Functional Programming for Logicians

Péter Mekis
Department of Logic, ELTE Budapest

Deadline: 2019 February 18 17:59 pm

- Define *any three of the following functions* in Haskell. Defining more than three is appreciated, but not necessary. Some of the exercises are follow-ups to others; it may be a good idea to choose them together.
- Also, define *three functions that aren't in this list*, based on your ideas, preferably connected to your main field of interest.
- Get ideas from the functions we defined in the first session. Don't use advanced tools like list comprehension, lambda abstraction, or importing modules. If Haskell has a built-in solution for an exercise, don't use it. If you need syntax that wasn't covered in the session, google it. If the description doesn't cover all cases, be creative!
- If you use code that was created by someone else, indicate it.
- Make sure you submit a code that compiles in ghci. Annotation is appreciated.
- The exercises range from the more elementary to the more advanced. Choose those that are at your level. Have fun! :)

1. **Type** `Int -> Int`

   Description Calculates the age of a person based on their birth year.

   Examples

   ```
   > age 2011
   8
   > age 1923
   96
   ```

2. **Type** `Int -> (Int -> Int)`

   Description Calculates the square sum of two integers.

   Examples

   ```
   > sqsum 5 7
   74
   > sqsum (-1) 1
   2
   ```

3. **Type** `Int -> Int`

   Description Calculates the absolute value of its input.

   Examples

   ```
   > abs' 17
   17
   > abs' (-17)
   17
   ```

4. **Type** `Bool -> Bool`

   Description It negates a truth value.

   Examples

   ```
   > not' True
   False
   > not' False
   True
   ```

5. **Type** `Int -> (Bool -> Int)`

   Description Calculates the age of a person, taking into account whether they have already had their birthday this year.

   Examples

   ```
   > age' 2011 True
   8
   > age' 1923 False
   95
   ```

6. **Type** `Bool -> (Bool -> Bool)`

   Description Implication (conditional): it returns False iff the first argument is true, and the second is false.

   Examples

   ```
   > age' 2011 True
   8
   > age' 1923 False
   95
   ```

7. **Type** `Int -> String`

   Description Returns `"negative"` if the input is negative; `"positive"` if it's positive; and `"zero"` otherwise.

   Examples

   ```
   > sign 17
   "positive"
   > sign (-17)
   "negative"
   ```

8. **Type** `Int -> String`

   Description If the input is a valid grade in the Hungarian academic system (5, 4, 3, 2, or 1), the function returns it as a term (`"excellent"`, `"good"`, `"fair"`, `"sufficient"`, `"fail"`). Otherwise it returns an error message.

   Examples

   ```
   > grade 4
   "good"
   > grade 7
   "not a grade!"
   ```

9. **Type** `Int -> (Int -> (Int -> Bool))`

   Description Checks if three integers form a Pythagorean triple. (Cf. `https://en.wikipedia.org/wiki/Pythagorean_triple`)

   Examples

   ```
   > pythtriple 5 12 13
   True
   > pythtriple 6 12 13
   False
   ```

10. **Type** `Int -> (Int -> (Int -> Int))`

    Description Returns the maximum of three integers.

    Examples

    ```
    > max3 5 7 3
    7
    > max3 (-5) (-7) (-3)
    -3
    ```