Functional Languages 5th practice

12. 03. 2019.

1. Calculate the average of a list of integers. Use `fromIntegral`.

   ```
   avg [1,2,3,4,5,6] == 3.5
   avg [5,10] == 7.5
   avg [100] == 100
   avg [] == 0
   ```

2. At an exchange office, the forint/euro rate is 363.82. Exchange euros to forints. Results should be integers.

   ```
   exchange :: Int -> Int

   exchange 0 == 0
   exchange 1 == 363
   exchange 100 == 36382
   exchange 1000 == 363820
   exchange 20 == 7276
   ```

3. Define a function `isPrime` which returns `True` only if its parameter is a prime number.

   ```
   not (isPrime 0)
   not (isPrime 1)
   not (isPrime 6)
   isPrime 2
   isPrime 3
   isPrime 7
   isPrime 101
   ```

   Challenge: try to give an efficient version that checks divisors up to the square root.

4. Define a constant `primes` which is an infinite list of primes.

   ```
   take 5 primes == [2, 3, 5, 7, 11]
   ```

5. Construct a list of all dominoes:
   `[(0,0), (0,1), ..., (0,6), (1,1), ..., (6, 6)]`

   ```
   length dominoes == 28
   ```

6. *Enumerate all integer pairs: `[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0), ...]`

7. Construct a numbering of the English alphabet.

   ```
   take 5 alphabet == [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
   length alphabet == 26
   ```

   Hint: using `zip` here helps.

8. Pick every third letter from the English alphabet using a list comprehension.

   ```
   everyThird == "cfilorux"
   ```

   Hint: `cycle` could help as it can create a cylic list. Try `take 6 (cycle [1, 2, 3])`

9. Check whether an integer is a square number.

   ```
   square 4
   square 16
   square 0
   square 1
   square 25
   not (square 5)
   not (square 12)
   ```

10. In Neptun, courses are stored in a list. Each course has name and students:

    ```
    courses =
      [ ("Calculus",               [("Simon", "Jones", "BDE91E"), ("Barack", "Obama", "DDA3KX")])
      , ("Imperative Programming", [("Simon", "Marlow", "ALX1K0"), ("John", "Hughes", "BDE91E")])
      , ("Functional Languages",   [("Philip", "Wadler", "ABCDE6"), ("Simon", "Thompson", "CDE560")])
      ]
    ```

    List all Functional Languages students using a single list comprehension. Avoid using indexing as we make no assumptions about the place of the course in the list.

    ```
    students == ["ABCDE6", "CDE560"]
    ```

    Make `students` a general function that takes course name as parameter.

11. *Enumerate days in (day, month) form of a 365-day year in a list.

    Hint: function `elem :: a -> [a] -> Bool` helps in deciding which months have 31 days and which have only 30.

    ```
    elem (31, 1) calendar
    not (elem (32, 1) calendar)
    elem (28, 2) calendar
    not (elem (29, 2) calendar)
    elem (30, 4) calendar
    not (elem (31, 4) calendar)
    length calendar == 365
    ```

12. Take the string `"aaaabccaadeeee"`. Compress it so that consecutive letters are packed into (length, letter) pairs:

    `[(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]`!

    Hint: `group` forms groups of letters, which helps in getting started.

    ```
    compress "aaaabccaadeeee" == [(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]
    compress "oh hello!!"     == [(1,'o'),(1,'h'),(1,' '),(1,'h'),(1,'e'),(2,'l'),(1,'o'),(2,'!')]
    compress ""               == []
    ```

13. Define a function `decompress` which restores the original string from a compressed form. This is the inverse function of `compress`.

    Hint: take a look at functions `concat :: [[a]] -> [a]` and `replicate :: Int -> a -> [a]`.

    ```
    decompress [(4,'a'), (1,'b'), (2,'c'), (2,'a'), (1,'d'), (4,'e')]            == "aaaabccaadeeee"
    decompress [(1,'o'),(1,'h'),(1,' '),(1,'h'),(1,'e'),(2,'l'),(1,'o'),(2,'!')] == "oh hello!!"
    decompress [] == ""
    ```