Exercises

```
(++) : {a : Type} → List a → List a → List a
[]        ++ ys = ys
(x :: xs) ++ ys = x :: (xs ++ ys)
```

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

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

```
(!!) : {a : Type} → List a → Nat → Maybe a
[]      !! _     = Nothing
(x::_)  !! Z     = Just x
(_::xs) !! (S n) = xs !! n
```

For CS 581: All functions in Idris should be total!

Exercises

```
sum : (b : Bool) → Single b → Nat
sum True  x       = x
sum False []      = 0
sum False (x::xs) = x + sum False xs
```

2. *Implement the function* `inc` *for incrementing values of type* `Single b`.

```
inc : (b : Bool) → Single b → Single b
inc True  x       = x+1
inc False []      = []
inc False (x::xs) = x+1::inc False xs
```

Exercises

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

```
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
```

Exercises

```
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

Exercises

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

6. Define the functions `head` *and* `tail` *for vectors.*

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

Types make empty vector impossible

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

Exercises

Height

Width

7. Define a type for matrices with n rows and m columns using nested vectors.

Type

```
Vect n (Vect m a)
```

Type Definition

```
Matrix : Nat → Nat → Type → Type
Matrix n m a = Vect n (Vect m a)
```

8. Define the functions `firstRow` *and* `firstCol` *for matrices.*

```
firstRow : Matrix (S n) m a → Vect m a
firstRow (xs :: _) = xs
```

```
firstCol : Matrix n (S m) a → Vect n a
firstCol xss = map head xss
```

```
firstCol []         = []
firstCol (xs::xss) = head xs::firstCol xss
```