1. Define the Idris function `(!!)` for extracting the nth element from a list (use zero for first element).

\[
(!!) : \{a : \text{Type}\} \rightarrow \text{List } a \rightarrow \text{Nat} \rightarrow \text{Maybe } a \\
[] \; \text{!! } \_ \; = \text{Nothing} \\
(x :: \_ ) \; \text{!! } \_ \; = \text{Just } x \\
(_ :: xs) \; \text{!! } (S \; n) = xs \; \text{!! } n
\]
Exercises

4. Define the Idris functions `eqNat` and `eqList` for comparing two natural numbers and two lists of values.

```idris
eqNat : Nat → Nat → Bool
eqNat Z Z = True
eqNat (S n) (S m) = eqNat n m
eqNat _ _ = False

data Nat : Type where
  Z : Nat
  S : Nat → Nat

eqList : Eq a => List a → List a → Bool
eqList [] [] = True
eqList (x::xs) (y::ys) = x==y && eqList xs ys
eqList _ _ _ = False
```
5. Define the Idris function `eqVect` for comparing two vectors of values.

```
eqVect : Eq a => Vect n a → Vect n a → Bool
eqVect []      []      = True
eqVect (x::xs) (y::ys) = x==y && eqVect xs ys
```

Types make 3rd case impossible and thus unnecessary.
6. Define the functions head and tail for vectors.

head : Vect (S n) a → a
head (x :: _) = x

tail : Vect (S n) a → Vect n a
tail (_, :: xs) = xs

data Vect : Nat → Type → Type where
  Nil : Vect Z a
  (∷) : a → Vect n a → Vect (S n) a

Types make empty vector impossible