MATHFUN
Discrete Mathematics and Functional Programming

Worksheet 4: Tuples, Strings and Lists

Introduction

This worksheet is intended to introduce you to writing functions using tuples, strings and lists. Begin by copying the Week4.hs file from the unit website to your file-space. This file defines a module Week4 containing most of the functions, and the type synonym `StudentMark`, from the lecture. Load this script into GHCi, and experiment by evaluating some expressions. Add the solutions to the programming exercises to the script. Note the use of the `testData` constant; this can be used to easily test those functions that involve student marks.

Tuples

1. Write a function:

   ```
   sumDifference :: Int -> Int -> (Int,Int)
   ```

   which returns both the sum and the difference between the first and second arguments. For example, `sumDifference 3 7 = (10,-4)`.

2. Write a function:

   ```
   grade :: StudentMark -> Char
   ```

   which returns a student's grade from a percentage mark. Marks of 70 or above get an A grade, marks between 60 and 69 get a B, between 50 and 59 a C, between 40 and 49 a D, and marks below 40 get an F.

3. Write a function:

   ```
   capMark :: StudentMark -> StudentMark
   ```

   which caps the mark of a student to a maximum of 40.

Lists and Strings

4. Write a function:

   ```
   firstNumbers :: Int -> [Int]
   ```

   that gives a list of the first n positive integers (e.g. `firstNumbers 3 = [1, 2, 3]`).

5. Write a function:

   ```
   firstSquares :: Int -> [Int]
   ```

   that gives a list of the first n squares (e.g. `firstSquares 3 = [1, 4, 9]`).

6. Using a list comprehension, write a function:

   ```
   capitalise :: String -> String
   ```

   which converts all the small letters in a string to capitals (e.g. `capitalise "Po1 3he"` is `"PO1 3HE"`).

7. Using a list comprehension, write a function:

   ```
   onlyDigits :: String -> String
   ```

   that strips all non-digit characters from a string (for example, `onlyDigits "ac245d62"` is `"24562"`).

8. Using your `capMark` function and a list comprehension, write a function:

   ```
   capMarks :: [StudentMark] -> [StudentMark]
   ```

   which caps all students' marks to a maximum of 40%. For example:

   ```
   capMarks [("Jo",37), ("Sam",76)] = [("Jo", 37), ("Sam", 40)]
   ```

9. Using your `grade` function and a list comprehension, write a function:

   ```
   gradeStudents :: [StudentMark] -> [(String,Char)]
   ```

   which grades a list of students marks. For example,

   ```
   gradeStudents [("Jo",47), ("Sam",76)] = [("Jo",'D'), ("Sam",'A')]
   ```

10. Using recursion, write a function:

    ```
    duplicate:: String -> Int -> String
    ```

    which joins copies of a string together (e.g. `duplicate "Hi" 3 = "HiHiHi"`).

11. Using a list comprehension, write a function:

    ```
    divisors :: Int -> [Int]
    ```

    which returns the list of the divisors of a positive integer (and `[]` for any other integer). (E.g. `divisors 15 = [1, 3, 5, 15]`.)

12. Using your `divisors` function, write a function:

    ```
    isPrime :: Int -> Bool
    ```

    which tests whether an integer is a prime number (hint: how many divisors do prime numbers have?)

13. Using list comprehensions, write a polymorphic function:

    ```
    split:: [(a,b)] -> ([a],[b])
    ```

    which transforms a list of pairs (of any types) into a pair of lists. For example,

    ```
    split [(1,'a'), (2,'b'), (3,'c')] = ([1,2,3], "abc")
    ```