0

Evaluation Strategy

Strict (or eager) evaluation.

Also known as call-by-value

Given an expression, which is a function application

 $f(E_1,\ldots,E_n)$

evaluate E_1, \ldots, E_n and then apply f to the resulting values.

Call-by-name:

Substitute the expressions E_1, \ldots, E_n into the definition of f and then evaluate the resulting expression.

Anuj Dawar

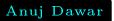
Lazy Evaluation

Also known as call-by-need.

Like call-by-name, but sub-expressions that appear more than once are not copied. Pointers are used instead.

Potentially more efficient, but difficult to implement.

Standard ML uses strict evaluation.





A list is an ordered collection (of any length) of elements of the same type

```
- [1,2,4];
> val it = [1, 2, 4] : int list
- ["a" , "", "abc", "a"];
> val it = . . . : string list
- [[1],[],[2,3]];
> val it = . . . : int list list
- [];
> val it = [] : 'a list
- 1::[2,3];
> val it = [1, 2, 3] : int list
```



There are two kinds of list:

nil or [] is the empty list h::t is the list with head h and tail t :: is an infix operator of type fn : 'a * 'a list -> 'a list $[x_1, \ldots, x_n]$ is shorthand for $x_1::(\cdots(x_n::nil)\cdots)$

Anuj Dawar

0

Built-in Functions 1

null

fn : 'a list -> bool
determines if a list is empty

hd

fn : 'a list -> 'a

gives the first element of the list

tl

fn : 'a list -> 'a list

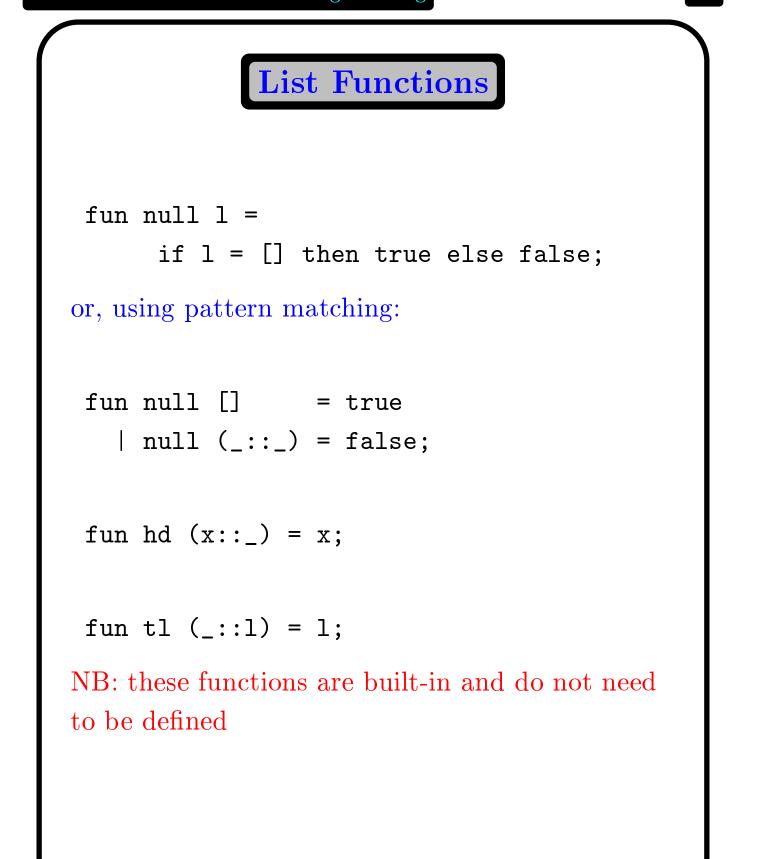
gives the tail of the list

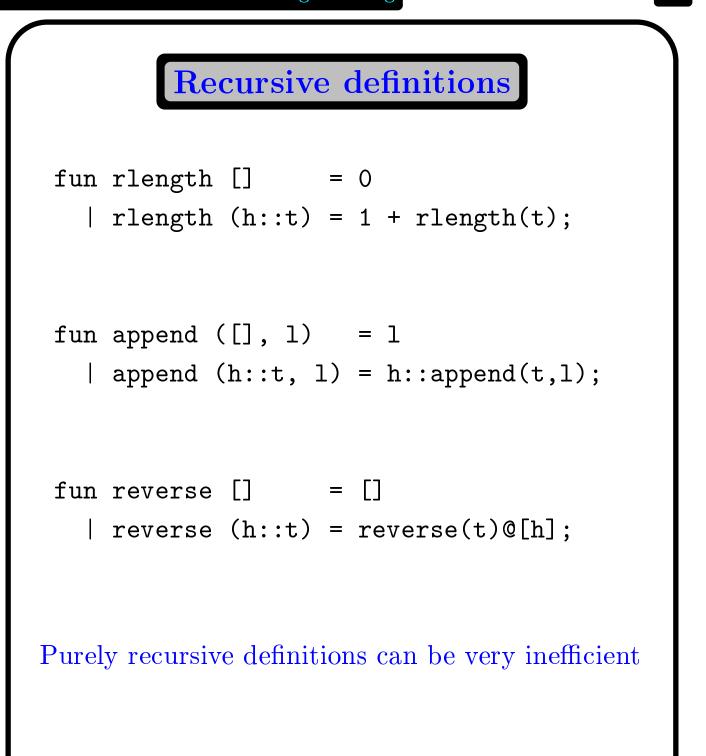
Anuj Dawar

Built-in Functions 2 length fn : 'a list -> int gives the number of elements in a list rev fn : 'a list -> 'a list gives the list in reverse order 0

appends two lists NB: infix!

Anuj Dawar



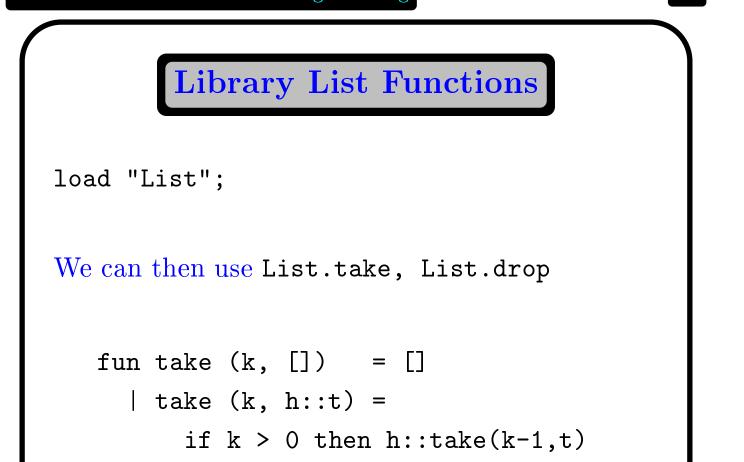


Anuj Dawar

Iterative Definitions

```
fun addlen ([],n) = n
  | addlen (h::t, n) = addlen (t, n+1);
fn : 'a list * int -> int
fun ilength l = addlen(1,0);
fun revto ([],1) = 1
  | revto (h::t, l) = revto (t, h::l);
fn : 'a list * 'a list -> 'a list
```

Anuj Dawar



else [];

Anuj Dawar