


[F18] 15-150 SML Cheatsheet

Sheet made on best-effort basis. Italics for commentary. Updated as syntax is introduced. Piazza with suggestions / for clarifications.

<p><u>Basic types</u></p> <pre>() : unit 1 : int 1.0 : real #"a" : char "a" : string true : bool [1,2,3] : int list [[1],[2,3]] : int list list (1, "5", 0) : int * string * int (* I'm a comment *) (* Also, Ctrl+D to quit SML *)</pre>	<p><u>Operators and random stuff</u></p> <pre>+ - * work for int both sides or real both sides ~5 ⇒ negative five 3 div 2 ⇒ 1 8 mod 3 ⇒ 2 3.0 / 2.0 ⇒ 1.5 "cabby" ^ "para" ⇒ "cabbypara" not false ⇒ true "a" = "b" ⇒ false (1=2) orelse (1=1) ⇒ true (1=2) andalso (1=1) ⇒ false 1 :: (2 :: []) ⇒ [1,2], nil = [] [1,2] @ [3,4] ⇒ [1,2,3,4]</pre>	<p><u>Functions</u></p> <pre>(* fact : int -> int *) fun fact (0 : int) : int = 1 fact (n : int) : int = n * fact (n-1) (* add : int * int -> int *) fun add (x: int, y: int) : int = x + y</pre> <p><u>Anonymous functions</u></p> <pre>(fn (x:int) => x) : int -> int (fn (x:int, y:int) => x+y) : int * int -> int</pre>
<p><u>Deciding what to do</u></p> <pre>if isRaining then "Pittsburgh" else "California" case n of 0 => "zero" 1 => "one" _ => "fake number"</pre>	<p><u>Scoping</u></p> <pre>let val pi = 3.14 val radius = 1.50 fun area (pi : real, r : real) = pi * r * r in area (pi, radius) end</pre> <p><i>You can nest let .. in .. end statements.</i></p>	<p><u>Pattern matching</u></p> <pre>fun sum ([] : int list) : int = 0 sum (x :: xs) = x + sum (xs) case [1,2,3] of [] => "won't happen since it isn't empty list" (x::xs) => "x is 1, xs is [2,3]" val (a,b) = (1,2) sets a = 1, b = 2</pre> <p><i>To match anything, use the wildcard pattern: _</i></p>
	<p><u>Proof writing</u></p> <pre>e =>* e' e evals to e' in finitely many steps e =>^k e' e evals to e' in k steps => is evaluation, = is equality = : (1+1) = 2 and 2 = (1+1) => : (1+1) => 2 but 2 does not => (1+1)</pre> <p><u>Types of induction</u></p> <pre>Simple induction Strong induction Structural induction</pre>	<p><u>Style nitpicking - DON'T BE LEFT, BE RIGHT</u></p> <pre>if blah then true else false ⇒ blah case x of true => ⇒ case (x,y) (case y of false => ..) of (true,false) => .. snake_case ⇒ camelCase</pre> <p>To unpack tuples, prefer <code>let val (x,y) = a in x + y end</code> over a <u>single</u> case, <code>case a of (x, y) => x + y</code></p>