Curso: Programación Lógica y Funcional

Unidad 1: Conceptos básicos

Sesión 4: Definición de Funciones, Reglas de Bifurcación y Tipos de Datos

Docente: Carlos R. P. Tovar

Dudas de la anterior sesión

INICIO
Objetivo de la sesión

- Al finalizar la sesión el estudiante será capaz de definir funciones en lenguajes declarativos, aplicar reglas de bifurcación (binaria y múltiple) y reconocer distintos tipos de datos en programación declarativa.

UTILIDAD

Aprender estos temas es importante porque:

- La definición de funciones es la base de la programación funcional.
- Las reglas de bifurcación permiten tomar decisiones en los programas.
- Conocer los tipos de datos ayuda a escribir programas más correctos y expresivos.
- Fortalece la capacidad de modelar problemas reales de forma clara y eficiente.

Definición de Funciones en Haskell

- Una función recibe argumentos y devuelve un resultado.
- Ejemplo:

```
-- Función que suma dos números
suma x y = x + y

-- Función recursiva (factorial)
factorial 0 = 1
factorial n = n * factorial (n-1)
```

Estructura básica en Haskell:

```
nombreFuncion :: TipoParam -> TipoRetorno
nombreFuncion parametro = expresion
```

- **Ejemplo:**

```
doble :: Int -> Int
doble x = x * 2

-- Uso:
-- doble 5 → 10
```

Funciones con múltiples parámetros (Currying):

```
suma :: Int -> Int -> Int
suma a b = a + b

-- Aplicación parcial
suma5 :: Int -> Int
suma5 = suma 5

-- suma5 3 → 8
```

Reglas de Bifurcación, Bifurcación Binaria (if-then-else)

```
mayorEdad edad = if edad >= 18 then "Adulto" else "Menor“
```

```
absoluto :: Int -> Int
absoluto n = if n >= 0 then n else -n
```

Reglas de Bifurcación, Bifurcación Múltiple (case)

```
clasificarNota nota
  | nota < 11  = "Desaprobado"
  | nota < 15  = "Regular"
  | nota < 18  = "Bueno"
  | otherwise  = "Excelente“
```

Reglas de Bifurcación. Bifurcación Múltiple (case)

```
-- Usando case
mensaje x = case x of
  1 -> "Uno"
  2 -> "Dos"
  _ -> "Otro número"
```

Reglas de Bifurcación. Bifurcación Múltiple (guards)

```
absoluto' :: Int -> Int
absoluto' n
   | n >= 0    = n
   | otherwise = -n
```

Reglas de Bifurcación. Pattern Matching (evaluación por casos):

```
factorial :: Int -> Int
factorial 0 = 1  -- Caso base
factorial n = n * factorial (n - 1)
```

- **Ejemplo con listas:**

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

Tipos de Datos en Haskell

- **Básicos:** Int, Float, Char, Bool, String
- **Listas:** [1,2,3], [True, False, True]
- **Tuplas:** ("Juan", 20)
- **Definidos por el usuario:**

data Dia = Lunes | Martes | Miercoles | Jueves | Viernes

PRACTICA
Ejercicios para desarrollar en clase

- Definir una función que calcule el cuadrado de un número.
- Escribir una función que determine si un número es par o impar.
- Crear una función que clasifique la edad de una persona en niño, joven, adulto y anciano.
- Definir un tipo de dato para representar figuras geométricas.
- Resolver ejemplos usando bifurcaciones múltiples con guards.

CIERRE
Conclusiones

- La definición de funciones permite reutilizar y estructurar el código.
- Las reglas de bifurcación (binaria y múltiple) facilitan la toma de decisiones.
- Los tipos de datos son fundamentales para modelar información en los programas.
- Estos conceptos son esenciales para continuar desarrollando programas más complejos en Haskell y otros lenguajes declarativos.

Universidad
Tecnológica
del Perú