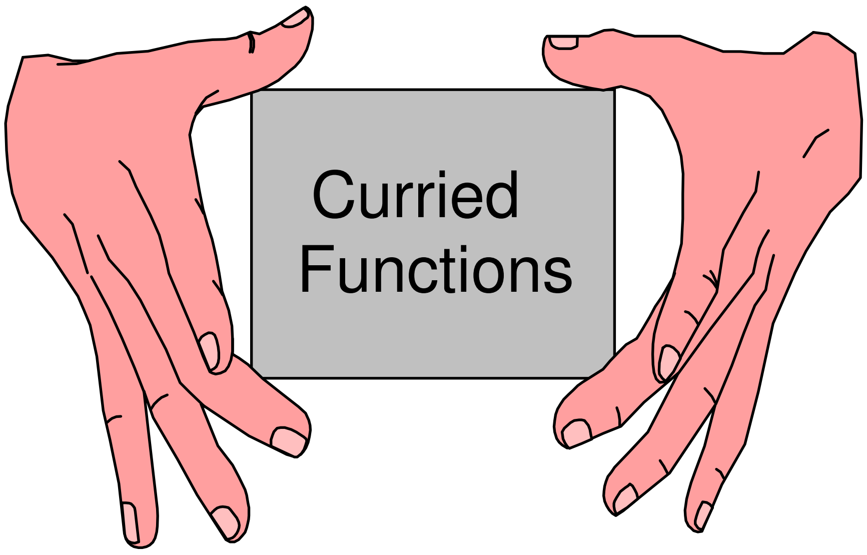
Standard ML



Side Note - Operators

- A function of two arguments can be treated using ***infix*** notation

```
fun d(x,y) = Math.sqrt(x*x+y*y);
val d = fn : real * real -> real
- d;
val it = fn : real * real -> real
- d(1.0,3.0);
val it = 3.16227766017 : real
```

- Convert to infix

```
- infix d;
infix d
- 1.0 d 3.0;
val it = 3.16227766017 : real
- 1.0 d 3.0 d 2.0 d 5.0;
val it = 6.2449979984 : real
```

Operators - a bit more

- Access

```
- d;
stdIn:40.1 Error: expression or pattern begins
   with infix identifier "d"
- op d;
val it = fn : real * real -> real
- op d(1.0,3.0);
val it = 3.16227766017 : real
```

Infix declaration can come before function definition

```
- infix d;
infix d
- fun x d y = Math.sqrt(x*x + y*y);
val d = fn : real * real -> real
```

ML Curried Functions.3

Curried Functions

- Any function of two arguments $(\alpha*\beta)\rightarrow\gamma$ can be expressed as a ***curried*** function of one argument $\alpha\rightarrow(\beta\rightarrow\gamma)$
- Example

```
- fun prefix (pre,post) = pre^post;
val prefix = fn : string * string -> string
```

- The curried version - using function as return value

```
- fun prefix pre = fn post => pre^post;
val prefix = fn : string -> string -> string
```

- Reminder: Arrow associate to the right. The next type is equivalent to the last one

```
val prefix = fn : string -> (string -> string)
```

ML Curried Functions.4

Partial Application

- You don't have to give the next arguments !

```
- prefix "Dr. ";
val it = fn : string -> string
- it "Tomer";
val it = "Dr. Tomer" : string
```

- As Always, functions are values …

```
- val doctorify = prefix "Dr. ";
val doctorify = fn : string -> string
- doctorify "Jackal";
val it = "Dr. Jackal" : string
```

- Observation

```
               prefix : string -> string -> string
          prefix "Dr. " : string -> string
prefix "Dr." "Tomer": string
```

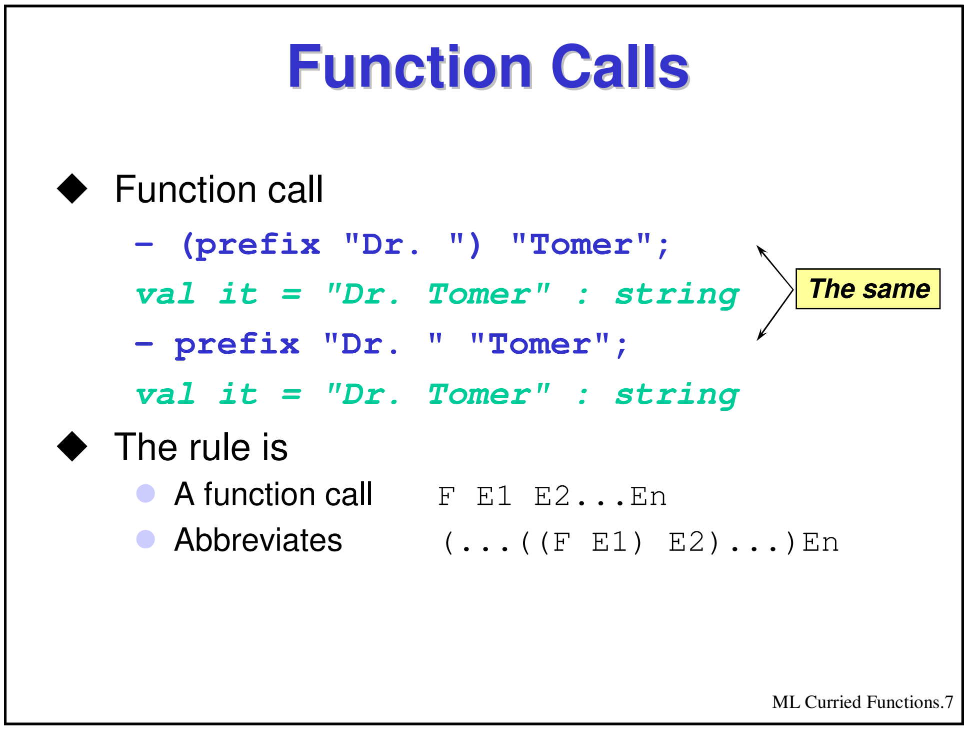
Curried - Syntactic Sugar

- Instead of using anonymous functions, A **fun** declaration may have several arguments, separated by spaces, for a curried function.

```
- fun prefix pre post = pre^post;
val prefix = fn: string -> string -> string
```

- Is equivalent to

```
- fun prefix pre = fn post => pre^post;
```



Function Calls

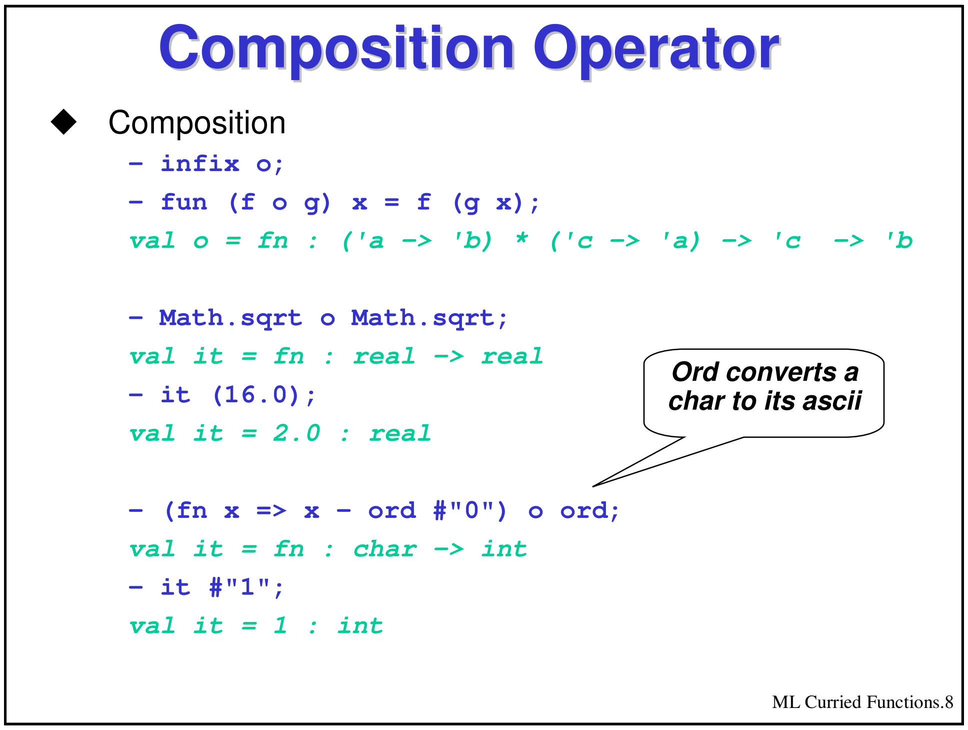
- Function call

```
- (prefix "Dr. ") "Tomer";
val it = "Dr. Tomer" : string
- prefix "Dr. " "Tomer";
val it = "Dr. Tomer" : string
```

The same

- The rule is
  - A function call `F E1 E2...En`
  - Abbreviates `(...((F E1) E2)...)En`

ML Curried Functions.7



Composition Operator

- Composition

```
- infix o;
- fun (f o g) x = f (g x);
val o = fn : ('a -> 'b) * ('c -> 'a) -> 'c  -> 'b

- Math.sqrt o Math.sqrt;
val it = fn : real -> real
- it (16.0);
val it = 2.0 : real

- (fn x => x - ord #"0") o ord;
val it = fn : char -> int
- it #"1";
val it = 1 : int
```

Ord converts a char to its ascii

ML Curried Functions.8

General-Purpose Functions - Sections

- Applying infix operator only on one operand - specific case

```
- fun add5 y = op+ (5, y);
val add5 = fn: int -> int
- add5 2;
val it = 7 : int
- fun mul5 y = op* (5, y);
val mul5 = fn: int -> int
```

- Now generalize the operator and operand

```
- fun something5 (f:int*int->int) y = f (5, y);
val something5 = fn: (int*int->int) -> int -> int
- val add5 = something5 op+;
val add5 = fn: int -> int
- fun intsec x (f:int*int->int) y = f(x,y);
val intsec =
   fn : int -> (int * int -> int) -> int -> int
```

ML Curried Functions.9

Recursive Curried Functions

- Recursion

```
- fun times n m = if m=0 then 0
=                 else n + times n (m-1);
val times = fn : int -> int -> int
- times 4 5;
val it = 20 : int
- val times_4 = times 4;
val times_4 = fn : int -> int
- times_4(8);
val it = 32 : int
```

- ***times_4*** is actually

```
fn m => if m=0 then 0
        else 4 + times 4 (m-1)
```

ML Curried Functions.10