Clase 2

1. Constructores de Listas y Errores Comunes

- **Constructores Fundamentales:** Las listas en Haskell tienen solo dos constructores:
    - [] (se lee "nil" o lista vacía).
    - : (se lee "cons" o agregar al principio).
- **Notación:** La forma [1, 2, 3] es solo una abreviatura de 1:(2:(3:[])).
- ¡Error Crítico! ++ no es un constructor: La función para concatenar o agregar listas (++) **no es un constructor**. Por lo tanto, **no puedes** usarla para *pattern matching*. Hacer algo como f (xs ++ ys) = ... es un error de construcción muy común.

2. Pattern Matching (Descomposición de Listas)

- El *pattern matching* funciona **únicamente** con los constructores (: y []) para descomponer una lista.
- **Ejemplos de patrones válidos:**
    - (x:xs): Separa la lista en su cabeza (x) y su cola (xs).
    - (_:[]): Identifica una lista con un solo elemento.
    - (_:x:_): Accede al segundo elemento de una lista de al menos dos elementos.
- **Funciones Parciales (head y tail):** Estas funciones fallarán si se les pasa una lista vacía ([]). Es más seguro usar *pattern matching* para evitar errores en tiempo de ejecución.

3. El Principio de la Recursión Estructural

La recursión estructural es la forma principal de recorrer listas en Haskell, ya que no existen bucles como while o foreach.

- **Estructura Fija:** Una función recursiva sobre listas siempre tiene dos casos:
    1. **Caso Base:** ¿Qué hacer cuando la lista es vacía ([])?.
    2. **Caso Recursivo:** ¿Qué hacer con una lista no vacía (x:xs)?.
- **Punto Clave del Caso Recursivo:** El truco está en asumir que ya tienes resuelto el problema para la cola de la lista (xs) y solo debes pensar en cómo incorporar la cabeza (x) a ese resultado.
    - **Ejemplo sumarTodos:** Si sumarTodos ns ya te da la suma de la cola, para la lista completa (n:ns) solo necesitas hacer n + sumarTodos ns.

4. Tipos de Recursión y Casos Especiales

- **Recursión Estructural Pura:** Sigue exactamente la estructura de reemplazo de constructores: [] se reemplaza por un valor y : por una operación. (Ej: sumarTodos, longitud).
- **Recursivas No Estrictamente Estructurales:** Algunas funciones necesitan un pequeño ajuste.
    - **Ejemplo last:** El resultado no depende del primer elemento, sino del final de la recursión. Esto requiere un caso base adicional para listas de un solo elemento (x:[]) para saber cuándo detenerse.

- **Funciones NO Recursivas (que usan auxiliares):** Ciertas funciones no pueden calcularse con un solo recorrido recursivo porque el resultado de la cola no tiene suficiente información.
    - **Ejemplo promedio:** El promedio de [20, 30] (que es 25) no te ayuda a calcular el promedio de [10, 20, 30]. En estos casos, se resuelve combinando otras funciones que sí son recursivas, como promedio ns = div (sumarTodos ns) (longitud ns).

5. Recursión sobre Números

- El principio es análogo al de las listas.
- **Caso Base:** 0.
- **Caso Recursivo:** Se define la función para n en términos de la misma función aplicada a n-1.
- **Aplicación:** Generalmente se usa para números naturales (no negativos).

6. Recursiones Anidadas

- Ocurre cuando una función recursiva opera sobre más de una estructura de datos (ej: dos listas o un número y una lista).
- **Ejemplo zip (x:xs) (y:ys):** Se hace *pattern matching* en ambas listas. La recursión avanza simultáneamente sobre las dos colas: (x,y) : zip xs ys.
- **Ejemplo losPrimerosN n (x:xs):** Se hace *pattern matching* sobre la lista y se reduce el número en la llamada recursiva: x : losPrimerosN (n-1) xs.