ML

- functional programming style
- static typing
- type inference
- pattern matching
- polymorphism
- exceptions
- first-class modules

Basic datatypes

- unit (like void)

```
- ();
() : unit
```

- bool, int, real, string

```
- 7 > 5;
true:bool
- ~24+1;
~23:int
- 3.14159;
3.141519:real
- "hi there"
"hi there":string
```

- lists

```
- 3 :: 4 :: nil;
[3,4] : int list
- ( 3 :: nil) :: ( 4 :: 5 :: nil) :: nil;
[[3],[4,5]]: int list list
- [3, "hi there"];
... error ...
- nil;
[] : 'a list
```


Basic datatypes (cont)

- tuples

```
- ("wow",6);
("wow",6): string * int
- (1.0, 2, true, "s");
(1.0, 2, true, "s"): real*int*bool*string
```

- records

```
- {name="J",age=1};
{name="J",age=1}:{name:string,age:int}
- {name="J",age=1}={age=3 mod 2,name="J"};
true:bool
```


Bindings

```
- val x = 23;
val x = 17 : int
- local val x = 10
  in
    val u = 230 * x
    val z = (true,22 + x)
  end;
val u = 2300 : int
val z = (true,32) : bool * int
```

Functions

```
- fun succ(x) = x + 1;
val succ = fn : int -> int
- fn x=>x + 1;
fn : int -> int
- succ 8;
9 : int
- (fn x=>x+1)(8);
9 : int
```

Pattern Matching

Means of decomposing compound values

```
- val x=(false,17);
...
- val (false,w) = x;
val w = 17:int
...
- val (true,w) = x;
Failure: match

- val l=["1", "2", "3"];
...
- val hd::tl = l;
val hd = "1" : string
val tl = ["2", "3"] : string list
...
- val hd::_ = l;
val hd = "1" : string
```


Pattern Matching with Functions

Means of defining function behavior by cases:

```
- fun fib(0)  = 1
  | fib(1)   = 1
  | fib(n)   = fib(n-1) + fib(n-2);
val fib = fn : int -> int

- fun is_empty(nil) = true
  | is_empty(_::_)  = false;
val is_empty = fn : 'a list -> bool

- fun append(nil,l)= l
  | append(hd::tl,l) = hd :: append(tl,l);
val append = fn : ('a list * 'a list)->'a list
```

Something is special about these last two functions...

Polymorphic Functions

Some functions are general

- can be used on arguments of different types

E.g.

- `length`
- `append`
- `is_empty`
- `first_of_pair`

In Scheme, dynamic typing allows these to be defined easily

How to define such functions in a statically-typed language?

- can't in C (or have to use casts)
- in ML: allows functions to have polymorphic types

Polymorphic types

A polymorphic type contains one or more **type variables**

E.g.

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

To make a regular type, replace type variables with regular types

- each occurrence of a type variable must be replaced with same type

When calling a polymorphic function, caller knows real type

- replace type variable with regular type

`length([3,4,5])`
replaces `'a` with `int`

`length([(3.0,"hi"),(2.0,"there")])`
replaces `'a` with `int * string`

Type Inference

System infers types of expressions automatically. Works most of the time:

```
- fun f(x,y) = x + y;
Error: overloaded variable cannot be resolved:+
```

Sometimes you have to give it a hint:

```
- fun f(x:int, y) = x + y;
val f = fn : int * int -> int
```

...or...

```
- fun f(x, y):int = x + y;
val f = fn : int * int -> int
```


Exceptions

Means of handling unusual cases

```
- exception Head;
exception Head
- fun head(nil)  = raise Head
  | head(x::_)   = x;
val head = fn : 'a list -> 'a
...
- head(nil);
Failure: Head
```

Handling exceptions:

```
- fun head2(l) = head(l) handle Head => 0;
val head2 = fn : int list -> int
- head2(nil);
0 : int
```

Can even have arguments:

```
- exception IOError of int

... raise IOError(-3) ...
... handle IOError(code) ...
```