Science may discover immortality, but it won't happen in the next eighty years.

You'll never find a programming language that frees you from the burden of clarifying your ideas.

You avoid your friend Mike because you're uncomfortably attracted to him.

Nice try, Mike. I get out of the well.

Aww.
curried versions of `containsP`, `map`  

`'a -> 'a list -> bool`

**input:** `x`  
**output:** procedure that takes in a list and returns true if `x` is in the list

```ml
let rec curried_containsP = function x ->
  function
    [] -> false
    | first::rest -> x=first || (curried_containsP x) rest
```

```ml
('a -> 'b) -> 'a list -> 'b list
```

```ml
let rec curried_map = fun f ->
  function
    [] -> []
    | a::rest -> f a::curried_map f rest
```
Curry with sugar

let rec curried_containsP = function x ->
  function
    [] -> false
  | first::rest -> x=first || (curried_containsP x) rest

let rec containsP = fun x mylist ->
  match mylist with
    [] -> false
  | first::rest -> x=first || containsP x rest

Looks like two arguments—but it is “syntactic sugar” for currying
Can even skip the fun!

let rec containsP x mylist = match mylist with
  [] -> false
  | first::rest -> x=first || containsP x rest

I recommend staying away from all this syntactic sugar. It rots your teeth.
In lab, you were introduced to another built-in parameterized type, a variant type, option.

(We could have defined it ourselves.)

```ml
type 'a option = Some of 'a | None
```

Useful for representing result when something can fail.

Let’s write a careful integer division that avoids divide by zero.

```ml
let careful_div = function
  | a,0 -> None
  | a,b -> Some (a /. b)
```

Type is `float * float -> float option`

Can use result with pattern-matching:

```ml
match careful_div (y, x) ->
  Some ratio -> atan ratio
  None -> pi /. 2
```
**input:** a pair consisting of a data object and a list

**output:** index in the list at which the data object is located, or … ?

```ocaml
let rec find_index = function
  _, [] -> ????????????
| x, hd :: tl, ->
  if hd == x then 1 else 1 + find_index (tl, item)
```

What should get returned when the search fails? Not an int!

But an int must be returned when success, and type system requires consisting output type.

Whenever multiple types must be bundled into a single type in Ocaml, use *variant type* option.
Storing intermediate results with `let` expression

Instead of

\[-b + \text{sort} (b^{*2} - 4 \times a \times c), \quad -b - \text{sort} (b^{*2} - 4 \times a \times c)\]

```reason
let sq_disc = sqrt(b^{*2} - 4*a*c) in
   -b + sq_disc, -b - sq_disc
```

let rec mergesort = function
    [] -> []
  | [a] -> [a]
  | mylist ->
    merge (mergesort (take (List.length mylist/2, mylist)),
           mergesort (drop (List.length mylist/2, mylist)))

let rec merge = function
    [], mylist -> mylist
  | mylist, [] -> mylist
  | a::restA, b::restB ->
    if a<b then a::merge (restA, b::restB)
    else b::merge (a::restA, restB)

let rec take = function
    0, _ -> []
  | n, a::rest ->
    a::take (n-1, rest)

let rec drop = function
    0, mylist -> mylist
  | n, a::rest -> drop (n-1, rest)
NOV 2nd @ 5PM
Mosaic+ general body meeting
3rd Floor Scili

Email us to join the listserv
mosaic.plus.brown@gmail.com