Starred exercises are optional.

1) Bubble sort is a sorting algorithm. It repeatedly steps through its input, comparing and swapping adjacent elements. This is repeated until the list is sorted. Implement bubble sort of lists over the binary alphabet $\{0, 1\}$ in

   - In a functional programming language (e.g. Haskell)
   - In an imperative high-level programming language (e.g. Java)
   - As a Turing machine

   Are these programs 'the same' in some reasonable way?
   To answer this you may consider questions such as: Does each of your programs terminate/halt on any input? Are the complexities of your programs the same? How do the sizes of their inputs relate to each other, e.g. can lists in Haskell be compared to words in Java?

2) a) A web developer develops websites for all freelancers that do not develop their own website. Is the web developer a freelancer?

      Hint: Approach the problem similarly to the diagonalisation argument from lecture 12, slide 16.

   b) Let $(f_i)_{i\in\mathbb{N}}$ be the family of functions $\mathbb{N} \to \mathbb{N}$ (where $\mathbb{N}$ includes 0) defined by $f_i(n) = i \cdot n$ for all i,n. Give a function g that 'diagonalises away' from the family, i.e. such that we have $g \neq f_i$ for all i. Make clear that g satisfies the condition by giving for each i some input n for which $g(n) \neq f_i(n)$.

3) One of the following:

   a) The same as in 2b), but now asking to construct infinitely many such and also different from the previous ones. That is, give for each $j > 0$ a function g_j that 'diagonalises away' both from the above family $(f_i)_{i\in\mathbb{N}}$ and the functions $g_0, \ldots, g_{j-1}$ constructed before g_j. You may specify the g_j as you like, e.g. by a mathematical definition, by graphing them, or by programming them. However, it should be made clear that/why they satisfy the condition.

   b) Propose an exam exercise together with your solution and the time it should take to answer it and write the solution down. Exercises can be on any topic covered in the course.
      Cf. the three exams available from last year's (`http://cl-informatik.uibk.ac.at/teaching/ws19/ds/schedule.php`) site for typical exercises and their solutions.

4*) The same as in 3b), but now for a bonus exercise. That is, the topic of the exercise should be closely related to a topic covered in the course, but need not be completely covered in the course.
   E.g. one can think of surprising applications of covered methods, or variations on covered problems or methods (e.g. if a shortest path algorithm was covered one could ask: what about longest paths?). Cf. the bonus exercises thus far in the PS and the in the above exams.