What is a programming language?

- Here are separable concepts for defining and learning a language:
  - syntax: how do you write the various parts of the language?
  - semantics: what do programs mean? (One way to answer: what are the evaluation rules?)
  - idioms: how do you typically use the language to express computations?
  - libraries: does the language provide “standard” facilities such as file-access, hashtables, etc.? How?
  - tools: what is available for manipulating programs in the language? (e.g., compiler, debugger, REP-loop)

ML basics

- A program is a sequence of *bindings*
- One kind of binding is a *variable binding*

  `val x = e ;` (semicolon optional in a file)

- A program is evaluated by evaluating the bindings in order
- A *variable binding* is evaluated by:
  - Evaluating the expression in the environment created by the previous bindings. This produces a *value*.
  - Extending the (top-level) environment to bind the variable to the value.
- Examples of values: 13, 4.8, true, “hello”, [3, 4, 5], (8, 8.2)

Critical language concepts

- Expressions have a *syntax* (written form)
  - E.g.: A constant integer is written as a digit-sequence
  - E.g.: Addition expression is written e1 + e2
- Expressions have *types* given their context
  - E.g.: In any context, an integer has type int
  - E.g.: If e1 and e2 have type int in the current context, then e1+e2 has type int
- Expressions *evaluate* to values given their environment
  - E.g.: In any environment, an integer evaluates to itself
  - E.g.: If e1 and e2 evaluate to c1 and c2 in the current environment, then e1+e2 evaluates to the sum of c1 and c2

Function definitions

- A second kind of binding is for functions (like Java methods without fields, classes, statements, ...)
- Syntax: fun x0 (x1 : t1, ..., xn : tn) = e
- Typing rules:
  1. Context for e is (the function’s context extended with) x1:t1, ..., xn:tn *and* :
  2. x0 : (t1 * ... * tn) -> t where :
  3. e has type t in this context
- (This “definition” is circular because functions can call themselves and the type-checker “guessed” t.)
- (It turns out in ML there is always a “best guess” and the type-checker can always “make that guess”.)

Function applications (aka calls)

- Syntax: e0 (e1,...,en) (parens optional for one argument)
- Typing rules (all in the application’s context):
  1. e0 must have some type (t1 * ... * tn) -> t
  2. ei must have type ti (for i=1, ..., i=n)
  3. e0 (e1,...,en) has type t
- Evaluation rules:
  1. e0 evaluates to a function f in the application’s environment
  2. ei evaluates to value vi in the application’s environment
  3. result is f ’s body evaluated in an environment extended to bind xi to vi (for i=1, ..., i=n).
- (“an environment” is actually the environment where f was defined)