Lab 9
Functional Programming (ITI0212)

1. Write `Semigroup` and `Monoid` instances for `Bool` such that:
```
> True <+> False
False
> False <+> True
False
> True <+> neutral
True
```

2. Write a function that reduces a list of elements of any type with a monoid structure to an element of that type. For example:
```
> reduce [True , False , True]
False
> reduce ["hello " , "brave " , "new " , "world"]
"hello brave new world"
> reduce $ the (List String) []
""
```

3. Working with the definitions from the script file from this week's lecture, write the
```
disjointUnion :  Set a -> Set b -> Set (Either a b)
```
function using the `do` syntax.

4. Write a function
```
join :  Set (Set a) -> Set a
```
that takes a set of sets and unions them together: e.g. `join` $\{\{a, b\}, \{b, c\}, \{b\}, \{a, d\}\} = \{a, b, c, d\}$. Try using the monadic style for extra conciseness.

5. Generalise the previous question by writing a function of type
```
join :  Monad t => t (t a) -> t a.
```

6. Take the `Tree` data type from Lecture 9

```
data Tree: Type -> Type where
    Leaf: (label: a) -> Tree a
    Node: (label: a) -> (child1: Tree a) -> (child2: Tree a) -> Tree a
```

   and write a function

```
gluetrees: Tree a -> Tree a -> Tree a -> Tree a
```

   such that `glueTrees t1 t2 t3` results in a tree that has `t2` and `t3` added as the left and right child of each of the leaves of `t1`.

7. Use `glueTrees` to come up with an implementation of `Applicative` and `Monad` for `Tree`.

8. Use the following function

```
sapling: Unit -> Tree Unit
sapling () = Node () (Leaf ()) (Leaf ())
```

   in conjunction with the monadic structure on `Tree` to write a function that takes a `Nat` n and generates a `Tree Unit` of depth 2^n where depth is defined as follows:

```
depth: Tree a -> Nat
depth (Leaf label) = 1
depth (Node label child1 child2) = 1 + max (depth child1) (depth child2)
```