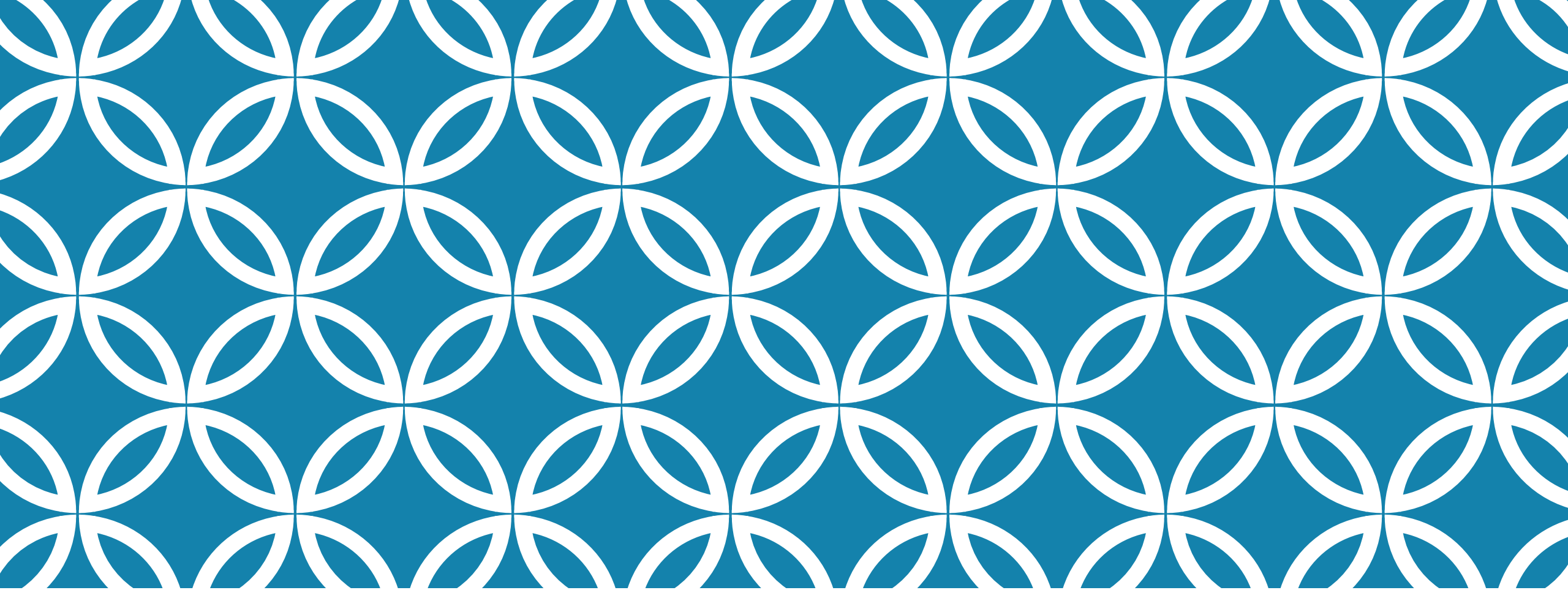

ESTRUCTURAS DISCRETAS

Javier Enríquez Mendoza
Mauricio E. Hernández Olvera

WRAPPERS

En programación se le conoce como wrapper a una expresión que tiene como propósito es envolver a otra.

Por ejemplo una función que llama a otra.

O un tipo de dato que se define a partir de otro.

```
Binario2 = Cero | Bin Binario1
```

LISTAS ... AGAIN

En Haskell las listas son indexadas

Esto quiere decir que podemos acceder al elemento n de una lista.

La forma de hacer esto es con la funcion **!!**.

Por ejemplo:

```
[0,1..] !! 100
100
```

PREÁMBULO

Como vimos en la primera clase, Haskell es un lenguaje de programación **puramente** funcional.

Es to quiere decir que en Haskell tenemos **transparencia referencial**.

Lo que podría verse como una limitante en ciertos aspectos.

Por ejemplo, en la siguiente función:

```
car :: [a] -> a
car [] = error “No hay elementos”
car (x:xs) = x
```

Estamos lanzando un error, que no es necesariamente un error.

MAYBE

Hay casos de algunas funciones en donde no es necesario regresar nada, sin embargo la transparencia referencial nos obliga a hacerlo.

La forma en la que le damos la vuelta a esto es con un tipo de dato **Maybe.** Definido de la siguiente manera:

```
Maybe a = Nothing | Just a

car :: [a] -> Maybe  a
car [] = Nothing
car (x:xs) = Just x
```

La forma de acceder al elemento de un Just es con la función **fromJust.**

DO

Una forma de simular en Haskell un comportamiento imperativo.

Es una recibe un bloque de código y ejecuta cada línea (comportamiento imperativo).

Es importante notar que esto lo hace sin perder la pureza del lenguaje.

Sintaxis:

```
do
    action1
    action2
    ...
    actionN
```

IO

Cuando se esta diseñando un programa es importante la interacción con el usuario final.

Sin embargo esto solo puede lograrse con un comportamiento imperativo, pues conlleva efectos secundarios.

En Haskell se separan las funciones puras, de aquellas en las que pueden existir efectos secundarios, definiendo estos efectos secundarios como valores de un tipo en particular.

A este tipo se le llama IO.

FUNCIONES PARA INTERACTUAR CON EL USUARIO

Para poder usar IO tenemos muchas funciones que nos permiten interactuar con el usuario de nuestro programa.

- **putStrLn** que sirve para imprimir una cadena en la consola.
  - putStrLn "Hello World"
- **getLine** espera que el usuario escriba algo y lo interpreta como una cadena.
- **<-** la asignación de todas las entradas al programa se hacen con este operador.
  - entrada <- getLine

Pero en Haskell como lo hemos visto hasta ahora no tiene sentido utilizar ninguna de estas funciones.

Las usamos dentro de sentencias **do** o en la función **main**.

MAIN

main es una función sin parámetros que regresa algo de tipo IO.

El cuerpo de la función **main** es lo que se va a ejecutar cuando se inicie el programa.

Al compilar un programa en Haskell, **GHC** busca la función main y crea un archivo ejecutable.

Al ejecutar este archivo el comportamiento del programa es el que se definió en la función main.