Funcional 1: Primeros ejercicios de funcional

Notas previas

En algunos ejercicios se van a utilizar algunas de las funciones que están en el Prelude por ej:

```
mod 20 3 = 2  -- el resto de la división entre 20 y 3 es 2.
div 14 3 = 4  -- parte entera de la división entre 14 y 3 es 4.
max 8 10 = 10 -- devuelve el max entre 2 números.
min 9 15 = 9  -- devuelve el min entre 2 números.
```

Ahora sí, los ejercicios

Básicos

1.1. Definir la función **esMultiploDeTres/**1, que devuelve True si un número es múltiplo de 3, p.ej:

```
Main> esMultiploDeTres 9
True
```

1.2. Definir la función **esMultiploDe/**2, que devuelve True si el segundo es múltiplo del primero, p.ej.

```
Main> esMultiplo 12 3
 True
```

1.3. Definir la función **cubo/**1, devuelve el cubo de un número.

1.4. Definir la función **area/**2, devuelve el área de un rectángulo a partir de su base y su altura.

1.5. Definir la función **esBisiesto/**1, indica si un año es bisiesto. (Un año es bisiesto si es divisible por 400 o es divisible por 4 pero no es divisible por 100)

Nota: Resolverlo reutilizando la función **esMultiploDe/2**

1.6. Definir la función **celsiusToFahr/**1, pasa una temperatura en grados Celsius a grados Fahrenheit.

1.7. Definir la función **fahrToCelsius/**1, la inversa de la anterior.

1.8. Definir la función **haceFrioF/**1, indica si una temperatura expresada en grados Fahrenheit es fría. Decimos que hace frío si la temperatura es menor a 8 grados Celsius.

1.9. Definir la función **mcm**/2 que devuelva el mínimo común múltiplo entre dos números,

de acuerdo a esta fórmula.

m.c.m.(a, b) = {a * b} / {m.c.d.(a, b)}

Para más información: buscar "Mínimo común múltiplo" en la wikipedia en castellano (es.wikipedia.org).

Nota: Se puede utilizar gcd.

1.10. Trabajamos con tres números que imaginamos como el nivel del río Paraná a la altura de Corrientes medido en tres días consecutivos; cada medición es un entero que representa una cantidad de cm.

P.ej. medí los días 1, 2 y 3, las mediciones son: 322 cm, 283 cm, y 294 cm.

A partir de estos tres números, podemos obtener algunas conclusiones.

Definir estas funciones:

1.10. a. dispersion, que toma los tres valores y devuelve la diferencia entre el más alto y el más bajo. Ayuda: extender max y min a tres argumentos, usando las versiones de dos elementos. De esa forma se puede definir dispersión sin escribir ninguna guarda (las guardas están en max y min, que estamos usando).

1.10. b. diasParejos, **diasLocos** y **diasNormales** reciben los valores de los tres días. Se dice que son días parejos si la dispersión es chica, que son días locos si la dispersión es grande, y que son días normales si no son ni parejos ni locos. Una dispersión se considera chica si es de menos de 30 cm, y grande si es de más de un metro.

Nota: Definir **diasNormales** a partir de las otras dos, no volver a hacer las cuentas.

1.11. En una plantación de pinos, de cada árbol se conoce la altura expresada en cm. El peso de un pino se puede calcular a partir de la altura así: 3 kg x cm hasta 3 metros, 2 kg x cm arriba de los 3 metros. P.ej. 2 metros -> 600 kg, 5 metros -> 1300 kg.
Los pinos se usan para llevarlos a una fábrica de muebles, a la que le sirven árboles de entre 400 y 1000 kilos, un pino fuera de este rango no le sirve a la fábrica.

Para esta situación:

- Definir la función **pesoPino**, recibe la altura de un pino y devuelve su peso.
- Definir la función **esPesoUtil**, recibe un peso en kg y devuelve True si un pino de ese peso le sirve a la fábrica, y False en caso contrario.
- Definir la función **sirvePino**, recibe la altura de un pino y devuelve True si un pino de ese peso le sirve a la fábrica, y False en caso contrario. Usar composición en la definición.

1.12. Desafío Café con Leche:
Implementar la función **esCuadradoPerfecto**/1, sin hacer operaciones con punto flotante. Ayuda: les va a venir bien una función auxiliar, tal vez de dos parámetros. Pensar que el primer cuadrado perfecto es 0, para llegar al 2do (1) sumo 1, para llegar al 3ro (4) sumo 3, para llegar al siguiente (9) sumo 5, después sumo 7, 9, 11 etc..
También algo de recursividad van a tener que usar.

Aplicación Parcial

Nota previa

Hay aplicación parcial cuando a una función la evaluamos con menor cantidad de argumentos con la que esta definida.

Veamos un Ejemplo:

La función (*) recibe dos enteros como argumento y devuelve el resultado de la multiplicación.

(*)::Int->Int->Int
Si queremos aplicar la función (*) con un solo argumento nos quedaría:

(2*) Esto nos devuelve una función que es (Int->Int), recibe un entero y devuelve otro entero.

Si esta última función la aplico sobre un entero, nos devuelve otro entero como resultado Ej:

```
Hugs> (2*) 9
18
```

Ejercicios

2.1. Definir una función **siguiente**, que al invocarla con un número cualquiera me devuelve el resultado de sumar a ese número el 1.

```
Main> siguiente 3
4
```

2.2. Definir la función **mitad** que al invocarla con un número cualquiera me devuelve la mitad de dicho número, ej:

```
Main> mitad 5
2.5
```

2.3. Definir una función **inversa,** que invocando a la función con un número cualquiera me devuelva su inversa.

```
Main> inversa 4
0.25

Main> inversa 0.5
2.0
```

2.4. Definir una función **triple,** que invocando a la función con un número cualquiera me devuelva el triple del mismo.

```
Main> triple 5
15
```

2.5. Definir una función **esNumeroPositivo,** que invocando a la función con un número cualquiera me devuelva true si el número es positivo y false en caso contrario.

```
Main> esNumeroPositivo (-5)
False

Main> esNumeroPositivo 0.99
True
```

Composición

Nota previa

Se base en el concepto matemático de composición de funciones.

$$g(f(x)) = (g \circ f)\ x$$

La imagen de f(x) tiene que coincidir con el dominio de g(x)

Acá la notación es (g . f) x . Veamos un Ejemplo:

```
g n =  even n
f n =  3 + n

Main> (g . f) 4
False

Main> (g . f) 3
True
```

Ejercicios

3.1. Resolver la función del ejercicio 1.2 de esta guía **esMultiploDe/**2, utilizando aplicación parcial y composición

3.2. Resolver la función del ejercicio 1.5 de esta guía **esBisiesto/**1, utilizando aplicación parcial y composición

3.3. Resolver la función **inversaRaizCuadrada/**1, que da un número n devolver la inversa su raíz cuadrada.

```
Main> inversaRaizCuadrada 4
0.5
```

Nota: Resolverlo utilizando la función **inversa** Ej. 2.3**, sqrt** y composición.

3.4. Definir una función **incrementMCuadradoN**, que invocándola con 2 números m y n, incrementa un valor m al cuadrado de n por Ej:

```
Main> incrementMCuadradoN 3 2
11
```

Incrementa 2 al cuadrado de 3, da como resultado 11.

Nota: Resolverlo utilizando aplicación parcial y composición.

3.5. Definir una función **esResultadoPar**/2, que invocándola con número n y otro m, devuelve true si el resultado de elevar n a m es par.

```
Main> esResultadoPar 2 5
True

Main> esResultadoPar 3 2
False
```

Nota: Resolverlo utilizando aplicación parcial y composición.

Tuplas

Nota previa

Una tupla es un tipo de dato compuesto, que contiene una cantidad fija de elementos y se pueden combinar distintos tipos de elementos, cada elemento está en una posición fija de la estructura.

Ej: Un alumno viene dado por una tupla de 2 elementos, que esta formada por el (legajo, promedio).

alumno = (“1104433-8”, 9).

Existen algunas funciones predefinidas para obtener cada uno de los elementos de la tupla.

Por Ej. si quiero obtener el legajo del alumno puedo hacer lo siguiente:

```
Main> fst alumno
"1104433-8"
```

Y si quiero obtener el promedio:

```
Main> snd alumno
9
```

Otro Ejemplo:

fst(4,6) = 4 -- devuelve el primer elemento de la tupla.
snd(5,7) = 7 -- devuelve el segundo elemento de la tupla.

Ejercicios

4.1. Definir las funciones **fst3**, **snd3**, **trd3**, que dada una tupla de 3 elementos devuelva el elemento correspondiente, p.ej.

```
Main> snd3(4,5,6)
5
```

```
Main> trd3(4,5,6)
6
```

4.2. Definir la función **aplicar,** que recibe como argumento una tupla de 2 elementos con funciones y un entero, me devuelve como resultado una tupla con el resultado de aplicar el elemento a cada una de la funciones, ej:

```
Main> aplicar (doble,triple) 8
(16,24)

Main> aplicar ((3+),(2*)) 8
(11,16)
```

4.3. Definir la función **cuentaBizarra,** que recibe un par y: si el primer elemento es mayor al segundo devuelve la suma, si el segundo le lleva más de 10 al primero devuelve la resta 2do – 1ro, y si el segundo es más grande que el 1ro pero no llega a llevarle 10, devuelve el producto. Ej

```
Main> cuentaBizarra (5,8)
40
```

```
Main> cuentaBizarra (8,5)
13
```

```
Main> cuentaBizarra (5,29)
24
```

4.4. Representamos las notas que se sacó un alumno en dos parciales mediante un par (nota1,nota2), p.ej. un patito en el 1ro y un 7 en el 2do se representan mediante el par (2,7).
A partir de esto

- Definir la función **esNotaBochazo**, recibe un número y devuelve True si no llega a 4, False en caso contrario. No vale usar guardas.
- Definir la función **aprobo**, recibe un par e indica si una persona que se sacó esas notas aprueba. Usar **esNotaBochazo**.
- Definir la función **promociono**, que indica si promocionó, para eso tiene las dos notas tienen que sumar al menos 14 y además haberse sacado 6 en cada parcial.
- Escribir una consulta que dado un par indica si aprobó el primer parcial, usando **esNotaBochazo** y composición. La consulta tiene que tener esta forma (p.ej. para el par de notas (5,8))

  ```
  Main> (... algo ...) (5,8)
  ```

4.5. Siguiendo con el dominio del ejercicio anterior, tenemos ahora dos parciales con dos recuperatorios, lo representamos mediante un par de pares ((parc1,parc2),(recup1,recup2)).
Si una persona no rindió un recuperatorio, entonces ponemos un "-1" en el lugar correspondiente.
Observamos que con la codificación elegida, siempre la mejor nota es el máximo entre nota del parcial y nota del recuperatorio.
Considerar que vale recuperar para promocionar.
En este ejercicio vale usar las funciones que se definieron para el anterior.

- Definir la función notasFinales que recibe un par de pares y devuelve el par que corresponde a las notas finales del alumno para el 1er y el 2do parcial.
  P.ej.

  ```
  Main> notasFinales ((2,7),(6,-1))
  ```

(6,7)
Main> notasFinales ((2,2),(6,2))
(6,2)
Main> notasFinales ((8,7),(-1,-1))
(8,7)

Escribir la consulta que indica si un alumno cuyas notas son `((2,7),(6,-1))` recursa o no. O sea, la respuesta debe ser True si recursa, y False si no recursa.
Usar las funciones definidas en este punto y el anterior, y composición.
La consulta debe tener esta forma
`Main> (... algo ...) ((2,7),(6,-1))`

Escribir la consulta que indica si un alumno cuyas notas son `((2,7),(6,-1))` recuperó el primer parcial. Usar composición. La consulta debe tener esta forma
`Main> (... algo ...) ((2,7),(6,-1))`

Definir la función **recuperoDeGusto** que dado el par de pares que representa a un alumno, devuelve True si el alumno, pudiendo promocionar con los parciales (o sea sin recup.), igual rindió al menos un recup.
Vale definir funciones auxiliares.
Hacer una definición que no use pattern matching, en las eventuales funciones auxiliares tampoco; o sea, manejarse siempre con fst y snd.

4.6. Definir la función **esMayorDeEdad**, que dada una tupla de 2 elementos (persona, edad) me devuelva True si es mayor de 21 años y False en caso contrario. Por Ej:.

Main> esMayorDeEdad(juan,18)
False

Nota: Definir la función utilizando aplicación parcial y composición.

4.7. Definir la función **calcular,** que recibe una tupla de 2 elementos, si el primer elemento es par lo duplica, sino lo deja como esta y con el segundo elemento en caso de ser impar le suma 1 y si no deja esté último como esta.

Main> calcular (4,5)
(8,6)

Main> calcular (3,7)
(3,8)

Nota: Resolverlo utilizando aplicación parcial y composición.