General Information

Detailed information about the lecture, tutorials and homework assignments can be found on the lecture website[1]. Solutions have to be submitted to Moodle[2]. Make sure your uploaded documents are readable. Blurred images will be rejected. Use Piazza[3] to ask questions and discuss with your fellow students.

Submission

Upload your solutions as a single file named 'hw05.ml' to moodle. Since submissions are tested automatically, solutions that do not compile or do not terminate within a given time frame cannot be graded and thus result in 0 Points. If you do not manage to get all your stuff compiling and/or terminating, comment the corresponding parts of the file!

Functional Programming

Since this is a course about functional programming, we restrict ourselves to the functional features of OCaml. In other words: The imperative and object oriented subset of OCaml must not be used.

[1] `https://www.in.tum.de/i02/lehre/wintersemester-1819/vorlesungen/functional-programming-and-verification/`

[2] `https://www.moodle.tum.de/course/view.php?id=44932`

[3] `https://piazza.com/tum.de/fall2018/in0003/home`

More Students!

In this assignment you are supposed to extend the students database introduced in assignment 5.4. All type definitions and functions developed in 5.4 are already in the file 'hw05.ml'.

Implement these new functions:

1. `remove_by_id : int -> database -> database` removes the student with the given id from the database. [1 Point]
2. `count_in_semester : int -> database -> int` counts the number of students in the given semester. [1 Point]
3. `student_avg_grade : int -> database -> float` computes the average grade of the student with the given id. If no student with the given id exists or the student does not have any grades, the function shall return `0.0`. [2 Points]
4. `course_avg_grade : int -> database -> float` computes the average grade achieved in the given course. If no grades in the given course exist, the function shall return `0.0`. [2 Points]

List Mishmash

Implement a function `interleave3 : 'a list -> 'a list -> 'a list -> 'a list` that interleaves three lists. So for input lists `[0;1;2]`, `[10;11;12]` and `[20;21;22]` the function must return the list `[0;10;20;1;11;21;2;12;22]`. If a list is out of elements (e.g. for inputs of different lengths), the function continues interleaving the remaining lists, such that for inputs `['a';'b']`, `['A';'B';'C';'D']` and `['!']` the output `['a';'A';'!';'b';'B';'C';'D']` is produced.

OCamlfication

Consider the following function `foo` implemented in an imperative programming language:

```
int foo(int x, int y, bool b) {
  if(x > y) {
    int t = x;
    x = y;
    y = t;
  }

  while(x < y) {
    if(b) {
      ++x;
    } else {
      --y;
```

```
    }
    b = !b;
  }

  return x;
}
```

Implement a semantically equivalent function `foo : int -> int -> bool -> int` in OCaml.

Polynomial Party

A polynomial

$$c_n x^n + c_{n-1} x^{n-1} + \ldots + c_2 x^2 + c_1 x + c_0$$

is represented by the list $[c_n ; c_{n-1} ; \ldots ; c_2 ; c_1 ; c_0]$ of its coefficients.

1. Implement `eval_poly : float -> float list -> float` that evaluates the polynomial (2. argument) for a given x (1. argument). [2 Points]
2. Implement `derive_poly : float list -> float list` that returns the first derivative of the given polynomial. [2 Points]

Longest Twins

Implement a function `lt_seq : 'a list -> 'a list` that returns the longest sequence of elements that appears at least twice in the given list. If “‘