Week 5: OCaml

Week 05 Tutorial 01 — Expressions

So far, you learned about the following types of expressions:

- Constants
- Variables
- Unary operators
- Binary operators
- Tuples
- Records
- Lists
- If-then-else
- Pattern matching
- Function definition
- Function application
- Variable binding

For each of the aforementioned types of expressions, give the general structure and two concrete examples with different subexpressions.

Week 05 Tutorial 01 — Expressions

-Constants

1 1. fun x → x + 7

-Variables

ab x 5

-Unary operators

- -5 not

-Binary operators

+ - * ÷

Week 05 Tutorial 01 — Expressions

-Tuples

(1,2) (2,1) (true, false) (1., 2.) (1, "Hello")

-Records(definition)

type t = {a : int ; b : string}

-Records(access)

{a = 10 ; b = "Hello"}

record.a

-Lists

[10 ; 5]

[] 10::5::[]

5::"Hello"::[]

Week 05 Tutorial 01 — Expressions

-If-then-else

```
if a = 5 then true else false
```

-Pattern matching

```
match list with
  | [] -> failwith "H"
  | c::x::z -> c
```

Week 05 Tutorial 01 — Expressions

-Function definition

(fun x,y -> x + 7) int * 'a -> int

-Function application

foo "x"

-Variable binding

Definition let x = (fun a -> 2*a) in x 3

Week 05 Tutorial 01 — Expressions

```
let a = fun x y -> x + 2 in a 3 8 :: [] int list
```

x + 2

fun x y -> x + 2

int

int -> 'a -> int

a 3 8

int

a 3 8 :: []

int list

[5]

Week 05 Tutorial 01 — Expressions

```
((fun x -> x::[]) (9 - 5), true, ('a', 7))
```

int list * bool * (char * int)

⇒ ([4], true, ('a', 7))

Week 05 Tutorial 02 -- What Is the Point

What's the point?

Using what you learned about tuple types in the lecture, implement functionality for computing with three-dimensional vectors.

1. **Define a suitable data type for your point.** 1 of 1 tests passing
   The type `vector3` should be a tuple of 3 float values.
2. **Define three points** 1 of 1 tests passing
   The points `p1`, `p2` and `p3` should all be different, but their exact values don't matter. Use them, along with other points, to test your functions.
3. **string_of_vector3** 1 of 1 tests passing
   Implement a function `string_of_vector3 : vector3 -> string` to convert a vector into a human-readable representation.
   For example, the string for the zero vector should be: `(0.,0.,0.)`.
   Hint: use `string_of_float` to convert components.
4. **vector3_add** 1 of 1 tests passing
   Write a function `vector3_add : vector3 -> vector3 -> vector3` that adds two vectors component-wise.
5. **vector3_max** 1 of 1 tests passing
   Write a function `vector3_max : vector3 -> vector3 -> vector3` that returns the larger argument vector (the vector with the greater magnitude).
6. **combine** 1 of 1 tests passing
   Write a function `combine : vector3 -> vector3 -> vector3 -> string` that adds its first argument to the larger of the other two arguments and returns the result as a string.

Week 05 Tutorial 03 -- Student Database

Student Database

In this assignment, you have to manage the students of a university.

1. ✓ **Type** 1 of 1 tests passing
   First you need to define some types.
   - Define a data type for a `student`.
     A student should be represented as a record of the students `first_name`, `last_name`, identification number `id`, number of the current `semester` as well as the list of `grades` received in different courses.
     The grades should be a pair of the course number and the grade value, a floating point number.
   - To actually manage student you need a `database` which shall be represented as a list of students.
2. ✓ **insert** 1 of 1 tests passing
   Write a function `insert : student -> database -> database` that inserts a student into the database.
3. ✓ **find_by_id** 1 of 1 tests passing
   Write a function `find_by_id : int -> database -> student list` that returns a list with all students with the given id (either a single student or an empty list, if no such student exists).
4. ✓ **find_by_last_name** 1 of 1 tests passing
   Implement a function `find_by_last_name : string -> database -> student list` to find all students with a given last name.