

Parcial 1 - Algoritmos I Taller: Tema H

Ejercicio 1

En las siguientes preguntas marque la respuesta correcta.

a. Si tengo una función con la siguiente signatura (definición):

```
f :: a -> (a -> b) -> b
```

puedo decir que:

1. Es una función polimórfica *ad-hoc*
2. Es un constructor
3. Es una función polimórfica paramétrica
4. Es una función parcial
5. Ninguna de las anteriores

b. *Currificación* me permite:

1. Definir funciones que reciban un tipo cualquiera no básico
2. Definir funciones parciales con parámetros fijos
3. Definir funciones que en algunos casos devuelvan `Nothing`
4. Es una técnica para definir tipos recursivos

c. Si tengo una función con la siguiente declaración:

Unset

```
f :: Int -> Bool
f n | n /= 42 = True
    | otherwise = False
```

puedo decir que:

1. La definición de la función usa *pattern-matching*
2. La definición de la función utiliza análisis por casos
3. Es una función recursiva
4. *a* y *b* son correctas

d. El mensaje de error:

Unset

```
Non-exhaustive patterns in function f
```

me indica que:

1. La función `f` no está definida usando *pattern-matching*
2. La función `f` no es recursiva
3. No es un mensaje de error de *Haskell*
4. La función `f` no está definida para todos los casos del *Dominio* de `f`

Ejercicio 2

Se van a representar los *Objetos Celestes* estudiados en la Astronomía con tipos en *Haskell*. Los objetos celestes que queremos tener en cuenta son los siguientes: **Estrella** y **Planeta**. La idea es poder detallar de cada objeto celeste sus características más importantes. Por tal motivo, identificaremos las siguientes características de cada uno de los objetos celestes:

Estrella

- **Luminosidad**, que es un tipo enumerado con las siguientes opciones: **Supergigante**, **Gigante**, **SecuenciaPrincipal**, **Enana**.
- **Temperatura**, tipo enumerado bajo la clasificación *Morgan–Keenan* con las opciones: **A**, **B**, **F**, **G**, **K**, **M** y **O**.
- **Nombre**, que es un sinónimo de **String** indicando el nombre designado a la estrella.

Planeta

- **Estructura**, que es un tipo enumerado con las opciones: **Rocoso** y **Gaseoso**.
- **NumSat**, que es un sinónimo de **Int** indicando la cantidad de satélites naturales que posee el planeta.
- **DistEstrella**, que es un sinónimo de **Float** indicando la distancia del planeta a la estrella que orbita.

Para ello:

- Definir el tipo **Astro** que consta de los constructores **Estrella** y **Planeta**, constructores con los parámetros descritos arriba (se deben definir también los tipos enumerados **Luminosidad**, **Temperatura**, **Estructura**, y los tipos sinónimos **NumSat** y **DistEstrella**). Los tipos **Astro**, **Estrella**, **Luminosidad** y **Temperatura**, **no deben** estar en la clase **Eq** ni **Ord**. Agregue la clase **Show** en los tipos que considere necesarios.
- Definir la función **masSatelites** dada por:

Unset

```
masSatelites :: [Astro] -> Int -> [Astro]
```

que dada una lista de **Astros** **as** y un valor **n** de Cantidad, devuelve la lista de **Astro** que son **Planeta** en **as** y que tienen tantos o más satélites que **n**.

Nota: Dejar como comentario en el código dos o tres ejemplos con los que probaste la función.

- Definir el orden para **Astro** de manera que:
 - Una **Estrella** es mayor que otra, si su **Luminosidad** y **Temperatura** son mayores, respectivamente. El orden de las temperaturas es el siguiente: **O < B < A < F < G < K < M**.

El orden de las luminosidades es el siguiente: Supergigante > Gigante > SecuenciaPrincipal > Enana.

- Un Planeta es mayor que otro si tiene más cantidad de satélites naturales.

Nota: Dejar como comentario en el código dos o tres ejemplos con los que probaste la igualdad.

Ejercicio 3

Queremos hacer un programa, para que las profesoras de una academia de Inglés puedan saber si sus alumnos de un nivel pueden pasar al siguiente o no.

a) Definir un tipo recursivo `NotasDeIngles`, que permite guardar las notas que tuvo cada estudiante de un nivel en el período. El tipo `NotasDeIngles`, tendrá dos constructores:

1) `EvolucionDelEstudiante`, que tiene 4 parámetros:

- `String`, para el nombre y apellido del alumno
- `Nivel` (Tipo Enumerado con el Nivel actual que está cursando: Uno, Dos, Tres)
- `Int` (con la nota del primer parcial, entre 1 y 10)
- `Int` (con la nota del segundo parcial, entre 1 y 10)
- `Int` (con la nota del final 1 a 10,)
- `NotasDeIngles`, recursión con el resto de las notas.

2) `NoHayMasEstudiantes`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que se terminaron las notas.

La condición para poder obtener el siguiente nivel se describen a continuación según las notas obtenidas en las diferentes instancias:

- Si el estudiante está en Nivel One o Two, debe sacar más de 8 en alguno de los parciales, y haber tenido en el final al menos un 7.
- Si el estudiante está en el Nivel Three debe tener al menos un 6 en los dos parciales, y al menos un 7 en el final.

b) Programar la función `pasaDeNivel`, que toma como primer parámetro `notas` del tipo `NotasDeIngles`, y como segundo parámetro el `nombre` del estudiante de tipo `String` y retorna un valor de tipo `Bool`, indicando si el estudiante con ese `nombre` es **pasa de nivel o no**.

```
pasaDeNivel :: NotasDeIngles -> String -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado `pasaDeNivel` con un parámetro de tipo `NotasDeIngles` que tenga al menos 3 estudiantes.

c) Programar la función `devolverNotaFinal` con la siguiente declaración:

```
devolverNotaFinal :: NotasDeIngles -> String -> Maybe Int
```

que toma una variable `notas` de tipo `NotasDeIngles`, y como segundo argumento un `nombre`, que identifica al estudiante, y en caso que este esté en `notas` (co una nota en el final `i`), retorna `Just i` y `Nothing` en caso contrario.

NOTA: Dejar como comentario un ejemplo donde hayas probado la función.