FPV Tutorübung

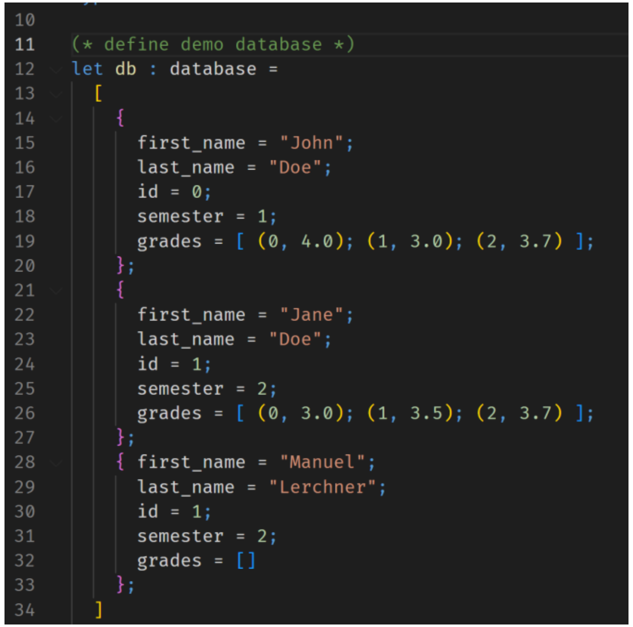
Woche 9

OCaml: Side Effects, Exceptions and Files

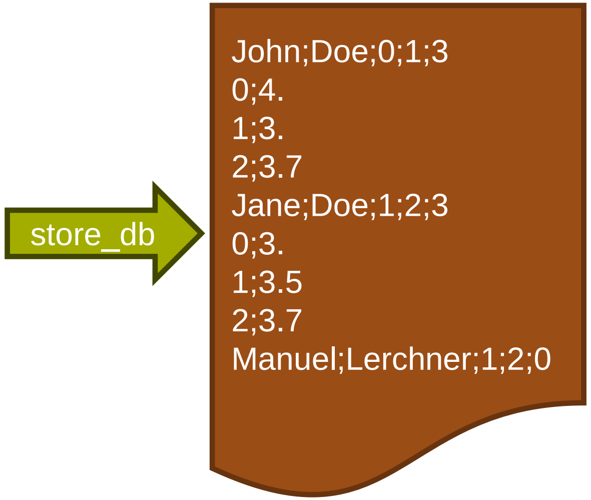
Manuel Lerchner

19.06.2023

T01: Students In Students Out

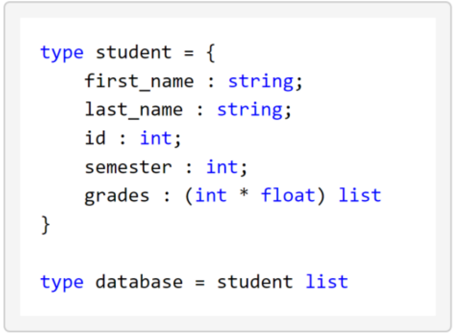


```
(* define demo database *)
let db : database =
  [
    {
      first_name = "John";
      last_name = "Doe";
      id = 0;
      semester = 1;
      grades = [ (0, 4.0); (1, 3.0); (2, 3.7) ];
    };
    {
      first_name = "Jane";
      last_name = "Doe";
      id = 1;
      semester = 2;
      grades = [ (0, 3.0); (1, 3.5); (2, 3.7) ];
    };
    { first_name = "Manuel";
      last_name = "Lerchner";
      id = 1;
      semester = 2;
      grades = []
    };
  ]
```



```
John;Doe;0;1;3
0;4.
1;3.
2;3.7
Jane;Doe;1;2;3
0;3.
1;3.5
2;3.7
Manuel;Lerchner;1;2;0
```

student_database.txt



```
type student = {
    first_name : string;
    last_name : string;
    id : int;
    semester : int;
    grades : (int * float) list
}

type database = student list
```

Now, we define a file format to store students that, for each student, contains a line

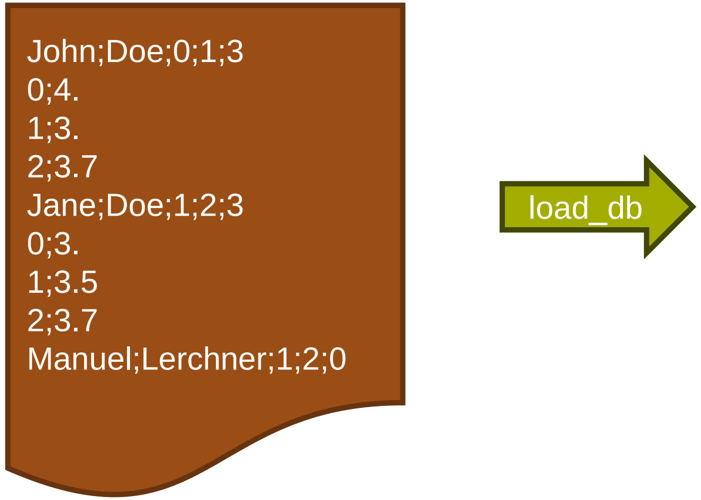
$$first\_name; last\_name; id; semester; grade\_count$$

followed by a number of lines

$$course; grade$$

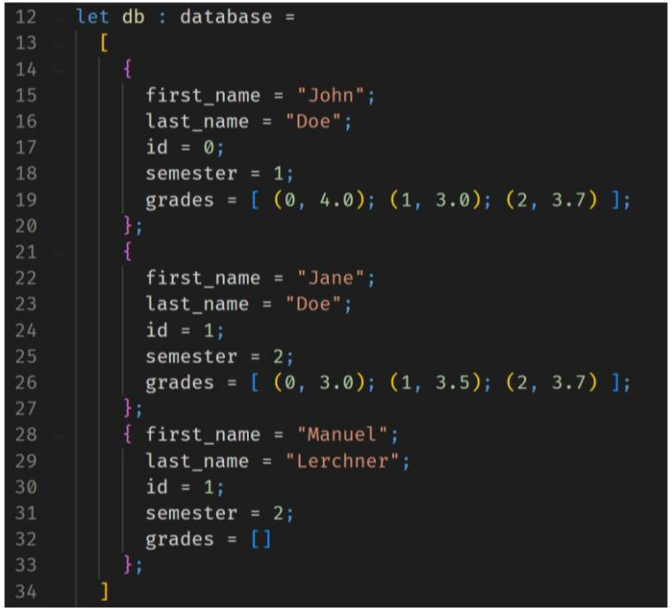
with grades.

T01: Students In Students Out



```
John;Doe;0;1;3
0;4.
1;3.
2;3.7
Jane;Doe;1;2;3
0;3.
1;3.5
2;3.7
Manuel;Lerchner;1;2;0
```

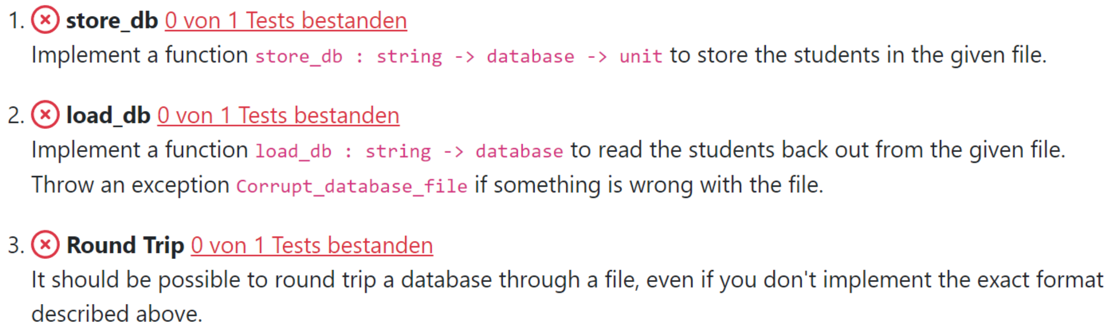
student_database.txt



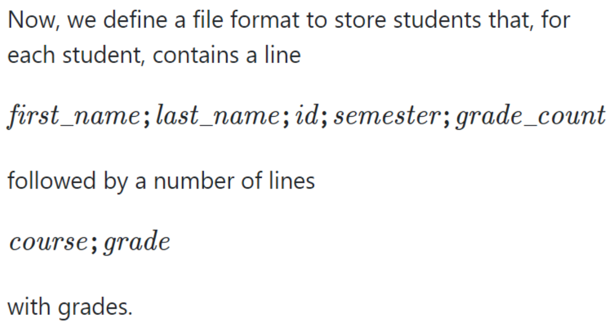
```
let db : database =
  [
    {
      first_name = "John";
      last_name = "Doe";
      id = 0;
      semester = 1;
      grades = [ (0, 4.0); (1, 3.0); (2, 3.7) ];
    };
    {
      first_name = "Jane";
      last_name = "Doe";
      id = 1;
      semester = 2;
      grades = [ (0, 3.0); (1, 3.5); (2, 3.7) ];
    };
    { first_name = "Manuel";
      last_name = "Lerchner";
      id = 1;
      semester = 2;
      grades = []
    };
  ]
```

T01: Students In Students Out

- File I/O
  - `open_in`
  - `open_out`
  - `close_in`
  - `close_out`
  - `input_line`
  - `output_string`
- Exceptions
  - `try` expr `with` exn -> expr
- Other helpful functions
  - `String.split_on_char`
  - `String.concat`
  - `List.iter`



1. ⊗ **store_db** 0 von 1 Tests bestanden
   Implement a function `store_db : string -> database -> unit` to store the students in the given file.
2. ⊗ **load_db** 0 von 1 Tests bestanden
   Implement a function `load_db : string -> database` to read the students back out from the given file. Throw an exception `Corrupt_database_file` if something is wrong with the file.
3. ⊗ **Round Trip** 0 von 1 Tests bestanden
   It should be possible to round trip a database through a file, even if you don't implement the exact format described above.



Now, we define a file format to store students that, for each student, contains a line

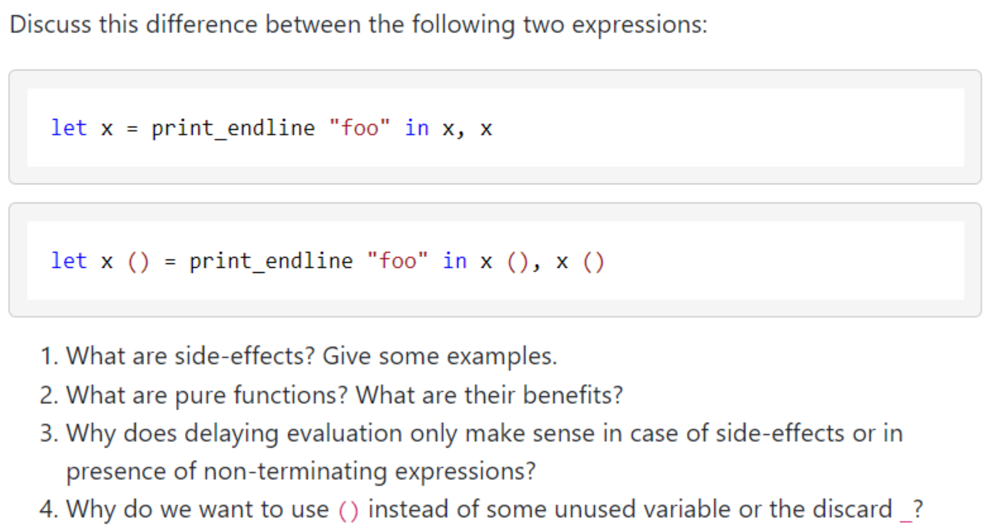
$$first\_name; last\_name; id; semester; grade\_count$$

followed by a number of lines

$$course; grade$$

with grades.

T02: (Delayed) Evaluation, Side-effects, Pure Functions



Discuss this difference between the following two expressions:

```
let x = print_endline "foo" in x, x
```

```
let x () = print_endline "foo" in x (), x ()
```

1. What are side-effects? Give some examples.
2. What are pure functions? What are their benefits?
3. Why does delaying evaluation only make sense in case of side-effects or in presence of non-terminating expressions?
4. Why do we want to use `()` instead of some unused variable or the discard `_`?