Práctico 9 - Paradigma Funcional

Utilizaremos el lenguaje Haskell *(y Python cuando se requiera un ejemplo de código imperativo* para la resolución de los ejercicios a continuación.

Pasos para instalar uno de los compiladores de Haskell, para Windows:
1. Descargar e instalar $GlasgowHaskellCompiler/Platform$
2. Descargar e instalar $WinGHCI$
3. Configurar Visual Studio Code para Haskell, o bien utilizar algún Editor de Texto preferido, o bien descargar e instalar *Scite* para editor y configurarle el editor al WinGHCI en file options, pegando link al disco `C:\Program Files (x86)\SciTE\SciTE.exe`

Nota: En este práctico no utilice en lo posible, listas por comprensión

1. Abrir WinGHCi y ejecutar en el *prelude* o terminal, una a una las siguientes líneas, observar los resultados de cada una. Este es el único ejercico que hará completo directamente escribiendo los comandos en la terminal.

```
5 / 3
14 `div` 4
div 14 4
2 == 2
2 > 3
2 <= 3
2 /= 3
round 6.7
sqrt 2
pi
not (2 == 3)
True && False
True && True
True || False
False || False
:type 'a'
'a' < 'b'
'a' == 'A'
'a' < 'A'
:type (<)
```

   A partir de los próximos ejercicios, todos deberán ser escritos en un archivo .hs que luego cargará y compilará, para ejecutar.

2. Usando un Editor de Texto, crear un archivo conversiones.hs y definir las siguientes funciones piesAmetros, metrosApies, millasAKms, kmAMillas, pulgadasACM, cmApulgadas. Cada función recibe un parámetro y retorna el valor apropiado. Luego cargar el archivo en WinGHCi y testear cada función en la consola o prelude.

3. Crear una variable llamada n y asignarle un valor numérico, por ejemplo 5.
   Luego, asignarle otro valor, por ejemplo 6. ¿Qué sucedió? Explique porqué.

4. ¿Que entiende por estado de un programa? ¿El Paradigma Funcional tiene estados?

5. Crear una función que sume dos parámetros numéricos.

6. Crear una función que calcule la distancia entre dos puntos (x_1, y_1) y (x_2, y_2).

7. Crear una función que calcule la hipotenusa dadas las longitudes de los otros dos lados del tríangulo.

8. Definir una función que determine si un número es positivo o no.

9. Crear una función que si el número recibido es menor a 100, lo duplique, sino lo retorne sin modificación alguna.

10. Crear una función que devuelva el mayor de dos números.

11. Definir la función *signo* la cual viene dada por

$$signo(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ 0 & \text{si} \quad x = 0 \\ -1 & \text{si} \quad x < 0 \end{cases}$$

12. Crear la función factorial. Recordar que la misma está definida matemáticamente como:
$0! = 1$
$n! = n(n-1)...1$
Comparar con una versión de la misma, realizada en paradigma imperativo, sin usar recursión, en Python.

13. Crear una función que tome un parámetro numérico, llamado n y calcule la serie de Fibonacci hasta ese valor n. Recordar que la serie de Fibonacci está dada por:
$f_0 = 0$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$

14. Crear una función *mult* que tome dos parámetros y retorne la multiplicación de ambos, pero definida de modo recursivo y sin usar producto *.

15. Te es dada la siguiente función $sumarUno\ n = n + 1$. Sin utilizar otra función que no sea esa, y en particular sin utilizar el +, definir una función *suma* que tome dos parámetros y defina la suma de ambos, recursivamente.

Listas

16. Definir una función, sin usar la función *length* predefinida en Haskell, que devuelva la cantidad de elementos de una lista. Sin utilizar listas por comprensión.

17. Definir una función sumarImpares que tome una lista como argumento y retorne la suma de todos los valores impares de la misma.

18. Crear una función que retorne la suma de todos los elementos de una lista de valores numéricos, sin utilizar la función *sum*.

19. Crear una función que invierta los valores de una lista. Ejemplo, si recibe $[2, 3, 7]$ retornará $[7, 3, 2]$ sin utilizar *reverse*

20. Definir una función que retorne el último elemento de una lista. Realizar dos versiones, una utilizando las funciones para listas que ya vienen predefinidas, y otra versión sin utilizarlas.

21. Definir una función que retorne la cantidad de letras *c* que haya en una frase. Extienda c para que sea cualquier letra puntual que se desee contar, no solamente el carácter 'c'.

Variado

22. Crear una función llamada *repetir* que tome la cantidad de veces que debe repetir la letra a. *Ejemplo: si recibe un 3, retornará 'aaa'. Si recibe un 8, retornará 'aaaaaaaa'.* Crear una segunda versión de esta función que tome la letra en cuestión a repetir.

23. Definir una función *primera* que dado un nombre, devuelva la primer letra del mismo.

24. Definir una función que duplique el primer elemento de una lista. Sin utilizar recursividad ni listas por comprensión.

25. Definir una función llamada *in* que permita determinar si un elemento está en una lista, ambos pasados como parámetros.