Curso: Programación Lógica y Funcional

Unidad 2: Programación funcional

Sesión 5: Tipos de datos

Docente: Carlos R. P. Tovar

Universidad Tecnológica del Perú

Dudas de la anterior sesión

INICIO
Objetivo de la sesión

- Comprender el sistema de tipos de Haskell
- Crear tipos de datos personalizados
- Utilizar tipos algebraicos de datos (ADTs)
- Implementar pattern matching con tipos personalizados
- Aplicar tipos en ejercicios prácticos

TRANSFORMACIÓN
Sistema de Tipos de Haskell

Características principales:

- Fuertemente tipado
- Inferencia de tipos
- Tipos algebraicos
- Polimorfismo paramétrico
- Clases de tipos

```
-- Ejemplo de inferencia
doble x = x * 2  -- Haskell infiere: doble :: Num a => a -> a
```

Tipos Básicos en Haskell

```
-- Tipos numéricos
entero :: Int
entero = 42

flotante :: Float
flotante = 3.14

doble :: Double
doble = 2.71828
```

```
-- Booleanos y caracteres
verdadero :: Bool
verdadero = True

caracter :: Char
caracter = 'A'

cadena :: String
cadena = "Hola Haskell"
```

Creación de Tipos Personalizados
Sinónimos de tipos (type):

```
type Nombre = String
type Edad = Int
type Altura = Float

type Persona = (Nombre, Edad, Altura)

-- Uso:
personaEjemplo :: Persona
personejemplo = ("Juan", 25, 1.75)
```

Tipos Algebraicos de Datos (ADTs)

Tipos suma:

```
data Figura = Circulo Float
            | Rectangulo Float Float
            | Triangulo Float Float
Float
```

Tipos producto:

```
data Persona = Persona String
Int Float


-- Uso:
persona :: Persona
persona = Persona "María" 30
1.65
```

Pattern Matching con Tipos Personalizados

```
area :: Figura -> Float
area (Circulo r) = pi * r * r
area (Rectangulo b h) = b * h
area (Triangulo b h _) = (b * h) / 2

-- Ejemplo:
circuloGrande :: Figura
circuloGrande = Circulo 10.0

areaCirculo = area circuloGrande  -- 314.15927
```

Tipos con Parámetros (Genéricos)

```
data Maybe a = Nothing | Just a

data Either a b = Left a | Right b

-- Ejemplos:
edadValida :: Maybe Int
edadValida = Just 25

edadInvalida :: Maybe Int
edadInvalida = Nothing

resultadoDiv :: Either String Float
resultadoDiv = Right 15.5

errorDiv :: Either String Float
errorDiv = Left "División por cero"
```

Registros (Records)

```
data Persona = Persona {
    nombre :: String,
    edad :: Int,
    altura :: Float,
    activo :: Bool
} deriving (Show)
```

```
-- Creación:
personaEjemplo :: Persona
personaEjemplo = Persona {
    nombre = "Carlos",
    edad = 28,
    altura = 1.78,
    activo = True
}

-- Acceso:
nombrePersona = nombre
personaEjemplo  -- "Carlos"
```

Derivación Automática

```
data Color = Rojo | Verde | Azul
   deriving (Show, Eq, Enum, Bounded)

-- Ejemplos:
show Rojo    -- "Rojo"
Rojo == Rojo -- True
[minBound .. maxBound] :: [Color]  -- [Rojo, Verde, Azul]
```

Ejercicio Práctico: Sistema de Geometría

```
data Punto = Punto Float Float
data Figura = Circulo Punto Float
            | Rectangulo Punto Float Float

area :: Figura -> Float
area (Circulo _ r) = pi * r * r
area (Rectangulo _ b h) = b * h

-- Uso:
miCirculo = Circulo (Punto 0 0) 5.0
miRectangulo = Rectangulo (Punto 1 1) 4.0 6.0
```

Ejercicio: Sistema de Usuarios

```
data TipoUsuario = Administrador | UsuarioNormal | Invitado

data Usuario = Usuario {
    username :: String,
    tipo :: TipoUsuario,
    email :: Maybe String,
    edad :: Int
}
```

```
esAdministrador :: Usuario -> Bool
esAdministrador usuario = tipo usuario == Administrador

puedeEditar :: Usuario -> Bool
puedeEditar usuario = case tipo usuario of
    Administrador -> True
    UsuarioNormal -> True
    Invitado -> False
```

Patrones de Diseño con Tipos - Smart constructors:

```
data Email = Email String

crearEmail :: String -> Maybe Email
crearEmail str
  | '@' `elem` str = Just (Email str)
  | otherwise = Nothing
```

Patrones de Diseño con Tipos - Tipos fantasmas (Phantom types):

```
data Validado
data NoValidado

data Formulario a = Formulario {
  nombre :: String,
  email :: String
}

validar :: Formulario NoValidado -> Maybe (Formulario Validado)
validar form
  | length (nombre form) > 2 && '@' `elem` email form =
    Just (Formulario (nombre form) (email form))
  | otherwise = Nothing
```

PRACTICA
Tarea para la Próxima Sesión

1. Crear un tipo ListaEnlazada que pueda ser vacía o contener un elemento y otra lista
2. Implementar funciones para: longitud, agregar elemento y concatenar listas
3. Crear un tipo ÁrbolBinario con operaciones básicas
4. Resolver problemas usando pattern matching con tipos customizados

Recursos:

- Haskell Data Types
- Learn You a Haskell - Types

CIERRE
Conclusiones

¿Cuándo usar type vs data?

¿Qué ventajas ofrecen los tipos algebraicos?

¿Cómo ayuda el sistema de tipos a prevenir errores?

¿Qué son las clases de tipos y cómo se relacionan?

Cierre

-- Los tipos son como documentación ejecutable

-- que nos ayuda a escribir código más seguro

-- y expresivo

¡Los tipos no son clases, pero las clases sí son tipos! 🎓

Universidad
Tecnológica
del Perú