1. Apresente uma definição recursiva da função (pré-definida) `enumFromTo :: Int -> Int -> [Int]` que constrói a lista dos números inteiros compreendidos entre dois limites.

   Por exemplo, `enumFromTo 1 5` corresponde à lista `[1,2,3,4,5]`

```
enumFromTo :: Int -> Int -> [Int]
enumFromTo m n
    | m > n      = []          -- caso de paragem , se m for maior que n, retornamos a lista vazia
    | otherwise = m : enumFromTo (m+1) n
```

2. Apresente uma definição recursiva da função (pré-definida) `enumFromThenTo :: Int -> Int -> Int -> [Int]` que constrói a lista dos números inteiros compreendidos entre dois limites e espaçados de um valor constante.

   Por exemplo, `enumFromThenTo 1 3 10` corresponde à lista `[1,3,5,7,9]`.

```
enumFromThento :: Int -> Int -> Int -> [Int]
enumFromThento x y z
    | x > z        = []        -- caso de paragem, se x for maior que z, retornamos uma lista vazia
    | otherwise = x : enumFromThenTo y ( y + (y - x)) z
```

3. Apresente uma definição recursiva da função (pré-definida) `(++) :: [a] -> [a] -> [a]` que concatena duas listas.

   Por exemplo, `(++) [1,2,3] [10,20,30]` corresponde à lista `[1,2,3,10,20,30]`.

```
(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys          -- caso de paragem, se a primeira lista está vazia o resultado è sempre a segunda lista
(x:xs) ++ ys = x : (xs ++ ys)
```

4. Apresente uma definição recursiva da função (pré-definida) `(!!) :: [a] -> Int -> a` que dada uma lista e um inteiro, calcula o elemento da lista que se encontra nessa posição (assume-se que o primeiro elemento se encontra na posição 0).

   Por exemplo, `(!!) [10,20,30] 1` corresponde a 20.

   Ignore os casos em que a função não se encontra definida (i.e., em que a posição fornecida não corresponde a nenhuma posição válida da lista).

```
(!!) :: [a] -> Int -> a
(x:xs) !! n
   | n == 0
   | otherwise = xs !! (n-1)    -- aqui reduzimos n atè ao caso de paragem e obtemos assim o elemento pretendido
```

5. Apresente uma definição recursiva da função (pré-definida) `reverse :: [a] -> [a]` que dada uma lista calcula uma lista com os elementos dessa lista pela ordem inversa.

   Por exemplo, `reverse [10,20,30]` corresponde a `[30,20,10]`.

```
reverse :: [a] -> [a]
reverse []     = []      -- caso de paragem
reverse (x:xs) = reverse xs : x
```

6. Apresente uma definição recursiva da função (pré-definida) `take :: Int -> [a] -> [a]` que dado um inteiro `n` e uma lista `l` calcula a lista com os (no máximo) `n` primeiros elementos de `l`.

   A lista resultado só terá menos de que `n` elementos se a lista `l` tiver menos do que `n` elementos. Nesse caso a lista calculada é igual à lista fornecida.

   Por exemplo, `take 2 [10,20,30]` corresponde a `[10,20]`.

```
take :: Int -> [a] -> [a]
take n _
   | n <= 0   = []          -- duplo caso de paragem para os dois casos, tanto n menor ou igual a 0 e a lista vazia
take _ []     = []
take n (x:xs) = x : take (n-1) xs
```

7. Apresente uma definição recursiva da função (pré-definida) `drop :: Int -> [a] -> [a]` que dado um inteiro `n` e uma lista `l` calcula a lista sem os (no máximo) `n` primeiros elementos de `l`.

   Se a lista fornecida tiver `n` elementos ou menos, a lista resultante será vazia.

   Por exemplo, `drop 2 [10,20,30]` corresponde a `[30]`.

```
drop :: Int -> [a] -> [a]
```

8. Apresente uma definição recursiva da função (pré-definida) `zip :: [a] -> [b] -> [(a,b)]` constói uma lista de pares a partir de duas listas.

   Por exemplo, `zip [1,2,3] [10,20,30,40]` corresponde a `[(1,10),(2,20),(3,30)]`.

9. Apresente uma definição recursiva da função (pré-definida) `replicate :: Int -> a -> [a]` que dado um inteiro `n` e um elemento `x` constói uma lista com `n` elementos, todos iguais a `x`.

   Por exemplo, `replicate 3 10` corresponde a `[10,10,10]`.

10. Apresente uma definição recursiva da função (pré-definida) `intersperse :: a -> [a] -> [a]` que dado um elemento e uma lista, constrói uma lista em que o elemento fornecido é *intercalado* entre os elementos da lista fornecida.

    Por exemplo, `intersperce 1 [10,20,30]` corresponde a `[10,1,20,1,30]`.