CS262 Artificial Intelligence Concepts

Worksheet 7

1 A Mapping Function

In addition to `do` and `loop`, the function `mapcar` can be used to perform iterative operations elegantly. To understand how it should be used, we can consider a situation where we need a function `list-add-one` that adds 1 to each number in a list and returns the resulting values in a new list.

The three functions below each do just this using `loop`, `do` and `mapcar` respectively.

```
(defun list-add-one (lis)
   (let ((newlist nil))
      (loop
         (cond ((null lis) (return newlist)))
         (setq newlist (append newlist (list (1+ (car lis)))))
         (setq lis (cdr lis)))))


(defun list-add-one (lis)
   (do ((oldlis lis (cdr oldlis))
        (newlis nil (append newlis (list (1+ (car oldlis))))))
       ((null oldlis) newlis)))


(defun list-add-one (lis)
   (mapcar '1+ lis))
```

So all of these functions will return `'(2 3 4 5)` if passed `'(1 2 3 4)`, regardless of the ways in which they are coded.

It can be seen that `mapcar` takes two arguments. The first is the name of the function and the second argument is a list. The function given as the first argument is applied to every element of the list that is the second argument in succession and the result is put into a new list, which is returned by `mapcar`.

The function `mapcar` is just one of several *mapping functions* that are available in LISP. These functions map a function over a list (ie. they apply the function to each item in the list). It is important to identify which types of iteration can use `mapcar`. First we can only use it when iterating on a list, so if our iteration is controlled by a counter or input we will have to use a `do`. Second, we can use `mapcar` only when the output from the our function is also a list, for example, if you want to add up all negative numbers in a list you cannot use `mapcar` as the output is not a list.

Exercise 1

1. Define a function call `list-decrement` with one parameter that holds a list of numbers. The function subtracts 1 from each element in the parameter list and returns the result as a list (in the same order).

2. Define a function called `embed-lists` that has one parameter, which is a list. The function returns a new list in which each item of the original list has been embedded in a list. For example:

```
> (embed-lists '(a (b) cat))
((A) ((B)) (CAT))
>
```

2 Mapping Over Multiple Lists

In the example and exercises of the previous section we mapped functions that accept one argument; but we can map functions that take more than one argument. To map a multiple-argument function, you have to provide a separate list for each argument. Then `mapcar` will apply the function to the successive elements of each list.

For example, suppose that we want to add together the corresponding elements of two lists and return a list of the resulting values. The following function performs that task:

```
(defun addlists (lis1 lis2)
   (mapcar '+ lis1 lis2))
```

```
> (addlists '(1 2 3 4) '(0 2 3 4))
(1 4 6 8)
>
```

In this example the function + is applied in succession to the values 1 and 0, 2 and 2, 3 and 3, and finally 4 and 4, with the results of each calculation being saved in the list that is returned.

Exercise 2

Define a function called `pair-up`, which takes two list of names and returns a new list, in which the corresponding names have been placed in embedded lists. For example:

```
> (pair-up '(john bob sam) '(mary alice karen))
((JOHN MARY)(BOB ALICE)(SAM KAREN))
>
```