mapping functions – intro

So far, when we have wanted to do something repeatedly, we have built recursive functions (or maybe you have found some of the iterative forms that Lisp uses). Lisp also has mapping functions.

Most mapping functions take 2 arguments: a function and a list, mapping functions use some strategy to apply the function over the list. **mapcar** is a mapping function which applies its function argument to each of the elements in its list argument & collects up the result...

```
> (mapcar #'1+ '(1 3 7 12 17))
(2 4 8 13 18)

> (mapcar #'first '((one 1)(two 2)(three 3)(four 4)(five 5)))
(one two three four five)
```

reduce is a mapping function which repeatedly applies its function argument accumulating a single result...

```
> (reduce #'+ '(1 3 7 12 17))
40
```

Note that you need to make sure the list is correctly structured...

```
> (reduce #'+ '((one 1)(two 2)(three 3)(four 4)(five 5)))
Error: `(one 1)' is not of the expected type `number'
```

You also need to make sure partial results are properly structured if you are using a function with reduce...

```
(defun add-2nd (a b)
  (+ (second a) (second b)))

> (add-2nd '(one 1) '(two 2))
3
> (reduce #'add-2nd '((one 1)(two 2)(three 3)(four 4)(five 5)))
Error: Attempt to take the cdr of 3 which is not listp.


(defun add-2nd-v2 (a b)
  (list 'dummy
        (+ (second a) (second b))
        ))

> (add-2nd-v2 '(one 1) '(two 2))
(dummy 3)
> (reduce #'add-2nd-v2 '((one 1)(two 2)(three 3)(four 4)(five 5)))
(dummy 15)
```

Sometimes it is possible to use the **:key** argument instead of writing an additional *helper* function...

```
> (reduce #'+ '((one 1)(two 2)(three 3)(four 4)(five 5)) :key
#'second)
15
```

Lisp supports various mapping functions – check them out in the online documentation. **remove-if** is a function that works as follows...

```
> (numberp 12)
t
> (numberp 'spam)
nil
> (remove-if #'numberp '(i ate 1 and he ate 3 too))
(i ate and he ate too)
> (remove-if-not #'numberp '(i ate 1 and he ate 3 too))
(1 3)
> (find-if #'numberp '(i ate 1 and he ate 3 too))
1
```

Note the example with remove-if-not. remove-if-not is (officially) deprecated ie: being phased out of use.

Often we want to use our own functions with Lisp's mappers (in preference to one that is already defined), eg...

```
(defparameter nouns '(cat rat mat bat))

(defun is-noun (word)
  (member word nouns))

> (remove-if #'is-noun '(the cat chased the rat with a bat))
(the chased the with a)
```

In some cases, the functions we want to use with mappers are only used once and/or rely on local values. In these cases we want to define functions in-line. We do this using **lambda**. **lambda** defines an anonymous, in-line function.

```
(defparameter determiners '(a the any all))

> (remove-if #'(lambda (word)
                          (member word determiners))
                        '(the cat chased the rat with a bat))
(cat chased rat with bat)


(defun filter (words sentence)
  (remove-if #'(lambda (word) (member word words))
             sentence))

> (filter determiners '(the cat chased the rat with a bat))
(cat chased rat with bat)
```

```
> (filter ($+ determiners nouns)
          '(the cat chased the rat with a bat))
```

A couple of other examples...

```
> (setf marks '(45 65 50 70))
(45 65 50 70)

> (reduce #'+ marks)    ;; the sum of marks
230

> (length marks)        ;; the number of marks
4

> (/ 230 4)             ;; the average (as a fraction)
115/2

> (round (/ 230 4))     ;; the average in a better form
58
-1/2

;; in-line average calculation for collections of marks
> (mapcar #'(lambda (lis)
               (round (/ (reduce #'+ lis)
                         (length lis))))
     '((45 65 50 70) (45 35 50 52) (68 71 56 58) (76 48 52 64)))
(58 46 63 60)
```