Recursion

So far, all the programs we can write will run very quickly! Roughly speaking, the number of steps they take can't be larger than the size of the program.

To obtain the computing power of ordinary languages, or Turing machines, etc., we need a way to **repeat** computations.

Most languages support **loops**. SML does have a loop construct:

```
while b do e
```

but this is completely useless in **pure** programs. (Why?)

Pure functional programs use **recursive** functions instead of iteration to achieve repetition. Crucial difference is that each recursive activation of a function provides **new** actual parameters to be bound to the function's formal parameters.

So, the function can compute something different each time it is called!

Recursion in SML

Recursive definitions are introduced by adding the keyword `rec` after the keyword `val`:

```
val rec f = fn n => if n = 0 then 0 else n * (f (n-1))
```

The effect of `rec` is to make sure that the right version of `f` is used; compare:

```
- val f = fn n => if n = 0 then 0 else n * (f (n-1));
std_in:2.43 Error: unbound variable or constructor: f
```

The expression on the RHS of a `val rec` **must** be a function (`fn`) expression.

The form

```
fun f x = e
```

is actually a derived form for

```
val rec f = fn x => e
```

Recursion is not hard

"To understand recursion
first you really have
to understand recursion
first you really have ..."

Recursive definition is one that mentions the thing being defined in the body of the definition.

- Define "ancestor."
- Factorial.
- Insertion sort.
- Quick sort.
- Towers of Hanoi.

Key idea: Recursive calls must be on some "smaller" or "simpler" argument.

Key idea: There must be one or more base cases to terminate the recursion.

Note direct correspondence to **induction**.

More Recursion on Integers

Fibonacci numbers:

```
fun fib n =
  if n < 2 then 1 else fib(n-1) + fib(n-2)
```

Traditional "for" loops must be done with recursions.

Sum up the first 10 integers:

```
val sum1to10 =
  let fun sumup n =
            if n = 10 then 10
            else n + (sumup (n+1))
  in sumup 1
  end
```

Sum up the first 20 Fibonacci numbers:

```
val fibsum1to20 =
  let fun sumup n =
            if n = 20 then fib 20
            else (fib n) + (sumup (n+1))
  in sumup 1
  end
```

There's a pattern here: we'll abstract soon...

Recursion on Lists

Computing the maximum of a list of integers:

```
- fun maxlist c =
  case c of
    nil => 0
  | (h::t) => max(h,maxlist t);
> val maxlist = fn : int list -> int
- maxlist [1,3,2];
> val it = 3 : int
```

The need to case over a function argument is so common that there is a special derived form that combines functions and case expressions into one, e.g.

```
fun maxlist nil = 0
  | maxlist (h::t) = max(h,maxlist t)
```

Insertion sort:

```
fun insert (x:int,c) =
  case c of
    nil => [x]
 | h::t => if x < h then x::c else h::(insert (x,t))
fun sort nil = nil
  | sort (h::t) = insert (h,sort t)
```

Remember the key ideas!

The ultimate in useless functions:

```
fun f x = f x
```

("I yam what I yam." - Popeye)

Nearly as bad:

```
fun f x = f (x-1)
```

Remembering to define the base case(s) is crucial for correctness and termination.

Also helps us understand "how recursion works."

If the argument is a data constructor, SML helps us remember base cases:

```
- fun f (h::t) = h+1;
std_in:0.0-0.0 Warning: match nonexhaustive
          h :: t => ...
val f = fn : int list -> int
```

Building Lists

Recursive structures must also be **built** by recursion:

```
- val fibbs1to10 =
  let fun f n =
        if n = 10 then [(10,fib 10)]
        else (n,fib n)::(f (n+1))
  in f 1
  end
val fibbs1to10 =
  [(1,1),(2,2),(3,3),(4,5),(5,8),(6,13),
   (7,21),(8,34),(9,55),(10,89)]
  : (int * int) list
```

Multiple base cases

Problem: Given a list of integers, return the same list with zeroes inserted between each.

```
- fun f(nil) = nil
=   | f(h::nil) = h::nil
=   | f(h::t) = h::0::(f t);
val f = fn : int list -> int list
- f [1,2,3];
val it = [1,0,2,0,3] : int list
- f [1,2];
val it = [1,0,2] : int list
- f [1];
val it = [1] : int list
- f [];
val it = [] : int list
```

"Theorem: All cats are the same color."