Materia:	1S7B-PROGR.LOG.FUNC
Nombre Informe:	Uso de algunos caracteres y funciones en haskell
Alumno(s):	Pizano Zamora Leonardo Gabriel

Objetivo

El objetivo de esta investigación es comprender el uso y aplicación de los siguientes elementos en Haskell:

El keyword "otherwise".
El carácter de subrayado "_" (underscore).
La función words().
La creación y manipulación de listas.

Marco Teórico

Haskell es un lenguaje de programación funcional que se caracteriza por su sintaxis concisa y su manejo eficiente de funciones y listas. A continuación, se detallan los elementos específicos que serán investigados:

Keyword "otherwise": Es una palabra clave que se utiliza en las guardas de las expresiones condicionales. Actúa como un caso por defecto en las evaluaciones de guardas.

Carácter "_" (underscore): En Haskell, el underscore se utiliza como un comodín que indica que un valor no es importante y no será utilizado. Es común en patrones y en expresiones lambda.

Función words: Es una función predefinida en Haskell que toma una cadena de texto y la divide en una lista de palabras, utilizando espacios en blanco como delimitadores.

Listas en Haskell: Las listas son una estructura de datos fundamental en Haskell y permiten almacenar secuencias ordenadas de elementos. Las listas pueden ser manipuladas mediante diversas funciones predefinidas.

Desarrollo

0) Uso del keyword "otherwise"

En Haskell, "otherwise" se utiliza en expresiones condicionales con guardas para representar el caso por defecto. Es simplemente una constante que equivale a True y facilita la legibilidad del código.

Ejemplo:

haskell

```
fact :: Int -> Int
fact n
  | n == 0           = 1
  | otherwise = n * fact (n - 1)
```

En este ejemplo, la función fact calcula el factorial de un número. Si n es 0, devuelve 1. En cualquier otro caso (indicado por otherwise), calcula el producto de n y el factorial de n-1.
1) Uso de "_" (underscore)

El underscore se utiliza para ignorar un valor en patrones, evitando así la necesidad de nombrar variables que no serán utilizadas.

Ejemplo 1:

haskell

```
fst' :: (a, b) -> a
fst' (x, _) = x
```

En esta función, se descompone una tupla, pero se ignora el segundo elemento.

Ejemplo 2:

haskell

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

Aquí, se ignora el primer elemento de la lista y se continúa el cálculo de la longitud con el resto de la lista.
2) Uso de la función words

La función words toma una cadena de texto y la divide en una lista de palabras, utilizando espacios en blanco como delimitadores.

Ejemplo:

haskell

```
example :: String -> [String]
example str = words str
```

haskell

```
main = print (example "Hola mundo en Haskell")
```

Salida: ["Hola","mundo","en","Haskell"]
3) Creación y manipulación de listas

Las listas en Haskell se crean utilizando corchetes y se pueden manipular de diversas formas.

Ejemplo de creación:

haskell

```
lista :: [Int]
lista = [1, 2, 3, 4, 5]
```

Manipulación de listas:

haskell

```
-- Concatenación de listas
listaConcatenada = [1, 2, 3] ++ [4, 5]

-- Agregar un elemento al inicio de la lista
listaExtendida = 0 : lista

-- Acceder al primer elemento (head)
primerElemento = head lista

-- Acceder al resto de la lista (tail)
restoLista = tail lista
```

Resultado

El uso de "otherwise", "_", words y la manipulación de listas en Haskell se han demostrado mediante ejemplos prácticos y comprensibles. Cada uno de estos elementos facilita la escritura de código más limpio y eficiente en Haskell.

Evaluación

Cada uno de los elementos investigados cumple un propósito específico y esencial en la programación funcional con Haskell. El uso adecuado de estas herramientas permite escribir programas más claros y concisos. Los ejemplos presentados muestran cómo aplicar estos conceptos en situaciones comunes.

Conclusiones}

El keyword "otherwise" simplifica las guardas en expresiones condicionales. El carácter "_" permite ignorar valores innecesarios en patrones. La función words facilita la división de cadenas en palabras. La creación y manipulación de listas es una habilidad fundamental en Haskell que se puede lograr con una variedad de funciones integradas. El conocimiento y aplicación de estos elementos son cruciales para escribir código efectivo en Haskell.