La longitud de la lista 1 es 5
La longitud de la lista 2 es 3
```


PRESENTACIÓN POR: LAURA ORTIZ USME & MIGUEL ÁNGEL SALGAR OLARTE

FUNCIONES DE PRIMER ORDEN

```haskell
sumList :: [Int] -> Int
sumList []     = 0
sumList (x:xs) = x + sumList xs
```

```haskell
let numbers = [1, 2, 3, 4, 5]
    total = sumList numbers
putStrLn ("La suma de la lista es " ++ show total)
```

Es una función versátil y flexible que Haskell puede interpretar como cualquier otro valor, sirviendo como variables o parámetros para otras funciones.

TRANSPARENCIA REFERENCIAL

```haskell
square :: Int -> Int
square x = x * x
```

```haskell
let result1 = square 5
    result2 = square 5
putStrLn ("El cuadrado de 5 es " ++ show result1)
putStrLn ("El cuadrado de 5 es " ++ show result2)
```

Es una función cuyo retorno solo depende de los valores que reciba y no del contexto donde se use, es decir, si siempre se le da la misma información dará el mismo resultado.

POLIMORFISMO PARAMÉTRICO

```haskell
lengthOfList :: [a] -> Int
lengthOfList []     = 0
lengthOfList (_:xs) = 1 + lengthOfList xs
```

```haskell
let list1 = [1, 2, 3, 4, 5]
    list2 = ["Haskell", "es", "genial"]
putStrLn ("La longitud de la lista 1 es " ++ show (lengthOfList list1))
putStrLn ("La longitud de la lista 2 es " ++ show (lengthOfList list2))
```

Datos y funciones de tipo genérico (sin un tipo de dato asignado inicialmente) de forma que su implementación es maleable y puede recibir distintos parámetros.

Ejemplos:

```haskell
-- Función de primer orden
sumaDosNumeros :: Int -> Int -> Int
sumaDosNumeros x y = x + y
```

```haskell
-- Transparencia referencial
-- Podemos reemplazar `sumaDosNumeros 3 4` con su valor resultante `7` sin cambiar el comportamiento del programa.
valor :: Int
valor = sumaDosNumeros 3 4
```

```haskell
-- Polimorfismo paramétrico
-- La función `id` puede tomar un valor de cualquier tipo `a` y devolver un valor del mismo tipo.
id :: a -> a
id x = x
```