1 Introductory Syntax Examples

- Simple Expressions

```
- 1;
val it = 1 : int
- 2 * 3;
- 4 + ~1;
```

- Bindings

```
- val x = 2;
- val b = true;
- x + 7;
- val y = x + 7;
```

- Local Bindings

```
- let
=       val a = 2;
=       val b = a * a;
= in
=       b * b
= end;
val it = 16 : int
```

- Tuples

```
- (1,2);
val it = (1,2) : int * int
- val x = (1, true, 7.2);
- #2 x;
- #3 x;
- #4 x;
stdIn:34.1-34.5 Error: operator and operand don't agree [record labels]
  operator domain: {4:'Y; 'Z}
  operand:         int * bool * real
  in expression:
    (fn {4=4,...} => 4) x
```

- Records

```
- {foo = 1, bar = 2};
- #foo it;
```

- Lists

```
- [1,2,3];
val it = [1,2,3] : int list
- [];
val it = [] : 'a list
- val x = [1,2,3];
- 0::x;
- hd x;
- tl x;
- hd x;
- tl x;
- x @ [4,5];
```

- Conditional

```
- if true
= then 1
= else 2;
- 7 + (if true then 1 else 2);
```

- Case

```
- val x = 2;
- case x of
= 0 => 1
= | 1 => 2
= | 2 => 3
= | y => y * 2
= ;
- val z = [];
- case z of
= [] => "empty"
= | x::xs => "non-empty";
```

2 Functions

- Basic

```
- fun add1 x = x + 1;
- add1 2;
- fun add(x,y) = x + y;
- add(2,3);
- fun add x y = x + y;
- add 2 3;
- add (2,3);
- add 1;
- fun id x = x;
- id 1;
- id true;
```

- Patterns

```
- fun fact n =
=       if n = 0
=       then 1
=       else n * (fact (n - 1))
= ;
- fun fact 0 = 1
=   | fact n = n * (fact (n - 1))
= ;
- fun sum [] = 0
=   | sum (x::xs) = x + (sum xs)
= ;
```

- Mutual Definition

```
- fun odd 0 = false
=   | odd n = even (n - 1)
= and
=     even 0 = true
=   | even n = odd (n - 1)
= ;
```

- Explicit Types

```
- fun add (x:int) (y:int) = x + y;
- fun add (x:real) (y:real) = x + y;
- fun id (x:int) = x;
```

- Anonymous

```
- fn x => x + 1;
- (fn x => x + 1) 2;
```

3 Higher-Order Functions

```
- fun apply f x = f x;
- fun square x = x * x;
- apply square 3;
- fun sumsquares [] = 0
=   | sumsquares (x::xs) = (square x) + (sumsquares xs)
= ;
- sumsquares [1,2,3];
- fun sumF [] f = 0
=   | sumF (x::xs) f = (f x) + (sumF xs f)
= ;
- sumF [1,2,3] square;
- fun mymap f [] = []
=   | mymap f (x::xs) = (f x) :: (mymap f xs)
= ;
- mymap square [1,2,3];
- fun forall f [] = true
```

```
=   | forall f (x::xs) = (f x) andalso (forall f xs)
= ;
- forall (fn x => x = 0) [0,0,0];
- fun exists f [] = false
=   | exists f (x::xs) = (f x) orelse (exists f xs)
- fun fake_right f [] comb base = base
=   | fake_right f (x::xs) comb base = comb (f x, fake_right f xs comb base);
- fun fold_right f b [] = b
=   | fold_right f b (x::xs) = f (x, fold_right f b xs);
- fold_right (op +) 0 [1,2,3];
- foldr (op +) 0 [1,2,3];
```

4 Algebraic Data Types

- Basic

```
- datatype foo = Bar | Baz;
- val x = Bar;
- case x of
=       Bar => 1
=     | Baz => 2
= ;
- datatype union =
=       INT of int
=     | BOOL of bool
=     | TUPLE of (int * int)
= ;
- val y = TUPLE (3,2);
- case y of
=       (INT x) => x
=     | (BOOL true) => 1
=     | (BOOL false) => 0
=     | (TUPLE (a,b)) => a + b
= ;
```

- Parameterized

```
- datatype 'a pair = PAIR of 'a;
- PAIR(1,2);
- datatype 'a tree = nil | node of ('a * 'a tree * 'a tree);
fun insert x nil = node (x , nil, nil)
  | insert x (node(y, l, r)) =
         if x < y
         then node(y, insert x l, r)
         else node(y, l, insert x r)
;     (* darn, only for int trees because of the < *)


(* generalized *)
fun insert comp x nil = node (x , nil, nil)
  | insert comp x (node(y, l, r)) =
         if comp(x,y)
```

```
            then node(y, insert comp x l, r)
            else node(y, l, insert comp x r)
    ;

    - insert (op <);
```

5 Exceptions

```
exception Foo;
exception Bar of int;
raise Foo;
raise Bar 2;
fun f true = raise Foo
  | f false = raise Bar 2;
f true
   handle Foo => 1
        | Bar x => x
;
```

6 I/O

```
open TextIO;
print "Hello, world!\n";
val fstr = TextIO.openIn("file");
TextIO.input1 fstr;
TextIO.lookahead fstr;
(output (stdErr, "Hello ");
 output (stdErr, "-- this is an error\n"));
```