CS320 Programming Languages
Homework #1

Due: 6 March 2019

1 Integers

1. Define the function `dollar2won`, which consumes an integer number of dollars and produces the won equivalent. Use the won/dollar conversion rate of 1100 won per dollar.
2. Write the function `volumeOfCuboid`, which consumes three integer numbers denoting lengths of three sides and produces the volume of the cuboid.
3. Write the function `isEven`, which consumes an integer number and returns whether the number is even.
4. Write the function `isOdd`, which consumes an integer number and returns whether the number is odd.
5. Write the function `gcd`, which consumes two integer numbers and returns the greatest common divisor of them.
6. Write the function `lcm`, which consumes two integer numbers and returns the least common multiple of them.

2 Pattern Matching

You have a type `COURSE`, which is either `CS320`, `CS311`, or `CS330`. `CS320` has two members: `quiz` for a number of quizzes and `homework` for a number of programming assignments. CS311 has one member: `homework` which is a number too. `CS330` has two members: `projects` for a number of projects and `homework` for a number of programming assignments.

```
trait COURSE
case class CS320(quiz: Int, homework: Int) extends COURSE
case class CS311(homework: Int) extends COURSE
case class CS330(projects: Int, homework: Int) extends COURSE
```

1. Define the function `numOfHomework`, which consumes a course and produces the number of programming assignments for the given course.
2. Define the function `hasProjects`, which consumes a course and produces true only when the given course is `CS330` with more than or equal to two projects, otherwise produces false.

3 List

1. Define the function `namePets`, which consumes a list of pets and produces a corresponding list of pets with names; it names all occurrences of `dog` with `happy`, `cat` with `smart`, `pig` with `pinky`, and keeps the other pets as unnamed. For example,

   ```
   namePets(List("dog", "tiger", "cat")) == List("happy", "tiger", "smart")
   ```

2. Generalize namePets to the function giveName. The new function consumes two strings, called old and new. It produces a function that gets a list of strings and replaces all occurrences of old by new in the list. For example,

```
namePets(List("dog", "tiger", "cat")) == List("happy", "tiger", "smart")
val nameBears: List[String] => List[String] = giveName("bear", "pooh")
nameBears(List("pig", "cat", "bear")) = List("pig", "cat", "pooh")
```