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 "Pet" type. A pet is simply a tuple (String, Int, Species) where the first coordinate is the pet's name, the second is their age in years and the last is their species, which can take one of the values Cat or Dog (no quotes around them). An example of one pet is provided.

You will write a series of functions that operate on pets. 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.
- Use *list comprehensions* for the problems asking you to work with lists.

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 pet's name. As you can see it is extremely short
2. Write a function age that given a pet returns their age, and similarly a function species that given a pet returns their species.
3. Write functions isCat and isDog which return a boolean depending on whether the pet is cat or dog. You can do this with or without using the species function. We have written two different start forms depending on which approach you want to take.

4. Write a function isOld that given a pet returns whether that pet is "old". We define old for dogs as being at least 10 years of age, and for cats as at least 11 years of age.

5. Write a function closeAge that takes two pet arguments and returns whether their ages are within 2 years of each other.

6. Write a function differentSpecies that takes two pet arguments and returns whether they are different species.

7. Write a function opposites that takes two pet arguments and returns whether they are "opposites", which for the purposes of the assignment we define as "at most 2 years apart and of different species".

8. Write a function allAges that given a list of pets returns a list of their ages.

9. Write a function sumOfAges that given a list of pets returns the sum of their ages. You can do this by using the allAges function along with the built-in sum function which adds up all numbers in a list.

10. Write a function allOld that given a list of pets returns whether they are all old. Use the function isOld with a list comprehension, which should produce a list of booleans, then combine that with the and function, which takes a list of booleans and looks at their "AND", i.e. whether they are all true.

11. Write a function ageOne that "ages" a pet by one year. It should keep all other information about a pet the same, but increase their age by 1. It returns the "new" pet.

12. Write a function ageAll that takes a list of pets and ages them all by 1, and returns the new list of aged pets.

13. Write a function describe that takes a pet and returns a string as follows: It will produce one of the messages "young cat", "young dog", "old cat", "old dog" depending on whether the pet is a cat or a dog and whether they are old as described above. You can use helper functions defined in a where clause.

14. Write a function introduce which given a pet "introduces" them by producing a string as follows: "This is <name>. <name> is a " and then the result of the describe function. Finally a closing dot at the end. Remember that you can use ++ to concatenate lists, in particular to concatenate strings.

15. Write a function makePet that is given a pair of a name and a species and creates a new pet with age 0.

16. Write a function makeMany that is given a list of pairs of a name and a species, and produces a list of new pets at age 0 from the corresponding pairs. You will want to use list comprehensions along with makePet.

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.

```
-- Assignment 1
-- Name:

-- The next couple of lines define some new types. Do not change them.
data Species = Cat | Dog deriving (Show, Eq)
type Pet = (String, Int, Species)

-- An example pet. Feel free to create more.
skye = ("Skye", 3, Cat) :: Pet

-- Function 1
name :: Pet -> String
name (pName, pAge, pSpecies) = pName

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

-- species :: Pet -> Species
-- species (pName, pAge, pSpecies) = ...

-- Functions 3
-- Uncomment the next few lines
-- isCat :: Pet -> Bool
-- Keep this start if you will NOT use the species function
-- isCat (pName, pAge, pSpecies) = ...
-- Keep this start if you WILL use the species function
-- isMale Pet = ...

-- isDog :: Pet -> Bool
-- Keep this start if you will NOT use the species function
-- isDog (pName, pAge, pSpecies) = ...
-- Keep this start if you WILL use the species function
-- isFemale Pet = ...

-- Function 4
-- Uncomment the next line
-- isOld :: Pet -> Bool

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

-- Function 6
-- Uncomment the next line
-- differentSpecies :: Pet -> Pet -> Bool

-- Function 7
-- Uncomment the next line
```

```
-- opposites :: Pet -> Pet -> Bool

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

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

-- Function 10
-- Uncomment the next line
-- allOld :: [Pet] -> Bool

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

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

-- Function 13
-- Uncomment the next line
-- describe :: Pet -> String

-- Function 14
-- Uncomment the next line
-- introduce :: Pet -> String

-- Function 15
-- Uncomment the next line
-- makePet :: (String, Species) -> Pet

-- Function 16
-- Uncomment the next line
-- makeMany :: [(String, Species)] -> [Pet]
```