Practice Exercises on Algebraic Data Types

1. Given the following datatype:

```
data Weekday =
    Monday
  | Tuesday
  | Wednesday
  | Thursday
  | Friday
```

we can say:

a) Weekday is a type with five data constructors
b) Weekday is a tree with five branches
c) Weekday is a product type
d) Weekday takes five arguments

2. Given the following datatype:

```
data Dimension a =
    Plan a a
  | Space a a a
```

we can say:

a) Plan is a type constructor
b) Dimension is a data constructor
c) Plan, Space and Dimension are all functions
d) Only Plan and Space are functions

3. With the same datatype definition in mind from exercise 1, what is the type of the function, **myDay**?

```
myDay Friday = "Miller Time"

a) myDay :: [Char]
b) myDay :: String -> String
c) myDay :: Weekday -> String
d) myDay:: Day -> Beer
```

3. Types defined with the data keyword

a) must have at least one argument
c) must be polymorphic
d) cannot be imported from modules

4. Define a recursive data type for a Binary Tree

a. Write an insert function for it
b. Write a fmap and foldr function for it.
c. Write a function that transform your Tree into a list

5. Create a data type called `Person` that stores a person's full name, address, and phone number. Create a function for getting a person's name and a function for changing their phone number.

6. Convert the data type created in exercise 5 to a record and state wether Person is a Sum or Product type. Justify your choice.

7. Given a data type for days of the week:

```
data Day =
    Monday
  | Tuesday
  | Wednesday
  | Thursday
  | Friday
  | Saturday
  | Sunday
```

write two functions:

a. ***isWednesday***, which takes a day of the week and returns **`True`** if it’s Wednesday and **`False`** otherwise.
b. **`nextDay`**, which takes a day of the week and returns the day of the week that comes after it.
c. Re-implement the previous functionality leveraging the ***Enum type class***

8. Write a ‘tail’ function for a list with the type signature of

`safeTail :: [a] -> Maybe [a]`. It should take a list and return the list without the first element, wrapped in **`Just`**. In case that is not possible, it should return **`Nothing`**.

9. Write a ‘head’ function for a list with the type signature of

`safeHead :: [a] -> Maybe a.` It should take a list and return the first element, wrapped in **`Just`**. In case that is not possible, it should return **`Nothing`**.

10. Write a ‘head’ function for a list with the type signature of

`safeHead :: [a] -> Maybe a`. It should take a list and return the first element, wrapped in **`Just`**. In case that is not possible, it should return **`Nothing`**.

11. Consider the following binary tree type:

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

Let us say that such a tree is balanced if the number of leaves in the left and right subtree of every node differs by at most one, with leaves themselves being trivially balanced. Define a function `balanced :: Tree a -> Bool` that decides if a binary tree is balanced or not.

12. Let's consider the following requirements regarding the contact information for a user in a given system:

A user contact information may:

a. Not exists
b. Only be a Phone number
c. Only be an Email Address
d. Be both Telephone and Email Address

Using algebraic sum data types, define the User and ContactInformation data types that capture the previous requirements

```haskell
data User = User {
  fullname :: Name
, dobirth  :: Date
, contact  :: ContactInfo
}


newtype Name = ...

newtype Date = ...

data ContactInfo = ...
```