ML groups information using **tuples**
You can think of tuples as **records** of values that describe a particular object

Examples:

```
- val joe = (32,185,"married","pilot");
val joe = (32,185,"married","pilot") : int * int * string * string

- val circle = ((2.5,3.6),5.0);
val circle = ((2.5,3.6),5.0) : (real * real) * real
```
We can extract specific values from tuples using projections.

E.g., to retrieve the $i^{th}$ value from tuple $X$:

\[
\text{#}_i X
\]

- val joe = (32,185,”married”,”pilot”);
- val age = #1 joe;
- val profession = #4 joe;

- val circle = ((2.5,3.6),5.0);
- val radius = #2 circle;
- val x = #1 (#1 circle);
- val y = ?
ML supports another kind of tuple called a **list**

A list is a tuple where all elements are of the **same type**

- val oddlist = [ 1, 3, 5, 7, 9 ];

val oddlist = [ 1, 3, 5, 7, 9 ] : int list

- val nested = [(1,2),(3,4)];
- val nested = [[1,2],[3,4]];  
- val nested = [[1,2],[3,4,5]];  
- val nested = [(1,2),(3,4,5)];

what is the type of these constructions?
There exists a special list → the **empty** list: `[]` or `nil`

- `val mylist = [];

```
val mylist = [ ] : 'a list
```

means empty list type
null – tests whether a list is empty

- null([ ]);  
val it = true : bool

- null([1,2,3]);  
val it = false : bool
@ - concatenates two lists

- [1,2,3] @ [4,5,6];
  val it = [1,2,3,4,5,6] : int list

- [“not”] @ [“married”];
  val it = [“not”,”married”] : string list
:: - (cons operator) glue elements together to form a list
the last elements has to always be a (empty) list

- 1::2::3::[];
val it = [1,2,3] : int list

What is the domain of the :: operator?
ML – List Operators

- **hd** – (head operator) return the first element of a list

- hd([“one”, ”two”, ”three”]);
  val it = “one” : string

- hd([true]);
  val it = true : bool

- hd([ ]);  
  >>>??
**tl** – (tail operator) return the list *without* its first element

- `tl(["one", "two", "three"]);`
  val it = ["two", "three"] : string list

- `tl([true]);`
  val it = [ ] : bool list

- `tl([ ]);`
  >>??
ML – List Operators

(a) - val x = ["hello"] @ ["there"];

(b) - val x = ["hello" ^ "there"];

(c) - val joe = (32, 185, "married","pilot");
    - val jack = (29, 160, "not married", "cook");
    - val people = [joe, jack];

(d) - val l = [[1,2,3],["one","two","three"]]);

(e) - val x = [1,2,3];
    - val h = hd(x);
    - val t = tl(x);
    - val l = h :: t;
    l?

(f) – val y = 1::2::3;