Clase 1

1. Modelos de Computación: La Distinción Fundamental

El curso contrasta dos formas de programar. Entender la diferencia es crucial:

- **Modelo Destructivo (Imperativo):**
    - Se basa en **comandos que producen efectos** y modifican un estado.
    - Un programa es una secuencia de pasos que transforma un estado inicial en uno final.
    - Ejemplo: Los procedimientos en Gobstones que mueven el cabezal o ponen bolitas cambian el estado del tablero.
- **Modelo Denotacional (Funcional):**
    - Se basa en **evaluar expresiones que describen valores**.
    - Un programa es en sí mismo una gran expresión, y su ejecución consiste en encontrar su valor final.
    - **No hay estado ni memoria mutable**.
    - El lenguaje utilizado para este modelo en la materia es **Haskell**.

---

2. Fundamentos de Haskell

Haskell es la herramienta para explorar el modelo funcional. Un programa es un conjunto de definiciones de funciones y constantes.

Sintaxis y Características Clave:

- **Llamada a funciones:** Los argumentos se separan por espacios, sin paréntesis ni comas (ej: sumar 2 3).
- **Agrupación:** Los paréntesis () solo se usan para agrupar expresiones y alterar el orden de evaluación (ej: sumar 1 (mult 2 3)).
- **Definición de tipos:** Se usa :: que se lee como "tiene tipo" (ej: dos :: Int).
- **Condicionales (if-then-else):**
    - Son expresiones, lo que significa que **siempre deben devolver un valor**.
    - Por esta razón, la rama else es **obligatoria**.
- **Errores explícitos:** Se puede usar la función error "mensaje" para detener la ejecución en casos no deseados, similar a boom en Gobstones.
- **Tipos de datos básicos:** Int (números), Bool (True, False), Char ('a'), y String ("hola").
    - **Importante:** Un String es simplemente una lista de caracteres [Char].

---

3. Tipos de Datos Algebraicos (TDA)

Son tipos de datos definidos por el programador usando la palabra clave data. La definición consiste en enumerar todas las formas posibles de construir un valor de ese tipo.

- **Tipos Enumerativos:**
    - Son los más simples. Se listan una serie de "constructores" sin argumentos que representan valores únicos.
    - Ejemplo: data Dir = Norte | Este | Sur | Oeste.
- **Registros:**
    - Tienen un único constructor que agrupa varios valores (campos), usualmente de distintos tipos.
    - Ejemplo: data Persona = P String Int String. Aquí, P es el constructor.
- **Sumas (Variantes con Argumentos):**
    - Combinan los dos anteriores. Hay múltiples constructores, y algunos (o todos) pueden tener argumentos.
    - Ejemplo: data Helado = Vasito Gusto | Cucurucho Gusto Gusto.

---

4. Pattern Matching: El Mecanismo Clave para Usar TDA

Es la forma de "desarmar" los tipos algebraicos para acceder a la información que contienen.

- **¿Cómo funciona?** Se usan los constructores en los parámetros de una función. Haskell verifica las ecuaciones en orden hasta que una coincide con el valor de entrada.
- **Uso para observar:** nombre (P n e d) = n. Aquí, n, e, y d son variables que toman los valores de los campos del registro Persona.
- **El orden de las ecuaciones es crucial.** Una regla general (esEsteMal d = False) antes que una específica (esEsteMal Este = True) impedirá que la regla específica se ejecute.
- **Comodín (_):** Se usa cuando no necesitas el valor de un parámetro.
- **Expresión case:** Es una alternativa al pattern matching en los parámetros.
    - Sintaxis: case d of { Norte -> Este; Sur -> Oeste; ... }.

---

5. Polimorfismo y Tuplas

- **Tuplas:**
    - Son como registros predefinidos pero sin nombres de campo.
    - Se escriben entre paréntesis: (3, "hola", True).
    - El tipo también se escribe así: (Int, String, Bool).
    - Se pueden desarmar con pattern matching: fst (x, y) = x.
- **Polimorfismo Paramétrico:**

- Permite crear funciones y tipos de datos genéricos que operan sobre cualquier tipo.
- Se usan **variables de tipo** (letras minúsculas como a, b) en las firmas de tipo.
- Ejemplo: fst :: (a, b) -> a. Esta función funciona para cualquier tupla, sin importar los tipos de sus componentes.
- **Limitación clave:** Una función polimórfica no puede usar operaciones específicas de un tipo (como + que es solo para números) sobre un parámetro de tipo genérico a. Si lo hace, deja de ser polimórfica.

---

6. Listas en Haskell

Son una estructura de datos predefinida y polimórfica ([a], una lista de "algún tipo a").

- **Constructores Fundamentales:**
  1. []: La lista vacía (también llamada "nil").
  2. (:): El operador "cons" que agrega un elemento al **principio** de una lista.
- **Punto Clave para el Examen:**
  - La notación [1, 2, 3] es solo una abreviatura cómoda de 1 : (2 : (3 : [])).
  - El operador de concatenación (++) **NO es un constructor**. Por lo tanto, **NO se puede usar en pattern matching**. Intentar hacer f (xs ++ ys) = ... es un error.
  - Se puede hacer pattern matching para casos comunes:
    - head (x:_) = x (extrae el primer elemento).
    - tail (_:xs) = xs (obtiene el resto de la lista).
    - esSingular (_:[]) = True (verifica si la lista tiene un único elemento).