```
> rlwrap sml
Standard ML of New Jersey v110.74 [built: Thu Aug 16 11:25:45 2012]

- (* Tutorial on ML datatypes.
=    ML datatypes are like constructs called "unions" or "variants" in other
=    languages.  They group multiple possible values in one type. *)

- (* Define a data type for days of the week *)
- datatype day = Mon | Tue | Wed | Thu | Fri | Sat | Sun;
datatype day = Fri | Mon | Sat | Sun | Thu | Tue | Wed
- (* Datatype constructors are output in alphabetical order. *)

- (* A function that tests whether its argument is a weekend day *)
- fun isWeekend d = (d=Sat orelse d=Sun);
val isWeekend = fn : day -> bool

- isWeekend Wed;
val it = false : bool

- isWeekend Sun;
val it = true : bool

- (* Pattern matching and user-defined datatypes are particularly convenient
=    features of ML, and the two work well together. *)
- fun dayToInt Sun = 0
=   | dayToInt Mon = 1
=   | dayToInt Tue = 2
=   | dayToInt Wed = 3
=   | dayToInt Thu = 4
=   | dayToInt Fri = 5
=   | dayToInt Sat = 6;
val dayToInt = fn : day -> int

- dayToInt Fri;
val it = 5 : int

- (* Datatype values can wrap around other values. *)
- datatype shape = Circle of real  (* i.e., Circle(radius) *)
=                 | Rectangle of real*real (* i.e., Rectangle(height, width) *)
= ;
datatype shape = Circle of real | Rectangle of real * real

- val shape1 = Circle(1.0); (*circle with radius of 1.0 *)
val shape1 = Circle 1.0 : shape

- val shape2 = Rectangle(2.0*1.0, 3.0+0.0);
val shape2 = Rectangle (2.0,3.0) : shape

- (* A function to calculate a shape's area *)
- (* Note that we must put parentheses around entire shape parameters so SML/NJ
=    knows to treat the entire shape pattern as a single parameter *)
- fun area (Circle(r)) = r * r * Math.pi
    | area (Rectangle(h,w)) = h*w;
[autoloading]
[library $SMLNJ-BASIS/basis.cm is stable]
[autoloading done]
val area = fn : shape -> real

- area shape1;
val it = 3.14159265359 : real

- area shape2;
val it = 6.0 : real
```

```
- (* Datatypes can be recursive. *)
- (* This is a datatype for a binary search tree of ints. *)
- datatype bst = Empty | Node of int*bst*bst;
datatype bst = Empty | Node of int * bst * bst

- val t1 = Node(6, Node(4,Empty,Empty),
=                 Node(15, Node(11,Empty,Empty), Node(24,Empty,Empty)));
val t1 = Node (6,Node (4,Empty,Empty),Node (15,Node #,Node #)) : bst

- val t2 = Node(157,Empty,Empty);
val t2 = Node (157,Empty,Empty) : bst

- val t3 = Node(102,t1,t2);
val t3 = Node (102,Node (6,Node #,Node #),Node (157,Empty,Empty)) : bst

- (* Insert a list of ints into a binary search tree *)
- fun treeInsert nil tree = tree
=   | treeInsert (n::ns) Empty = treeInsert ns (Node(n,Empty,Empty))
=   | treeInsert (n::ns) (Node(i,left,right)) =
=       if n < i
=       then treeInsert ns (Node(i, treeInsert [n] left, right))
=       else treeInsert ns (Node(i, left, treeInsert [n] right));
val treeInsert = fn : int list -> bst -> bst

- val f = treeInsert [6,157,15,4,24,11];  (* partial instantiation *)
val f = fn : bst -> bst

- val t4 = f (Node(102,Empty,Empty));
val t4 = Node (102,Node (6,Node #,Node #),Node (157,Empty,Empty)) : bst
- (* t4 equals t3 *)

- (* Perform inorder traversal of a binary search tree to print all the nodes
=    in ascending order *)
- fun printInorder Empty = ()
=   | printInorder (Node(i,left,right)) =
=       (printInorder left; print(Int.toString i ^ " "); printInorder right);
[autoloading]
[autoloading done]
val printInorder = fn : bst -> unit

- printInorder t3;
4 6 11 15 24 102 157 val it = () : unit

- printInorder t4;
4 6 11 15 24 102 157 val it = () : unit

- (* Print a list of ints in ascending order *)
- fun sortInts L = printInorder (treeInsert L Empty);
val sortInts = fn : int list -> unit

- sortInts [6,8,2,9,4,0,2,5,3,9,1,0,4];
0 0 1 2 2 3 4 4 5 6 8 9 9 val it = () : unit
```