Standard ML

- Functional language:
  - evaluation of **expressions**
  - no (little) use of **assignments** and **side effects**
  - **composition** of functions (instead of ;)
  - **recursive** functions (instead of while loops)
  - heavy use of **lists** (instead of arrays)

  Note: **ML** actually has assignments (through references), ; operator, while loops and arrays, to be used in **very specific** situations. Some other functional languages (like Haskell) do not.

- Strongly typed language:
  - everything is a **value** (no variables; functions and procedures *are* values)
  - every value has a **type** (type checking at compile time)
  - **no** runtime type errors possible (as opposed to Java's `ClassCastException`)

- ML constant types
  - Type defines a set of values that can be used. (expressions must have type consistency)
- Fundamental types in ML
  - Integer – a string of digits (0-9), optionally preceded by a tilde (`~`) to indicate negative
  - Real – string of digits (possibly preceded by `~`), followed by at least one of a decimal point and one or more digits the letter E, followed by an integer
  - Boolean – true or false
  - Character – Ascii characters, e.g., `#"A"`
  - String – a sequence of characters enclosed with double-quotes e.g., `"abc"`
- Identifiers
  - The `val` statement associates an identifier with a value. Type consistency is maintained.

- Arithmetic Operators
  - Operator precedences: unary minus -> multiplicative operators -> additive operators
  - Parentheses has the highest precedence
- String operator
  - (^) for string concatenation
- Comparison operator
  - `=, <, >, <=, >=, <>`
  - Reals can not be compared using = and <>
- logic operators
  - `andalso, orelse, not`
- The if-then-else expressions

```
if 1<2 then 7 else 10;
```

- Tuples
  - A tuple groups sets of values. A tuple is an ordered set of data values; the values do not have to be the same type.
  - A tuple is surround by parentheses, and items are separated by commas, e.g. `val t = (4, 5.0, "six")`
  - The type of the tuple is the ordered types of its values, with a * in between.
  - access components of tuples: `#1(t)`
- Lists
  - A list of elements with same types.
  - A list is surrounded by square brackets, items are separated by commas, e.g. `val t = [4, 5, 6]`
  - Empty list: `[]` or `nil`
  - List operations: `null`, `hd`, `tl`.
  - List concatenation: `@`, or `::` (cons)

- Types are **polymorphic**

```
'a                          (any type)
''a                         (any type with equality)
'a * 'b
'a * 'a
'a * int * string
'a -> 'b
string -> int
'a -> unit
'a list
(('a -> 'b) * 'a) list -> 'b list
```

- Type inference mechanism:

  Input: `fun apply l = ...`

  Output: `(('a -> 'b) * 'a) list -> 'b list`

```
- apply [(square, 3), (inv, 1), (inv, 2), (square, 2)];
val it = [9.0,1.0,0.5,4.0] : real list
- apply [(strlen, "123"), (parse, "123")];
val it = [3,123] : int list
```

SML "New Jersey"
(used in the textbook)

- Interactive compiler / interpreter
  - available as /usr/local/bin/sml on CS cluster
  - available for Unix and Windows:
    http://www.smlnj.org/
  - documentation on standard library:
    http://www.standardml.org/Basis/
  - version of reference:
    Standard ML of New Jersey v110.75 [built: Thu Jan 24 09:10:50 2013]

- sml-mode for emacs users
  http://www.smlnj.org/doc/Emacs/sml-mode.html

SML Interaction: /usr/local/bin/sml

```
Standard ML of New Jersey v110.75 [built: Thu Jan 24 09:10:50 2013]
- 42;
val it = 42 : int
- 5 * ~1;
val it = ~5 : int
- [1,2,3];
val it = [1,2,3] : int list
- [];
val it = [] : 'a list
- ("foo", [2.3]);
val it = ("foo",[2.3]) : string * real list
- #2 it;
val it = [2.3] : real list
- val l = it;
val l = [2.3] : real list
- hd;
val it = fn : 'a list -> 'a
- hd l;
val it = 2.3 : real
- (4, []);
val it = <poly-record> : int * 'a list
- (4, []:int list);
val it = (4,[]) : int * int list
- (4, []): int * int list;
val it = (4,[]) : int * int list
- if "foo" < "Foo" then 10 else 20;
val it = 20 : int
- List.map;
val it = fn : ('a -> 'b) -> 'a list -> 'b list
```

Common Errors

```
- 1.2 > 2;
stdIn:1.1-259.5 Error: operator and operand don't agree
[literal]
  operator domain: real * real
  operand:         real * int
  in expression:
    1.2 > 2
- if hd l > 1.0 then "big" else 0;
stdIn:1.1-259.29 Error: types of rules don't agree
[literal]
  earlier rule(s): bool -> string
  this rule: bool -> int
  in rule:
    false => 0
- hd = tl;
stdIn:1.1-259.5 Error: operator and operand don't agree
[equality type required]
  operator domain: ''Z * ''Z
  operand:         ('Y list -> 'Y) * ('X list -> 'X list)
  in expression:
    hd = tl
- false andalso "true";
stdIn:1.1-259.18 Error: types of rules don't agree
[tycon mismatch]
  earlier rule(s): bool -> string
  this rule: bool -> bool
  in rule:
    false => false
```

```
- if hd l > 0 orelse "foo"="Foo" then 5;
= ;
= ;
= ;
stdIn:259.35-261.2 Error: syntax error: deleting  SEMICOLON
SEMICOLON SEMICOLON
- hd 2;
stdIn:44.1-44.5 Error: operator and operand don't agree [literal]
  operator domain: 'Z list
  operand:         int
  in expression:
    hd 2
- #1 [23];
stdIn:1.1-2.3 Error: operator and operand don't agree [type mismatch]
  operator domain: 1:'Y; 'Z
  operand:         int list
  in expression:
    (fn 1=1,... => 1) (23 :: nil)
```

Syntax

```
f x
```

`f x` is application of function `f` to parameter `x`
`f(x)` is the same thing
`f x+1` is `(f x) + 1` (function application has high precedence)
`f g(x)` is `(f g) x` (where `(f g)` is a function)
`f(g(x))` is `f` applied to `g(x)`
`f(g x)` is the same thing

If `f` has type `int -> int -> int` (which stands for `int -> (int -> int)`)
`f 2` (or `f(2)`) has type `int -> int`
`f 2 4` (or `f(2) 4`, or `(f 2) 4`) has type `int`

Note: ML functions always have **exactly one argument**

```
- 5 + 3;
val it = 8 : int
- op +;
val it = fn : int * int -> int
- op + (5,3);
val it = 8 : int
- op =;
val it = fn : ''a * ''a -> bool
- op ::;
val it = fn : 'a * 'a list -> 'a list
- #"a" :: [#"b"];
val it = [#"a",#"b"] : char list
- hd (tl [1,2,3,4]);
val it = 2 : int
- op o;
val it = fn : ('a -> 'b) * ('c -> 'a) -> 'c -> 'b
- (hd o tl) [1,2,3,4];
val it = 2 : int
- explode "CS-671";
val it = [#"C",#"S",#"-",#"6",#"7",#"1"] : char list
- rev it;
val it = [#"1",#"7",#"6",#"-",#"S",#"C"] : char list
- implode it;
val it = "176-SC" : string
- implode (rev (explode "CS-671"));
val it = "176-SC" : string
- (implode o rev o explode) "CS-671";
val it = "176-SC" : string
```

Defining Functions

```
- fun f x = x;
val f = fn : 'a -> 'a
- fun g x = x+1;
val g = fn : int -> int
- fun h x = x+1.0;
val h = fn : real -> real
- fun revString s = implode (rev (explode s));
val revString = fn : string -> string
- val revString = implode o rev o explode;
val revString = fn : string -> string
- val revString = fn s => implode (rev (explode s));
val revString = fn : string -> string
- revString "New Jersey";
val it = "yesreJ weN" : string
- fun firstChar s = hd (explode s);
val firstChar = fn : string -> char
- firstChar;
val it = fn : string -> char
- firstChar "SML";
val it = #"S" : char
- fun chop s = implode (tl (explode s));
val chop = fn : string -> string
- chop "SML";
val it = "ML" : string
- chop it;
val it = "L" : string
- chop it;
val it = "" : string
- chop it;
uncaught exception Empty  raised at: boot/list.sml:37.38-37.43
```

```
- fun even n = if n mod 2 = 0 then true else false;
val even = fn : int -> bool
- fun even n = n mod 2 = 0;
val even = fn : int -> bool
- val odd = not o even;
val odd = fn : int -> bool
- odd 42;
val it = false : bool
- fun nonNull n = n <> 0;
val nonNull = fn : int -> bool
- nonNull 10;
val it = true : bool
- nonNull 0;
val it = false : bool
- fun prec n = if n > 0 then n-1 else n;
val prec = fn : int -> int
- prec 12;
val it = 11 : int
- prec 0;
val it = 0 : int
- prec ~3;
val it = ~3 : int
```

```
- fun first (x,y) = x;
val first = fn : 'a * 'b -> 'a
- fun first (x,_) = x;
val first = fn : 'a * 'b -> 'a
- first("SML", "Java");
val it = "SML" : string
- #1("SML", "Java");
val it = "SML" : string
- #2("SML", "Java");
val it = "Java" : string
- #3("SML", "Java");
stdIn:312.1-312.18 Error: operator and operand don't agree
[record labels]
  operator domain: 3:'Y; 'Z
  operand:         string * string
  in expression:
    (fn 3=3,... => 3) ("SML","Java")
- fun swap (x,y) = (y,x);
val swap = fn : 'a * 'b -> 'b * 'a
- fun id p = swap (swap p);
val id = fn : 'a * 'b -> 'a * 'b
- fun id p = (swap o swap)p;
val id = fn : 'a * 'b -> 'a * 'b
- fun f(x,y) = swap(x, y+2);
val f = fn : 'a * int -> int * 'a
- fun swap2 ((x1,y1),(x2,y2)) = (swap (x2,y2), swap (x1,y1));
val swap2 = fn : ('a * 'b) * ('c * 'd) -> ('d * 'c) * ('b * 'a)
- fun swap2 (a,b) = (swap b, swap a);
val swap2 = fn : ('a * 'b) * ('c * 'd) -> ('d * 'c) * ('b * 'a)
```

```
- fun first3 l = (hd l, hd (hd l), hd (hd (hd l)));
val first3 = fn : 'a list list list -> 'a list list * 'a list * 'a
- fun first3 l = (hd l, hd (tl l), hd (tl (tl l)));
val first3 = fn : 'a list -> 'a * 'a * 'a
- first3 [1,2,3,4,5];
val it = (1,2,3) : int * int * int
- first3 [1,2];
uncaught exception Empty
  raised at: boot/list.sml:36.38-36.43
- fun first3 l = [hd l, hd (tl l), hd (tl (tl l))];
val first3 = fn : 'a list -> 'a list
- first3 [1,2,3,4,5];
val it = [1,2,3] : int list
- [1,2,3] @ [4,5,6,7,8];
val it = [1,2,3,4,5,6,7,8] : int list
- List.rev [1,2,3,4,5];
val it = [5,4,3,2,1] : int list
- List.nth ([1,2,3,4,5],3);
val it = 4 : int
```

```
- val l = [1,2,3];
val l = [1,2,3] : int list
- fun f x = x :: l;
val f = fn : int -> int list
- f 5;
val it = [5,1,2,3] : int list
- val l = ["foo", "bar"];
val l = ["foo","bar"] : string list
- f 5;
val it = [5,1,2,3] : int list
- fun g x = x+1;
val g = fn : int -> int
- fun f x = (x, g x);
val f = fn : int -> int * int
- g 3;
val it = 4 : int
- fun g x = x-1;
val g = fn : int -> int
- f 3;
val it = (3,4) : int * int
- fun length l = 0;
val length = fn : 'a -> int
- length [1,2,3];
val it = 0 : int
- List.length [1,2,3];
val it = 3 : int
```

```
- open List;
opening List
  datatype 'a list = :: of 'a * 'a list | nil
  exception Empty
  val null : 'a list -> bool
  val hd : 'a list -> 'a
  val tl : 'a list -> 'a list
  val last : 'a list -> 'a
  val getItem : 'a list -> ('a * 'a list) option
  val nth : 'a list * int -> 'a
  val take : 'a list * int -> 'a list
  val drop : 'a list * int -> 'a list
  val length : 'a list -> int
  val rev : 'a list -> 'a list
  val @ : 'a list * 'a list -> 'a list
  val concat : 'a list list -> 'a list
  val revAppend : 'a list * 'a list -> 'a list
  val app : ('a -> unit) -> 'a list -> unit
  val map : ('a -> 'b) -> 'a list -> 'b list
  val mapPartial : ('a -> 'b option) -> 'a list -> 'b list
  val find : ('a -> bool) -> 'a list -> 'a option
  val filter : ('a -> bool) -> 'a list -> 'a list
  val partition : ('a -> bool) -> 'a list -> 'a list * 'a list
  val foldr : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
  val foldl : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
  val exists : ('a -> bool) -> 'a list -> bool
  val all : ('a -> bool) -> 'a list -> bool
  val tabulate : int * (int -> 'a) -> 'a list
- length [1,2,3];
val it = 3 : int
```

Recursive Functions

```
- fun fact n =
= if n=0 then 1
= else n * fact(n-1);
val fact = fn : int -> int
- fact 10;
val it = 3628800 : int
```

```
- fun firstN (n,l) =
= if n=0 then []
= else hd l :: firstN (n-1, tl l);
val firstN = fn : int * 'a list -> 'a list
- firstN (4,[1,2,3,4,5,6,7]);
val it = [1,2,3,4] : int list
```

Recursive Functions

```
- fun append (l1, l2) =
= if null l1 then l2
= else hd l1 :: append(tl l1, l2);
val append = fn : 'a list * 'a list -> 'a list
- append([1,2,3], [4,5,6]);
val it = [1,2,3,4,5,6] : int list
- fun nth (l,n) =
= if n=0 then hd l
= else nth(tl l, n-1);
val nth = fn : 'a list * int -> 'a
- nth(["foo", "bar"], 1);
val it = "bar" : string
- nth(["foo", "bar"], 5);
uncaught exception Empty
  raised at: boot/list.sml:37.38-37.43
```

```
- fun concat l =
= if null l then []
= else append(hd l, concat (tl l));
val concat = fn : 'a list list -> 'a list
- concat [[1,2], [], [3,4,5], [6]];
val it = [1,2,3,4,5,6] : int list
- fun badRev l =
= if null l then l
= else badRev (tl l) @ [hd l];
val badRev = fn : 'a list -> 'a list
- badRev [1,2,3,4,5];
val it = [5,4,3,2,1] : int list
- val l10000 = (* some code *);
val l10000 = [1,2,3,4,5,6,7,8,9,10,11,12,...] : int list
- List.rev l10000;
val it = [10000,9999,9998,9997,9996,9995,9994,9993,9992,9991,9990,9989,...] : int list
- badRev l10000;
GC #0.0.0.1.84.4285:   (0 ms)
GC #0.0.0.2.85.4390:   (20 ms)
GC #0.0.0.2.86.4490:   (0 ms)
GC #0.0.0.2.87.4582:   (0 ms)
GC #0.0.0.2.88.4670:   (0 ms)
GC #0.0.0.2.89.4752:   (0 ms)
GC #0.0.0.2.90.4829:   (10 ms)
GC #0.0.0.2.91.4898:   (0 ms)
...
GC #0.0.0.2.155.7801:   (0 ms)
GC #0.0.0.2.156.7809:   (0 ms)
val it = [10000,9999,9998,9997,9996,9995,9994,9993,9992,9991,9990,9989,...]
  : int list
```

```
- fun f x = x+1;
val f = fn : int -> int
- (f 5; f 10);
val it = 11 : int
- print;
val it = fn : string -> unit
- (print "foo"; print "bar");
foobarval it = () : unit
- op ^;
val it = fn : string * string -> string
- fun println x = print (x ^ "\n");
val println = fn : string -> unit
- println "foo";
foo
val it = () : unit
- fun f x = (print (Int.toString x); print "\n"; x+1);
val f = fn : int -> int
- f 5;
5
val it = 6 : int
- fun prec x =
= if x > 0 then x-1
= else (print "WARNING: nonpositive number unchanged\n"; x);
val prec = fn : int -> int
- prec 3;
val it = 2 : int
- prec ~3;
WARNING: nonpositive number unchanged
val it = ~3 : int
- fun expl s = (if s="" then print "empty\n" else (); explode s);
val expl = fn : string -> char list
```

Configuration Variables

- SML does have **assignable variables** (references), which will be discussed later
- some can be used to **customize** the interactive environment

```
- Control.polyEqWarn;
val it = ref true : bool ref
- Control.polyEqWarn := false;
val it = () : unit
- val letters = explode "supercalafragelisticespialodocious";
val letters =
  [#"s",#"u",#"p",#"e",#"r",#"c",#"a",#"l",#"a",#"f",#"r",#"a",...]
  : char list
- Control.Print.printLength;
val it = ref 12 : int ref
- Control.Print.printLength := 50;
val it = () : unit
- letters;
val it =
  [#"s",#"u",#"p",#"e",#"r",#"c",#"a",#"l",#"a",#"f",#"r",#"a",#"g",#"e",#"l",
   #"i",#"s",#"t",#"i",#"c",#"e",#"s",#"p",#"i",#"a",#"l",#"o",#"d",#"o",#"c",
   #"i",#"o",#"u",#"s"] : char list
- Control.Print.linewidth;
val it = ref 79 : int ref
- Control.Print.linewidth := 96;
val it = () : unit
- letters;
val it =
  [#"s",#"u",#"p",#"e",#"r",#"c",#"a",#"l",#"a",#"f",#"r",#"a",#"g",#"e",#"l",#"i",#"s",#"t",
   #"i",#"c",#"e",#"s",#"p",#"i",#"a",#"l",#"o",#"d",#"o",#"c",#"i",#"o",#"u",#"s"] : char list
-
```

helper.sml

```
(* This file contains helper functions *)
fun intSquare x = x * x
fun square x = Real.* (x, x)
fun cube x : real = x * x * x

val pi = 3.1415926535
fun degreeToRadian x = x * pi / 180.0
```

main.sml

```
(* This is the main file *)
use "helper.sml"; (* <- this semicolon is required *)

val c = cube pi
val d = degreeToRadian 90.0

fun norm (x,y) = Math.sqrt (square x + square y)
```

/usr/local/bin/sml main.sml

```
> /usr/local/bin/sml main.sml
Standard ML of New Jersey v110.72 [built: Mon Mar 19 09:27:02 2012]
[opening main.sml]
[opening helper.sml]
[autoloading]
[library $SMLNJ-BASIS/basis.cm is stable]
[autoloading done]
val intSquare = fn : int -> int
val square = fn : real -> real
val cube = fn : real -> real
val pi = 3.1415926535 : real
val degreeToRadian = fn : real -> real
val it = () : unit
[autoloading]
[autoloading done]
val c = 31.0062766776 : real
val d = 1.57079632675 : real
val norm = fn : real * real -> real
- val a = degreeToRadian 90.0;
val a = 1.57079632675 : real
- val l = norm (1.0, 1.0);
val l = 1.41421356237 : real
- square;
val it = fn : real -> real
- square l;
val it = 2.0 : real
- ^D
>
```