Fuctions

Functions are the most important building blocks, they have arguments and results

Inside the function body you can call the function, thus creating recursion. In ML helper functions must be defined before their uses. Anything you can do with a loop you can do with recursion.

Function bindings

Syntax: the keyword fun, a variable name `fun x0(x1: t1, ..., xn: tn) = e`

Type Checking: the end result of type checking a functions body, we take the function name `x0` and add it to the static environment with the type takes `x0 : (t1 *...* tn) -> t` *if* the function binding **aka the function body** type checks appropriately.

We type check the body of the function `e` using:

- everything in the enclosing static environment (earlier bindings)
- `x1: t1, ... , xn: tn` using the arguments with their types
- `x0: (t1 *...* tn) -> t` the function `x0` type can be used in `e`, it is in the static environment.

Evaluation: a function is already a value, what happens is the `x0` gets added to the environment so later expressions can call it. The evaluation of the function happens when you call the function.

Type checking with more detail

`fun x0 (x1:t1, ,..., xn:tn)` The result type `(t1, *...* tn)->t` the overall type checking result is to give `x0` this type. The variable names `x1...xn` are not added in the static environment after this binding, their only in the static environment during the body of this function. Those arguments are only in scope for `e`.

The type checker type checks `e` gets some type and the return type of `x0` is the type of `e`

Function calls

Syntax: Written with one expression: what function we want to call and some other expressions comma seperated.

Type-checking:

1. `e0` has to have some type `(t1*...*tn)-> t`
2. `e1` has type t1, ... en has type tn
3. then `e0`(e1,...en) has type t

Evaluation:

1. Look up `e0` in the dynamic environment to figure out which function you are going to call
2. Evaluate all of the arguments, to get their values. Eagerly evaluate the arguments
3. Result is evaluate of `e` the function body, in an environment extended to map x1 to v1 ... xn to vn, "an environment" is the environment the function was defined and includes `x0` for recursion.