Mapping

The aim of this practical session is to help you consider the dis/advantages of different symbolic data structures and different styles of symbolic processing. There is quite a bit to do here so do not expect to complete it in an hour.

The practical is based on three datastructures which capture information about students and their courses, these datastructures are as follows...

1. an association list with a primary ordering of course attribute and a secondary ordering of student, eg:
```
((course (alf . ncc) (sue . aat) (ralf . ncc) (nancy . ncc) ...)
 (year   (sue . 2)   (nancy . 1) (ralf . 3) ...)
 (age    (ralf . 22) (alf . 29) ...)
 ...etc...
)
```

2. a flat unordered list of name-value associations (NB: this is not what we have described as an association list because the name-value pairs are not sublisted), eg:
```
((alf age 29 year 2 course ncc)
 (sue course aat year 3 age 22)
 ...etc...
)
```

3. a set of tuples with an implicit ordering of name-age-course-year, eg:
```
((alf   29 ncc 2)
 (sue   22 aat 3)
 (nancy 23 ncc 1)
 ...etc...
)
```

For each of the datastructures above you are required to **attempt** to build a lookup function using each of the different approaches (i) - (iv) below. The lookup function should take a student name and field name as well as a datastructure as its args so (lookup 'alf 'course data) should return **ncc** for example.

(i) using the matcher
(ii) using a lisp mapping function (like find-if/remove-if-not & you can also use member) with some predicate or an anonymous lambda form.
(iii) using **->** but not using the matcher
(iv) writing a (recursive?) function without using the matcher or **->**

You may tackle the work in any order you choose but you should aim to attempt all sections. You should time yourself, spend a few minutes creating sample data then do not spend more than 35 mins on any one of the datastructure types.

For each of the approaches you try record the time taken and level of difficulty you experienced with (a) the problem solving and (b) the codeing. Classify the level of difficulty as Easy/Moderate/Hard/Impossible.

Use the attached sheet to record your results.

Code Style	Data Type	Time	Level of Difficulty			
matcher	1		Easy	Moderate	Hard	Impossible
	2		Easy	Moderate	Hard	Impossible
	3		Easy	Moderate	Hard	Impossible
mapping fn	1		Easy	Moderate	Hard	Impossible
	2		Easy	Moderate	Hard	Impossible
	3		Easy	Moderate	Hard	Impossible
using ->	1		Easy	Moderate	Hard	Impossible
	2		Easy	Moderate	Hard	Impossible
	3		Easy	Moderate	Hard	Impossible
recursion	1		Easy	Moderate	Hard	Impossible
	2		Easy	Moderate	Hard	Impossible
	3		Easy	Moderate	Hard	Impossible