2 Programming Component

The data structure TaggedList t a contains a list-like data structure, where data of type a in the list could potentially be tagged by tags of type t. If a piece of data x in the list is tagged by tag t, then it will be represented as `Cons(Just t,x,tl)` where tl is the rest of the tagged list.

```
data TaggedList t a =
       Nil
     | Cons (Maybe t, a, TaggedList t a)
```

Question 7: Make `TaggedList t` an instance of the `Functor` type class. When `fmap`ping a TaggedList, the provided function should be applied to the data of type a everywhere in the list, and should keep the associations maintained between mapped data and original tags.

Question 8: Given an input `tag` of type t and an input of type `TaggedList t Int`, calculate the sum of all Ints contained in the list that are tagged by the input `tag`.

A Warn-Once monad is a middle-ground between the WarningAccumulator monad the Maybe monad. In the Warn-Once monad, functions can return one of three states: (1) the computation succeeds with no issues, (2) the computation succeeds, but generates a warning, and (3) the computations fails.

```
data WarnOnce a =
    WarnOnceOk a
  | WarnOnceWarn a
  | WarnOnceError
```

Question 9: Implement the WarnOnce monad. A `return`ed piece of data should use the constructor WarnOnceOk. When binding a computation `f` with type `a -> WarnOnce b` on a piece of data `x` with type `WarnOnce a`, the desired constructor should be as follows:

If `x` matches the pattern `WarnOnceOk v`, the result of the bound computation should simply be the computation of `f v`. If `x` matches the pattern `WarnOnceError`, the computation should simply result in an error. If `x` matches the pattern `WarnOnceWarn v`, the behavior is more interesting. If the result of the computation is an error, an error should be returned. If the result of the computation is a warning, an error should be returned (only one warning is permitted). If the result of the computation is Ok, a warned result should be returned.

A discrete probability distribution monad calculates the output distribution of a probabilistic computation. The representation of a discrete probability distribution is as follows:

```
data DiscProbDist a = DiscProbDist [(Float,a)]
```

The sum of all the floats in the list should add up to 1.0. The discrete probability distribution monad acts similar to a combination of the probability monad and the list monad. Namely: when binding a computation f to a list of values `xs`. The result of that computation is the concatenation of applying f to every element of x, then flattening the result. The individual probabilities are multiplied such that, if $\Sigma_{(p,x)\in xs} p = 1$ and for all x, $\Sigma_{(p,y) inf(x)} p = 1$, then $\Sigma_{(p,y)\in(xs>>=f)} p = 1$.

Question 10 – Extra Credit: Implement the DiscProbDist monad. There are no test cases for this monad. This is extra credit, feel free to ignore this problem.