LISTA DE EJERCICIOS 2

1. Define una función **quitaUno** que quite de una lista dada la primera aparición (si existe alguna) de un elemento dado. Ejemplo: quitaUno 5 [1,3,5,7,3,5,8] = [1,3,7,3,5,8]

2. Define una función **quitaRep** que dada una lista elimine los elementos repetidos.
Ejemplo: quitaRep [1,3,5,7,3,5,8] = [1,3,5,7,8]

3. Usando **quitaUno**, define una función **dif** que realice la diferencia de dos listas.
Ejemplo: dif [1,2,3,7,4,2,8,4] [3,5,2,4,2] = [1,7,8,4]

4. Usando **dif**, define una función **perm** que decida si una lista es permutación de otra.
Ejemplos: perm "abc" "acb" = True perm [1,2,2,4] [2,4,1] = False

5. Usando **perm** y **filter**, define una función **sonpermde1** que, dada una lista de listas **xss** , obtenga la sublista de **xss** formada por aquellos elementos que son permutaciones de la cabeza de **xss**.
Ejemplo: sonpermde1 ["abc", "ac", "bca", "bbc", "cab"] = ["abc", "bca", "cab"]

6. Usando alguna versión de **fold**, define una función **aDecimal** que convierta una lista de dígitos en el número decimal que representa. Define su función inversa **aDigitos**.
Ejemplos: aDecimal [9,6,3,8] = 9638 aDigitos 9638 = [9,6,3,8]

7. Define las funciones **decimalAbinario** y **binarioAdecimal** para transformar un número decimal a binario y viceversa.
Ejemplos: decimalAbinario 13 = 1101 binarioAdecimal 1101 = 13

8. Define un predicado que decida si una lista está ordenada.
Ejemplos: ordenada "ayu" = False ordenada [1,3,7,12] = True

9. Usando **takeWhile** y **dropWhile**, define la función **palabras** que dada una frase (string) obtenga la lista de palabras (lista de strings) de la frase.
Ejemplo: palabras " Estoy en el laboratorio" = ["Estoy", "en", "el", "laboratorio"]

10. Define la función **posiciones** que devuelva las posiciones de un elemento dado en una lista dada. Ejemplo: posiciones 8 [8,9,5,8,2] = [0,3]

11. Define una función **paraTodo** que dado un predicado y una lista decida si todos sus elementos satisfacen el predicado, y una función **existe** que dado un predicado y una lista decida si alguno de sus elementos satisface el predicado.
Compáralas con las predefinidas: **all**, **any** :: (α -> Bool) -> [α] -> Bool

12. Define una función **permutar** que devuelva todas las permutaciones de una lista dada. Ejemplo: permutar [2,8,3] = [[2,8,3],[8,2,3],[8,3,2],[2,3,8],[3,2,8],[3,8,2]]

13. Una lista es una **sublista** de otra si los elementos de la primera aparecen en la segunda en el mismo orden. Una lista es una **subsecuencia** de otra si aparece en ella como secuencia de elementos contiguos. Define las funciones sublista y subsecuencia.
Ejemplos: sublista “palma” “tapa la mesa” = True
subsecuencia “palma” “tapa la mesa” = False

14. Evalúa la siguiente expresión, descrita mediante una lista intensional:
[j | k <– [1,-1, 2, -2], k>0, j <– [1.. k]]

15. Da definiciones alternativas a las funciones de los ejercicios 10, 11 y 12 utilizando listas intensionales.

16. Dado un string, define una función **diag** que devuelva cada carácter del string en una línea y en diagonal. Ejemplo:

```
? diag “haskell”
 h
  a
   s
    k
     e
      l
       l
```

17. Define una función **repLong** que dado un string devuelva tantas copias del string como su longitud y una en cada línea. Ejemplo:

```
? repLong “hugs”
 hugs
 hugs
 hugs
 hugs
```

18. Prueba inductivamente las siguientes propiedades:

1. length(s1 ++ s2)= length s1 + length s2 (s1, s2 listas finitas)
2. length (zip s1 s2) = min (length s1) (length s2) (s1, s2 listas finitas)
3. take n s ++ drop n s = s (s lista finita, n núm. natural)
4. take m . drop n = drop n . take (n+m) (m,n números naturales)
5. init = reverse . tail . reverse
6. map (f . g) = (map f) . (map g)
7. filter p . concat = concat . map (filter p)
8. filter p . map f = map f . filter (p . f)

19. Sintetiza una definición recursiva de **concat** a partir de la siguiente:

```
concat = foldr (++) []
```