Assignment 1

The first assignment is meant to give you practice in writing simple Haskell functions. A code snippet at the end of this document contains the starting code for your assignment.

This assignment deals with a "Person" type. A person is simply a tuple (String, Int, Gender) where the first coordinate is the person's name, the second is their age in years and the last is their gender, which can take one of the values Male or Female (no quotes around them). An example of one student (Susie) is provided.

You will write a series of functions that operate on persons. Here are some things to pay attention to:

- Each function starts with its type/signature. These are commented out so that you can compile the script file. Uncomment the function type when you start working with it.
- The first couple of functions also provide a start point with the first line of the function definition. You should also uncomment those when you start working on the function.
- Use good names for the variables you use in your function. Camel-case them where appropriate.
- Pay attention to the form of the function: Avoid very long or complicated expression.
- Many functions are meant to use previously defined functions. Pay attention to these opportunities.
- Most of your answers will be one-liners.

Description of functions to write

We describe here the functions that you have to write.

1. The first function is provided for you. It is a function called name that returns the person's name. As you can see it is extremely short
2. Write a function age that given a person returns their age, and similarly a function gender that given a person returns their gender.
3. Write functions isMale and isFemale which return a boolean depending on whether the person is male or female. You can do this with or without using the gender function. We have written two different start forms depending on which approach you want to take.

4. Write a function canDrink that given a person returns whether that person is old enough to legally drink (i.e. 21 or above).
5. Write a function closeAge that takes two (curried) Person arguments and returns whether their ages are within 2 years of each other.
6. Write a function oppositeGender that takes two (curried) Person arguments and returns whether they are of opposite gender.
7. Write a function bioCompatible that takes two person arguments and returns whether they are "biologically compatible", which for the purposes of the assignment we define as "at most 2 years apart and of opposite gender".
8. Write a function allAges that given a list of Persons returns a list of their ages. Use the map function for it.
9. Write a function sumOfAges that given a list of Persons returns the sum of their ages. You can do this by using the allAges function along with the built-in sum function.
10. Write a function allCanDrink that given a list of Persons returns whether they are all eligible to drink. You will find the function all useful. It has signature `all :: (a->Bool) -> [a] -> Bool`. So it takes a function that to each a returns a boolean (such functions are called predicates), then it takes a list of as, and applies the function to all of them and determines if they were all True.
11. Write a function ageOne that "ages" a person by one year. It should keep all other information about a person the same, but increase their age by 1. It returns the "new" person.
12. Write a function ageAll that takes a list of Persons and ages them all by 1, and returns the new list of aged Persons.
13. Use guarded expressions to write a function isYoung that takes a person and returns a string as follows: If the person is at most 5 years old the string should be "a toddler". If they are at most 18 years old it should be "a teenager". If they are over 18 years old it should be "young at heart".
14. Write a function describe which given a person "describes" them by producing a string as follows: "This is <name>. He/She is " and then the result of the isYoung function. Finally a closing dot at the end. You should use the appropriate pronoun for the given gender. Think of breaking this function up in smaller pieces using where. Remember that you can use ++ to concatenate lists, in particular to concatenate strings.
15. Write a function makePerson that is given a pair of a name and a gender and creates a new person with age 0.
16. Write a function makeMany that is given a list of names and a list of genders, and produces a list of new people at age 0 from the corresponding pairs. You will want to use zip, map and makePerson.

Starting code

Copy this code and save it into a new Haskell script file. It should have the extension .hs and it should have a filename of the form: LastnameFirstname1.hs. Email me this file to submit your assignment.

```haskell
--- Assignment 1
--- Name:

-- The next couple of lines define some new types. Do not change them.
data Gender = Male | Female deriving (Show, Eq)
type Person = (String, Int, Gender)

-- An example person. Feel free to create more.
susie = ("Susie", 22, Female) :: Person

-- Function 1
name :: Person -> String
name (pName, pAge, pGender) = pName

-- Functions 2
-- Uncomment the next few lines
-- age :: Person -> Int
-- age (pName, pAge, pGender) = ...

-- gender :: Person -> Gender
-- gender (pName, pAge, pGender) = ...

-- Functions 3
-- Uncomment the next few lines
-- isMale :: Person -> Bool
-- Keep this start if you will NOT use the gender function
-- isMale (pName, pAge, pGender) = ...
-- Keep this start if you WILL use the gender function
-- isMale person = ...

-- isFemale :: Person -> Bool
-- Keep this start if you will NOT use the gender function
-- isFemale (pName, pAge, pGender) = ...
-- Keep this start if you WILL use the gender function
-- isFemale person = ...

-- Function 4
-- Uncomment the next line
-- canDrink :: Person -> Bool

-- Function 5
-- Uncomment the next line
-- closeAge :: Person -> Person -> Bool

-- Function 6
-- Uncomment the next line
-- oppositeGender :: Person -> Person -> Bool

-- Function 7
-- Uncomment the next line
-- bioCompatible :: Person -> Person -> Bool

-- Function 8
-- Uncomment the next line
-- allAges :: [Person] -> [Int]

-- Function 9
-- Uncomment the next line
-- sumOfAges :: [Person] -> Int

-- Function 10
-- Uncomment the next line
-- allCanDrink :: [Person] -> Bool

-- Function 11
-- Uncomment the next line
-- ageOne :: Person -> Person

-- Function 12
-- Uncomment the next line
-- ageAll :: [Person] -> [Person]

-- Function 13
-- Uncomment the next line
-- isYoung :: Person -> String

-- Function 14
-- Uncomment the next line
-- describe :: Person -> String

-- Function 15
-- Uncomment the next line
-- makePerson :: (String, Gender) -> Person

-- Function 16
-- Uncomment the next line
-- makeAll :: [String] -> [Gender] -> [Person]
```