MODELOS DE PARCIAL

1) Definir una función que, para una lista cualquiera y otra lista de enteros, devuelva una lista con los elementos de la primera cuyas posiciones aparecen indicadas en la segunda, en el orden en que ésta indique (numerando desde 1).

Ejemplo: damePosicionesLista ([5,6,8,10], [2,2,1,4]) -> [6,6,5,10]

```
damePosicionesLista :: ([a],[Int]) -> [a]
damePosicionesLista ([],y) = []
damePosicionesLista (x,[]) = []
damePosicionesLista (x,(y:ys)) = enesimoElemento (x, y) : damePosicionesLista (x, ys)

enesimoElemento :: ([a], Int) -> a
enesimoElemento (x:y, 1) = x
enesimoElemento (x:y, n) = enesimoElemento(y, n-1)
```

2) Definir una función que, dados los enteros n, d y h, devuelva una matriz de n columnas cuyas sucesivas filas tengan los números desde d hasta h (todos ellos iguales por fila).

Ejemplo: dameMatriz (2, 3, 6) -> [[3,3],[4,4],[5,5],[6,6]]

```
dameMatriz :: (Int, Int, Int) -> [[Int]]
dameMatriz (0, d, h) = []
dameMatriz (n, d, h) = if d <= h then dameFila (n, d) : dameMatriz (n, d+1, h) else []

dameFila :: (Int, Int) -> [Int]
dameFila (0, m) = []
dameFila (n, m) = m : dameFila(n-1,m)
```

3) Definir una función que dadas dos matrices de cualquier tamaño y contenido y dado un entero n, asumiendo que ambas matrices tienen igual cantidad de filas, devuelva la matriz que resulta de insertar la 2da matriz en la 1er matriz luego de la columna enésima.

Ejemplo: combinarMat ([[1,2,3],[4,5,6]], [[8,9],[9,7]], 2) -> [[1,2,8,9,3],[4,5,9,7,6]] -}

```
combinarMat :: ([[a]], [[a]], Int) -> [[a]]
combinarMat (x, [], n) = x
combinarMat ([], y, n) = y
combinarMat ((x:xs), (y:ys), n) = combinarFilas (x, y, n) : combinarMat (xs, ys, n)

combinarFilas :: ([a], [a], Int) -> [a]
combinarFilas (x, y, 0) = append (y, x)
combinarFilas ((x:xs), y, n) = x : combinarFilas (xs, y, n-1)
```

4) Definir una función que, dada una lista de enteros, determine sus dos valores mayores y devuelva ambos. (Para listas de menos de dos elementos, interpretar y decidir qué conviene hacer).

Ejemplos:
dameDosMayores [1,3,0,4,2,1] -> (4,3)
dameDosMayores [2,3,5,1,1,5] -> (5,5)

```
dameDosMayores :: [Int] -> [Int]
dameDosMayores [] = []
dameDosMayores (x:[]) = [x]
dameDosMayores x = [maximoLista x, maximoLista (borrarElementoLista (x, maximoLista x))]

maximoLista :: [Int] -> Int
maximoLista (x:[]) = x
maximoLista (x:(y:z)) = if x < y then maximoLista (y:z) else maximoLista (x:z)

borrarElementoLista :: ([Int], Int) -> [Int]
borrarElementoLista ([], y) = []
borrarElementoLista ((x:y), n) = if x == n then y else x : borrarElementoLista (y, n)
```

5) Definir una función que, dada una matriz de cualquier tamaño y contenido, y un elemento más del mismo tipo que los elementos de dicha matriz, devuelva una matriz como la anterior, pero con ese elemento adicional al final de cada fila.

Ejemplo: agregarAMatriz ([[1,2,3],[4,5,6]], 2) -> [[1,2,3,2],[4,5,6,2]]

```
agregarAMatriz :: ([[a]], a) -> [[a]]
agregarAMatriz ([], n) = []
agregarAMatriz ((x:xs), y) = agregarAlFinalLista (x, y) : agregarAMatriz (xs, y)

agregarAlFinalLista :: ([a], a) -> [a]
agregarAlFinalLista ([], y) = [y]
agregarAlFinalLista ((x:xs), y) = x : agregarAlFinalLista (xs, y)
```

6) Definir una función que, dada una matriz de enteros, devuelva la cantidad de columnas que tengan todos valores iguales.

Ejemplo: columnasIguales ([[6,6,4,6,1,1,1], [6,1,4,1,5,5,1]]) -> 3

```
columnasIguales :: [[Int]] -> Int
columnasIguales ([]:y) = 0
columnasIguales x = if todosIguales(cabezaListas x) then 1 + columnasIguales (colaListas x) else columnasIguales (colaListas x)

todosIguales :: [Int] -> Bool
todosIguales (x:[]) = True
todosIguales (x:y) = if x == cabeza y then todosIguales y else False
```

7) Dada una lista y dos enteros d y n, devolver el borrado de n elementos a partir del elemento d de la lista.

Ejemplo: borrarDeLista ([1,2,3,4,5,6], 3, 2) -> [1,2,5,6]

```
borrarDeLista :: ([a], Int, Int) -> [a]
borrarDeLista ([], d, n) = []
borrarDeLista (x, 1, n) = borrarNElementosLista (x, n)
borrarDeLista (x:y, d, n) = x : borrarDeLista (y, d-1, n)

borrarNElementosLista :: ([a], Int) -> [a]
borrarNElementosLista ([],y) = []
borrarNElementosLista (x,0) = x
borrarNElementosLista (x:y, n) = borrarNElementosLista (y, n-1)
```

8) Escribir una función que, dados un entero n y una matriz de enteros de cualquier tamaño, determine cuantos valores iguales a n hay dentro de dicha matriz.

Ejemplo: cuantosIgualesA (52, [[4,2,3,52], [52,52,1,20]]) -> 3

```
cuantosIgualesA :: (Int, [[Int]]) -> Int
cuantosIgualesA (x, []) = 0
cuantosIgualesA (n, x:y) = cuantas (n, x) + cuantosIgualesA (n, y)

cuantas :: (Int, [Int]) -> Int
cuantas (n, []) = 0
cuantas (n, (x:y)) = if n == x then 1 + cuantas (n, y) else cuantas (n, y)
```

9) Definir una función que, dada una matriz de enteros, devuelva dos listas: una con los totales (sumas) por fila y otra con los totales (sumas) por columna.

Ejemplo: sumarFilasyColumnasMat [[1,2,3], [4,5,6], [7,8,9]] -> ([6,15,24], [12,15,18])

```
sumarFilasyColumnasMat :: [[Int]] -> ([Int],[Int])
sumarFilasyColumnasMat [] = ([],[])
sumarFilasyColumnasMat x = (sumarFilasMat x, sumarColumnasMat x)

sumarFilasMat :: [[Int]] -> [Int]
sumarFilasMat [] = []
sumarFilasMat (x:y) = suma x : sumarFilasMat y

sumarColumnasMat :: [[Int]] -> [Int]
sumarColumnasMat [] = []
sumarColumnasMat ([]:y) = []
sumarColumnasMat x = suma (cabezaListas x) : sumarColumnasMat (colaListas x)

suma :: [Int] -> Int
suma [] = 0
suma (x:y) = x + suma y

cabezaListas :: [[a]] -> [a]
cabezaListas [] = []
cabezaListas (x:y) = cabeza x : cabezaListas y

colaListas :: [[a]] -> [[a]]
colaListas [] = []
colaListas (x:y) = cola x : colaListas y

cabeza :: [a] -> a
cabeza (x:y) = x

cola :: [a] -> [a]
cola (x:y) = y
```

10) Definir una función que, dado un string, determine si es palíndromo (se leen igual de ambos lados) o no.

```
esPalindromo :: [Char] -> Bool
esPalindromo x = x == rev x

rev :: [a] -> [a]
rev [] = []
rev (x:y) = append (rev y, x:[])

append :: ([a], [a]) -> [a]
append ([], z) = z
append ((x:y), z) = x : append (y, z)
```

11) Definir una función que, dado un string, devuelva la lista de las vocales que preceden inmediatamente a consonantes. Ignorar acentos y mayúsculas.

Ejemplo: vocalesPrecedenConsonantes "seis aeroplanos ahora" -> "ieoaoao"

```
vocalesPrecedenConsonantes :: [Char] -> [Char]
vocalesPrecedenConsonantes [] = []
vocalesPrecedenConsonantes (x:[]) = []
vocalesPrecedenConsonantes (x:(y:z)) = if (esVocal x && not(esVocal y) && y /= ' ') then x : vocalesPrecedenConsonantes(y:z)
                                       else vocalesPrecedenConsonantes(y:z)

esVocal :: Char -> Bool
esVocal x = if (x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u') then True else False
```

12) Definir una función que, dada una matriz de enteros, de cualquier tamaño, determine la cantidad de columnas que no tienen números impares.

Ejemplo: cantColumnasNoImpares [[7,2,3,5], [4,6,8,5]] -> 1

```
cantColumnasNoImpares :: [[Int]] -> Int
cantColumnasNoImpares ([]:y) = 0
cantColumnasNoImpares x = if hayImpar(cabezaListas x) then cantColumnasNoImpares(colaListas x)
                                                      else 1 + cantColumnasNoImpares(colaListas x)

hayImpar :: [Int] -> Bool
hayImpar [] = False
hayImpar (x:y) = if impar x then True else hayImpar y

impar :: Int -> Bool
impar 0 = False
impar 1 = True
impar x = impar (x - 2)
```

13) Definir una función que, dada una lista de cadenas de caracteres, devuelva aquellas cadenas que contengan en algún lado tres caracteres seguidos que no sean vocales. Ignorar acentos y diferencias entre mayúsculas y minúsculas.

Ejemplo:
masDeTresNoVocalesSeguidasMat ["java", "c++", "haskell", "basic", "python", "php", "html"] -> ["c++", "python", "php", "html"]

```
masDeTresNoVocalesSeguidasMat :: [[Char]] -> [[Char]]
masDeTresNoVocalesSeguidasMat [] = []
masDeTresNoVocalesSeguidasMat (x:y) = if masDeTresNoVocalesSeguidas x then x : masDeTresNoVocalesSeguidasMat y
                                                                       else masDeTresNoVocalesSeguidasMat y

masDeTresNoVocalesSeguidas :: [Char] -> Bool
masDeTresNoVocalesSeguidas [] = False
masDeTresNoVocalesSeguidas (x:(y:[])) = False
masDeTresNoVocalesSeguidas (x:(y:z)) = if not(esVocal x) && not(esVocal y) && not(esVocal (cabeza z)) then True
                                       else masDeTresNoVocalesSeguidas(y:z)
```

14) Definir una función que, dado un entero n y otro elemento e de cualquier tipo, devuelva una lista de n listas cada una de ellas con puras repeticiones del elemento e, entre 1 y n veces, en orden creciente de longitud.

Ejemplo: dameListasDeElemento(3,5) -> [[5],[5,5],[5,5,5]]

```
dameListasDeElemento :: (Int, a) -> [[a]]
dameListasDeElemento x = rev (dameListasDeElementoRev x)

dameListasDeElementoRev :: (Int, a) -> [[a]]
dameListasDeElementoRev (0, a) = []
dameListasDeElementoRev (n, e) = dameFilaElementoNVeces (n, e) : dameListasDeElementoRev (n-1, e)

dameFilaElementoNVeces :: (Int, a) -> [a]
dameFilaElementoNVeces (0, a) = []
dameFilaElementoNVeces (n, e) = e : dameFilaElementoNVeces (n-1, e)
```

15) Definir una función que, dada una matriz, devuelva una lista con el borde inferior de la matriz, es decir, una lectura de la matriz siguiendo una U en sentido antihorario, sin repetir los elementos de las dos esquinas.

Ejemplos:
dameBordesMat [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]] -> [1,6,11,12,13,14,15,10,5]
dameBordesMat [[1],[2],[3]] -> [1,2,3,2,1]
dameBordesMat [[1,2,3,4]] -> [1,2,3,4]

```
dameBordesMat :: [[a]] -> [a]
dameBordesMat [] = []
dameBordesMat x = flat [bordeIzquierdoMat x, bordeInferiorMat x, bordeDerechoMat x]

bordeIzquierdoMat :: [[a]] -> [a]
bordeIzquierdoMat (x:[]) = []
bordeIzquierdoMat (x:y) = cabeza x : bordeIzquierdoMat y

bordeDerechoMat :: [[a]] -> [a]
bordeDerechoMat (x:[]) = []
bordeDerechoMat (x:y) = append(bordeDerechoMat y, [ultimo x])

bordeInferiorMat :: [[a]] -> [a]
bordeInferiorMat x = ultimo x

flat :: [[a]] -> [a]
flat [] = []
flat (x:y) = append (x, flat y)

ultimo :: [a] -> a
ultimo x = cabeza (rev(x))
```

16) Dada una cantidad arbitraria de listas, devolver un par con las dos de menor longitud (en caso de empates resolver éstos de cualquier manera).

Ejemplo: lasDosListasDeMenorLongitud ["Javascript", "APL", "Basic", "Snobol", "C", "Java"] -> ("C", "APL")

```
lasDosListasDeMenorLongitud :: [[a]] -> ([a], [a])
lasDosListasDeMenorLongitud z = (laListaDeMenorLongitud z,
                                 laListaDeMenorLongitud(eliminarListaXLongitud(z, minimaLongitud z)))

laListaDeMenorLongitud :: [[a]] -> [a]
laListaDeMenorLongitud (x:[]) = x
laListaDeMenorLongitud (x:(y:z)) = if long x < long y then laListaDeMenorLongitud(x:z) else laListaDeMenorLongitud(y:z)

eliminarListaXLongitud :: ([[a]], Int) -> [[a]]
eliminarListaXLongitud ([], n) = []
eliminarListaXLongitud (x:y, n) = if long x == n then y else x : (eliminarListaXLongitud(y, n))

minimaLongitud :: [[a]] -> Int
minimaLongitud z = long (laListaDeMenorLongitud z)
```

17) Dada una lista de enteros, devolver una lista de listas con esos enteros repetidos tantas veces como indica el propio valor en cada caso.
Ejemplo: dameListasDeEnterosRep [3,2,0,1,4] -> [[3,3,3],[2,2],[1],[4,4,4,4]]

```
dameListasDeEnterosRep :: [Int] -> [[Int]]
dameListasDeEnterosRep [] = []
dameListasDeEnterosRep (x:y) = if x > 0 then dameElementoNVeces (x, x) : dameListasDeEnterosRep y
                                        else dameListasDeEnterosRep y

dameElementoNVeces :: (Int, Int) -> [Int]
dameElementoNVeces (e, 0) = []
dameElementoNVeces (e, n) = e : dameElementoNVeces(e, n-1)
```

18) Dada una cantidad arbitraria de cadenas de caracteres, devolver aquella que tenga la máxima cantidad de vocales. Cada vocal cuenta, aunque se repita en la cadena; si hubiera más de una cadena con esa condición, devolver la última de las que lo cumplan.

Ejemplo: cantMaxVocales ["APL", "C++", "Javascript", "Basic", "Snobol"] -> "Javascript"

```
cantMaxVocales :: [[Char]] -> [Char]
cantMaxVocales (x:[]) = x
cantMaxVocales (x:(y:z)) = if cantVocales x > cantVocales y then cantMaxVocales(x:z) else cantMaxVocales(y:z)

cantVocales :: [Char] -> Int
cantVocales [] = 0
cantVocales (x:y) = (if esVocal x then 1 else 0) + cantVocales y
```

19) Dada una lista de listas de tamaños arbitrarios y elementos de cualquier tipo, devolver una lista con los elementos que aparezcan en las posiciones pares de todas ellas seguidos de los elementos que aparezcan en las posiciones impares de todas ellas, siempre en el orden en que vienen.

Ejemplo: elementosPosicionesParesEImpares [[3,7,8,9], [8,0], [], [1,2,3,4,5]] -> [7,9,0,2,4,3,8,8,1,3,5]

```
elementosPosicionesParesEImpares :: [[a]] -> [a]
elementosPosicionesParesEImpares [] = []
elementosPosicionesParesEImpares x = flat [damePosPares (flat x), damePosImpares (flat x)]

damePosPares :: [a] -> [a]
damePosPares [] = []
damePosPares (x:[]) = []
damePosPares (x:(y:[])) = [y]
damePosPares (x:(y:z)) = y : damePosPares z

damePosImpares :: [a] -> [a]
damePosImpares [] = []
damePosImpares (x:[]) = [x]
damePosImpares (x:(y:[])) = [x]
damePosImpares (x:(y:z)) = x : damePosImpares z
```

20) Dada una cantidad arbitraria de listas, devolver una lista con el último carácter de cada una de las listas que tengan longitud mayor que el promedio de longitudes de todas.
Ejemplo: ej20 ["C", "Javascript", "Basic", "APL", "Haskell"] -> "tl"

```
ej20 :: [[a]] -> [a]
ej20 [] = []
ej20 x = listaMayorAlProm(x, promedioLong x)

listaMayorAlProm :: ([[a]], Float) -> [a]
listaMayorAlProm ([], p) = []
listaMayorAlProm (x:y, p) = if longF x > p then (ultimo x) : listaMayorAlProm (y, p)
                            else listaMayorAlProm (y, p)

promedioLong :: [[a]] -> Float
promedioLong [] = 0
promedioLong x = sumaLong x / longF x

sumaLong :: [[a]] -> Float
sumaLong [] = 0
sumaLong (x:y) = longF x + sumaLong y

longF :: [a] -> Float
longF [] = 0
longF (x:y) = 1 + longF y
```

21) Dada una matriz de cualquier tamaño y contenido, representadas por filas como es habitual, y dado un entero c, devolver la matriz que se forma desde la columna c en adelante.

Ejemplo: dameMatrizDesdeColumna ([[3,1,4,1,5],[9,2,6,5,3]], 4) -> [[1,5], [5,3]]

```
dameMatrizDesdeColumna :: ([[a]], Int) -> [[a]]
dameMatrizDesdeColumna ([], y) = []
dameMatrizDesdeColumna (x, 1) = x
dameMatrizDesdeColumna (x:y, c) = dameListaDesde(x, c) : dameMatrizDesdeColumna(y, c)

dameListaDesde :: ([a], Int) -> [a]
dameListaDesde (x, 1) = x
dameListaDesde ([], y) = []
dameListaDesde ((x:y), n) = dameListaDesde (y, n-1)
```

22) Dadas dos listas de listas de enteros, y un entero b, determinar si es cierto que la primera de ambas contiene más veces que la segunda al entero b.

Ejemplo: primerListaContieneMasVecesAlElmento ([[4,7,8,9],[8,0]], [[],[8,8,8]], 8) -> False

```
primerListaContieneMasVecesAlElmento :: ([[Int]], [[Int]], Int) -> Bool
primerListaContieneMasVecesAlElmento (x, y, n) = cuantas(n, flat x) > cuantas(n, flat y)
```

23) Definir una función que, dada una lista de cualquier tamaño y contenido, devuelva la misma sin el elemento central, o sin el primero de los dos centrales en caso de longitud par.

Ejemplos
ej23 [1,2,4,8,16,32] -> [1,2,8,16,32]
ej23 "DESPLEGAR" -> "DESPEGAR"
ej23 "MI" -> "I"

```
ej23 :: [a] -> [a]
ej23 [] = []
ej23 x = listaSinElmentoCentral (x, damePosicionAEliminar x)

damePosicionAEliminar :: [a] -> Int
damePosicionAEliminar x = (long x + 1) `div` 2

listaSinElmentoCentral :: ([a], Int) -> [a]
listaSinElmentoCentral (x:y, 1) = y
listaSinElmentoCentral (x:y, n) = x : listaSinElmentoCentral(y, n-1)
```

24) Definir una función que, dada una lista de cadenas de caracteres, indique cuantas vocales tiene la cadena que precede a la última de las que tienen alguna vocal.

Ejemplo: ej24 ["Haskell", "C++", "Basic", "Javascript", "Cobol", "php", "html", "bf"] -> 3

```
ej24 :: [[Char]] -> Int
ej24 [] = 0
ej24 x = cantVocales (dameAnteUltimoElemento (damePalabrasConVocales x))

damePalabrasConVocales :: [[Char]] -> [[Char]]
damePalabrasConVocales [] = []
damePalabrasConVocales (x:y) = if cantVocales x > 0 then x : damePalabrasConVocales y else damePalabrasConVocales y

dameAnteUltimoElemento :: [a] -> a
dameAnteUltimoElemento (x:(y:[])) = x
dameAnteUltimoElemento (x:y) = dameAnteUltimoElemento y
```

25) Definir una función que, dada una matriz, un entero f y un entero c, devuelva la matriz "rotada" f filas hacia arriba y c columnas hacia la izquierda.

Ejemplo: rotarMatriz ([[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]], 2, 3) -> [[14,15,11,12,13], [4,5,1,2,3], [9,10,6,7,8]]

```
rotarMatriz :: ([[a]], Int, Int) -> [[a]]
rotarMatriz ([], f, c) = []
rotarMatriz (matriz, f, c) = rotarColumnasMatriz(rotarFilasMatriz(matriz, f), c)

rotarColumnasMatriz :: ([[a]], Int) -> [[a]]
rotarColumnasMatriz ([], n) = []
rotarColumnasMatriz (x, 0) = x
rotarColumnasMatriz (x, n) = rotarColumnasMatriz(mover1eroAlFinalL x, n-1)

rotarFilasMatriz :: ([[a]], Int) -> [[a]]
rotarFilasMatriz ([], n) = []
rotarFilasMatriz (x, 0) = x
rotarFilasMatriz (x, n) = rotarFilasMatriz(mover1eroAlFinal x, n-1)

mover1eroAlFinalL :: [[a]] -> [[a]]
mover1eroAlFinalL [] = []
mover1eroAlFinalL (x:y) = mover1eroAlFinal x : mover1eroAlFinalL y

mover1eroAlFinal :: [a] -> [a]
mover1eroAlFinal [] = []
mover1eroAlFinal (x:y) = append(y, x:[])
```