Introducción al Paradigma Funcional

Hernan Quintana (hquintan@ulima.edu.pe)

Definición

- A diferencia de Paradigma Imperativo, en vez de dar secuencias de instrucciones, lo que se hace es **definir** el comportamiento del program.
- Qué es lo que hace? VS Cómo lo hace?

Por qué Haskell?

- Es un lenguaje funcional **puro**.
- Es ocioso (*lazy*).
- Tiene **tipado estático**, pero lo suficientemente inteligente para tener **tipado implícito**.
- Compilado e interactivo.

Qué necesitamos para empezar?

- Entorno de desarrollo instalado y configurado.
- Dos formas de trabajo:
- Ejecución de programa compilado.
- Ejecución interactiva (REPL)

Formas de trabajo interactiva

1. Crear las definiciones que vamos a utilizar en un archivo .hs

```
sumar n1 n2 = n1 + n2
```

2. Abrir una terminal desde VSCode (CTRL + ` o CTRL + ñ) y escribir comando **ghci**.
3. Cargar definiciones con comando de ghci:

```
ghci> :l definiciones
```

definiciones es el nombre del archivo .hs

Formas de trabajo interactiva (cont.)

4. Desde la terminal ghci, utilizar la función cargada.

```
ghci> sumar 3 3
```

- **Nota 1:** En caso que se modifique el archivo de definiciones (.hs), debe recargarlo (**:r**) o volver a poner **:l definiciones**.
- **Nota 2:** Para salir del ghci, teclear CTRL + d

Forma de trabajo compilada

```haskell
main = do
    let doubleMe x = x + 1
    putStrLn (show (doubleMe 7))
```

```
$ ghc compilada.hs
```

Expressions vs Declarations

Expresión (*Expression*)

- Porción de código que representa un valor
- Puede ser el resultado de la evaluación de una función.

```
sumar 3 3
```

```
1 + 1
```

```
"Hola"
```

Declaración

- Es la manera como relacionamos un nombre con un valor.

```
nombre = "Pepe"
numero = 23
```

- Podemos combinar declaraciones con expresiones.

```
resultado = sumar 3 4
```

Funciones

Definición

- Representación de las funciones matemáticas:

f(X) -> Y

- X: Dominio. Conjunto de posible valores que la función puede recibir. **argumentos/parámetros de entrada**.
- Y: Rango. Posible valores que la función puede devolver.
- f: Nombre la función.

Declaración de funciones

```
sumar n1 n2 = n1 + n2
```

- **sumar**: nombre de la función
- **n1, n2**: argumentos de entrada
- **parte luego del =** : cuerpo de la función.

Ejecución (Evaluación)

```
sumar 3 4
```

- **Nota:** La ejecución de funciones son expresiones.

```
sumar 3 sumar 1 1 ????
```

Precedencia

- Hay contextos en los cuales Haskell no sabrá el orden de ejecución (evaluación) de las funciones.
- Para estos casos, se podrá utilizar paréntesis.

```
sumar 3 (sumar 1 1)
```

La función if

- En lenguajes funciones, no hay estructuras de control (if, switch, etc).
- Existen **expresiones** que nos dan un comportamiento similar, como la función if.

Ej: Implementar una función que reciba un número y lo multiplique por 2 en caso que sea mayor que 100, y caso contrario, lo divida entre 2.

```haskell
multiplicarDividirNumero n = if n > 1000
                             then n * 2
                             else n / 2
```

if es una expresión por lo que retorna un valor.

- Qué pasaría si al resultado del if quisiera incrementarle uno?

```haskell
multiplicarDividirNumero n = (if n > 1000
                              then n * 2
                              else n / 2) + 1
```

Valores

- Los valores de haskell pueden ser los siguientes:
- enteros, caracteres, cadenas de caracteres, booleanos, listas, etc.
- Haskell tiene **tipado estático** por lo que podemos definir nombres que solo acepten determinados tipos de valor.

Tipos de datos

- Un tipo nos dice qué categorías de valores podrían ser relacionados con determinada etiqueta o nombre.

Tipo	Ejemplos
Int	num = 10
Float / Double	numDec = 10.5
Char	letra = 'a'
String	nombre = "Pepe"
Bool	res = True o res = False

Declaración

Implícita

```
edad = 10
```

Explícita

```
edad :: Int = 10
```

Cómo declarar funciones de forma explícita?

Ej: Implementar una función llamada mayorEdad que reciba como argumentos de entrada un nombre (String) y una edad (Int) y que responda True en caso que la edad sea mayor o igual a 18 años, y False en los otros casos.

```haskell
mayorEdad :: String -> Int -> Bool
mayorEdad nombre edad =
    if
        (edad >= 18)
    then
        True
    else
        False
```