Pattern Matching

- Important in ML
- Used to bind variables

- val \((x, y) = (5 \text{ div} 2, 5 \text{ mod} 2)\);
  
  val \(x = 2 : \text{int}\)

  val \(y = 1 : \text{int}\)

- val \(\{a = x, b = y\} = \{b = 3, a = "one"\}\);
  
  val \(x = "one" : \text{string}\)

  val \(y = 3 : \text{int}\)
Pattern matching on lists

- val head::tail = [1,2,3];

stdin:67.1–67.25 Warning: binding not exhaustive

head :: tail = ...

val head = 1 : int

val tail = [2,3] : int list

- val [x, _, y] = [4, 5, 6];

val x = 4 : int

val y = 6 : int
Pattern matching in Functions

- Can do pattern matching in functions

```haskell
fun product [] : int = 1
| product (h::t) = h * product t;
```
Pattern matching in Functions

- fun oneTo 0 = [] | oneTo n = n::(oneTo(n-1));

val oneTo = fn : int -> int list

- oneTo 5;

val it = [5,4,3,2,1] : int list
Exercise

- Write a function to reverse a list

```haskell
fun reverse [] = [] | reverse(h::t) = ___
reverse(t) @ [h];
```
Function Composition

- The factorial function can be defined as
  
  ```
  fun fact n = product (oneTo n);
  ```

- Equivalent to writing
  
  ```
  val fact = product o oneTo
  ```

- The operator `o` is function composition
  (similar to `f o g`)
Sequential Declarations

- val x = 12;
val x = 12 : int
- val y = x + 2;
val y = 14 : int
Parallel (Simultaneous) Declarations

- val x = 2 and y = x + 3;
val x = 2 : int
val y = 15 : int
Mutual Recursion

- Take alternate elements example from a list

fun take [] = []

| take (h::t) = h::(skip t)

and skip [] = []

| skip (h::t) = take t;
Mutual Recursion

- take [1,2,3,4,5,6];
  val it = [1,3,5] : int list
- skip [1,2,3,4,5,6];
  val it = [2,4,6] : int list
Local Declarations

- **Syntax:** `let decl in exp end`
- **Factorial example** - `fact_local.sml`
- ** Allows naming intermediate values**
Hiding Declarations

- Declarations can be hidden with `local`
- Syntax: `local decl in decl-list end`
- Example: `fact_hidden.sml`
Quicksort Example

- Integer List quicksort example
- qsort.sml