

# CS421 Lecture 5b: Option<sup>1</sup>

Mark Hills

`mhills@cs.uiuc.edu`

University of Illinois at Urbana-Champaign

June 8, 2006

---

<sup>1</sup>Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, and Elsa Gunter

## Option Example

# Objectives

Option is useful (really!), but maybe not obviously so. This lecture supplement provides a short example of using the option type. At the end of this supplement, you should:

- ▶ better understand why you would use option
- ▶ know how to write functions which match over option correctly

# The Example: Association Lists

Association lists provide a means of associating a keyword with a data value. This example will illustrate creating association lists using the option type.

# Association List: Type Definition

We will use strings for keys, but keep the other type of data open.

```
1 # type 'a alist = (string * 'a) list;;  
2 type 'a alist = (string * 'a) list
```

# Constructing Association Lists

We can build association lists by taking a list of keys and a list of values and zipping them together.

```
1 # let make_alist list1 (list2:( 'a list)) : ( 'a alist) =  
2   zip list1 list2;;  
3 val make_alist : string list -> 'a list -> 'a alist = <fun>
```

# Source Lists

These lists provide the source data...

```
1 # let l1 = ["Jim";"Steve";"Anne";"James";"Rachel"];;  
2 val l1 : string list = ["Jim"; "Steve"; "Anne";  
3                       "James"; "Rachel"]  
4  
5 # let l2 = [21;32;22;19;20];;  
6 val l2 : int list = [21; 32; 22; 19; 20]  
7  
8 # let l3 = ["English";"Math";"Computer Science";  
9           "Biology";"Physics"];;  
10 val l3 : string list =  
11    ["English"; "Math"; "Computer Science"; "Biology"; "Physics"]
```

# Some Sample ALists

...and these are the resulting association lists.

```
1 # let a1 = make_alist l1 l2;;
2 val a1 : int alist =
3   [("Jim", 21); ("Steve", 32); ("Anne", 22); ("James", 19);
4     ("Rachel", 20)]
5
6 # let a2 = make_alist l1 l3;;
7 val a2 : string alist =
8   [("Jim", "English"); ("Steve", "Math");
9     ("Anne", "Computer Science");
10    ("James", "Biology"); ("Rachel", "Physics")]
```

# Looking up AList Items

To look up an item in an association list, we should provide the key. If the key isn't there, though, what should we return? This is where option comes in handy...

```
1 # let rec lookup_alist key list =
2   match list with
3     | [] -> None
4     | (k,v)::vs ->
5       if key = k then Some v else lookup_alist key vs;;
6 val lookup_alist : 'a -> ('a * 'b) list -> 'b option = <fun>
7
8 # lookup_alist "Mark" a1;;
9 - : int option = None
10 # lookup_alist "James" a1;;
11 - : int option = Some 19
12 # lookup_alist "Rachel" a2;;
13 - : string option = Some "Physics"
```

# Using Option Values

Once we get option values back how do we use them? Pattern matching!

```
1 # let print_major name alist =
2   let result = lookup_alist name alist
3   in match result with
4     | None -> print_string "No record found!"
5     | Some s -> print_string (name ^ "'s major is " ^ s);;
6 val print_major :
7   string -> (string * string) list -> unit = <fun>
8
9 # print_major "Mark" a2;;
10 No record found!- : unit = ()
11
12 # print_major "Anne" a2;;
13 Anne's major is Computer Science- : unit = ()
```