Práctico 10 - Paradigma Funcional

Listas por Comprensión

1. Generar una lista de los primeros 10 números pares positivos.
2. Crear una lista de los cuadrados de los números del 1 al 10.
3. Generar una lista de los números del 1 al 100 que son divisibles por 3.
4. Crear una lista de los primeros 10 números impares.
5. Crear una lista de los primeros 10 números naturales, pero elevados a la tercera potencia.
6. Dada una lista de números, obtener una lista con el cuadrado de cada número.
7. Crea una lista de los números pares en el rango de 1 a 50, pero excluyendo aquellos que son divisibles por 4.
8. Crear una lista por comprensión que produzca todos los números del 1 al 100, que sean múltiplos de 2, 3 y 5.
9. Dada una lista de números, obtener una lista con los números mayores a 10.
10. Dada una lista de números, obtener una lista con los números impares y mayores a 5.
11. Crear una lista por comprensión que produzca todos los pares de números entre 1 y 20, cuya suma sea menor o igual a 20.
12. Dada una lista de palabras, obtener una lista con la longitud de cada palabra.
13. Definir la función len que retorne la cantidad de elementos de una lista, esta vez sin usar recursión, y utilizando listas por comprensión como parte de la resolución.
14. Generar una lista de las vocales extraidas de una cadena de caracteres *(texto)*.
15. Definir una función *quitarMinusculas* que dada una frase, retorne una frase con solo las letras mayúsculas que haya en la misma.
16. Definir una versión propia de la función *zip* que genera una nueva lista, con tuplas cuyo primer elemento pertenece a la primer lista, y cuyo segundo elemento pertenece a la segunda lista. Ejemplo: mizip [1,2,3] [9,9,9] retornaría [(1,9), (2, 9), (3, 9)]

Funciones de Orden Superior

En los siguientes ejercicios, debe utilizar map y/o filter para su resolución, según corresponda, a menos que se indique lo contrario. En lo posible, realizar dos versiones, una creando la función *f* que map o filter necesitarán de modo tradicional, y otro, siempre que sea posible, definiendo una función anónima.

17. Crea una función *doblarNumeros* que tome una lista de números y devuelva una lista con el doble de cada número.

18. Dada una lista de números, retornar una nueva lista con todos los números negativos extraidos de la lista original.

19. Dada una lista de edades, retornar una lista indicando mayoria de edad.

```
Ejemplo:
Prelude> mayoriaEdad [15,20,17]
["Menor","Mayor","Menor"]
```

20. Dada una lista de palabras, retornar una lista con dichas palabras invertidas.

```
Ejemplo:
Prelude> inversa ["hola","mundo"]
["aloh","odnum"]
```

21. Definir una función *filtrarPares* que tome una lista de enteros y devuelva una lista con solo los números pares.

22. Crear una función *calcularCuadrados* que tome una lista de números y devuelva una lista con el cuadrado de cada número.

23. Crear una función *calcularPromedio* que tome una lista de números y devuelva el promedio de esos números.

24. Crear una función *aplicarALista* que tome dos parámetros: una función y una lista, y retorne la lista de haberle aplicado dicha función a cada elemento. *(No puede utilizar map)*

25. La siguiente definición, reescribala para que en vez de la sección where utilice una función lambda.

```
map f xs where f x = x * 2 + 3
```

26. Repase el concepto del algoritmo de ordenación llamado *quicksort* e implementelo. Considere utilizar las funciones filter, map o funciones lambda.

Variado

27. Definir una función que remueva la primer letra de una palabra, y la ubique al final. Si la palabra que recibe es vacía, retornará una cadena vacía. Este ejercicio lo puede realizar sin necesidad de utilizar nada en particular. Resuélvalo como ud. desee.