Tutorial Sheet: House database functions and some expertise

Christopher Buckingham
Aston University

This lab will create some functions for operating on a list of houses that an estate agent might want. For example, the estate agent will want to find all the houses with a certain number of rooms, or a range of rooms. Selecting the most appropriate houses will depend on more subtle ratings of the house features and how well they match your own requirements.

This lab will introduce some more lisp functions and you can now obtain a full set from BlackBoard in the *Everything you need for lisp on this module* folder. For now, you need to know how the `dolist` function and some conditional functions work first. You also need to know how to create local variables within a function using the `let` function.

1 dolist

Function which allows you to apply operations to each member of the list.

```
(dolist (var list-form)
  body-form*)
```

1. `var` is the variable name that has each element of the list returned by `list-form` passed to it.
2. `dolist` is a very rare function in that it allows a variable to be set up and used without being declared first.
3. the variable only has a scope within the body-form of the function.
4. `list-form` must return a list.
5. `body-form` defines what happens to each member of the list: it operates on `var` in some way.

```
;; takes a list of numbers and doubles them, returning the new list
;;
(defun double (numbers)
  (let ((double-list nil))
    (dolist (element numbers double-list)
            (setf double-list (cons (* 2 element) double-list)))))
```

Note:

1. `let` is the function that sets up a local variable (see later).
2. element is the variable name given to each member of the list as the list is processed

3. numbers is the list being processed
4. `cons` is a function that adds its first argument to the second argument and returns the new list with the first argument on the front (the head of the list).
   (a) Note that `cons` does NOT change any of its arguments, so `double-list` does not have the new element on its head just by running the `cons` function.
   (b) To save the new list back into `double-list`, you need to explicitly set `double-list` to the new value, which is what `setf` is doing in that same form.
   (c) In general, lisp functions do not change their arguments . . . because they are functions. You can, of course, write functions that *do* change the arguments, which is a side effect of the function, but you should be careful about doing this and always state that behaviour clearly in the comment explaining the function's operation.
5. double-list is the result to be returned by the dolist function
6. If there is no result form, then the dolist function returns nil

Let's see how it works:

```
>(double '(1 2 3))
(6 4 2)
```

If we leave off the result form (i.e. double-list) so that the first line looks like *dolist (element l)*, then dolist and thus also the function returns nil.

If you want to return the list in the same order, then you have to use the `reverse` function call on the result:

```
(reverse double-list)
```

Try it and see.

2 The `let` function

`let` is a function which allows you to declare local variables. So double-list in the previous `double` function is a local variable which only exists within the `let` statement.

```
(let ((<parameter 1><initial value 1>)
      ...
      (<parameter n><initial value n>)
  <form 1>
  ...
  <form n>))
```

All the forms indented within the `let` function can use the variables declared by the `let`. If you try to access the variable outside the `let`, you will get an error because the "scope" of the variable is within the `let` function only.

2.1 Global and local variables

Global variables differ from local ones by remaining in the lisp environment after a program has finished executing. They should be distinguished from local ones by having an asterisk in front of their name: *<*global-variable*>* and should be used **as rarely as possible**. For intelligent knowledge-based systems, they would be used for the knowledge structure, which is very large and everything requires it. Hence our house database would be a global. During testing, it is also useful to have some global test variables but these will not be part of the final program.

3 The *if* choice function

```
(if <test> <then form> <else form>)

(defun biggest (x y)
  (if (> x y)
      x
    y))

>(biggest 3 5)
5

>(biggest 5 3)
5
```

Note that there is only one form for the if part and one form for the else part. So if you want to carry out multiple steps, you have to use a function called `progn` which "bags" together many forms inside one form. See what happens when you try to put two forms into an `if` function without enclosing them within `progn`:

```
;; Note: else part of the if statement consists of two forms, not just
;; the one allowed.
;;
(defun bigger (x y)
  (if (> x y)
      (format t "~%first argument is bigger")
    (format t "~%second argument is bigger")
    (format t "~%and here is a second form for the else bit~%")))

>(bigger 3 4)

Error: Too many arguments.
Error signalled by IF.
Broken at IF.  Type :H for Help.
```

The function *progn* can contain any number of forms:

```
(defun bigger (x y)
```

```
  (if (> x y)
      (format t "~%first argument is bigger")
    (progn (format t "~%second argument is bigger")
           (format t "~%and here is a second form for the else bit~%"))))

>(bigger 3 4)

second argument is bigger
and here is a second form for the else bit
NIL
```

progn returns the value of its **last** form whereas a similar function called *prog1* returns the value of its first form. So if we add a second form to the "then" part of the if statement using progn and make this form just a single number, it should be returned by the progn:

```
(defun bigger (x y)
  (if (> x y)
      (progn (format t "~%first argument is bigger")
             1)
    (progn (format t "~%second argument is bigger")
           (format t "~%and here is a second form for the else bit~%"))))

;; now call it in lisp

>(bigger 4 3)

first argument is bigger
1
```

4 When and unless

```
(when <test> <then forms>)

(unless <test> <else forms>)
```

Both *when* and *unless* can have any number of forms.
They return the value given by the last one.

5 Boolean operators

Your big document of Lisp functions (in the "labs" folder inside a folder called "Everything you need for lisp on this module") gives you all the boolean operators you will need but ones for this lab class are all self-explanatory and include: `=, <, >, >=, <=`. Try them out to see how they work. For example, how does `(> 7 7 4 2)` compare to `(>= 7 7 4 2)`?

Also, you will find `and` and `or` useful. The `and` boolean returns true if all the parameters following it are true; `or` returns true as soon as it finds ONE parameter true: `(and (= 2 2)(< 2 4)), (or (= 1 2)(< 4 8)(>= 8 9))`.

6 TASK: using these new functions

Suppose you have a list of students and their marks. Write a function called `sort-students` that takes the student list and two parameters, the bottom mark range and the top mark range. The function returns a list of students whose marks lie in between the mark range. The following code shows how it works for a particular list of students:

```
>(setf students '((dave 81)(dee 26)(dozy 58)(beaky 52)(mick 45)(tich 64)))

>(sort-students students 50 80)
>((dozy 58)(beaky 52)(tich 64))
```

7 Setting up a database of houses

So far, you have written some functions that operate on a single house. However, an estate agent or letting agency would have a large collection of houses on the books and will want to have functions that match particular accommodation with the requirements of a client. The next set of tasks will establish the database as a global list and will define useful functions for selecting particular properties from it.

Suppose you have set up a list of houses as follows:

```
(setf *house-db
      '(
;; first house
((name red-house)
 (type terrace)
 (cost 175000)
 (rooms ((kitchen ((dimensions (20 12))(appliances (cooker fridge dishwasher))))
 (bathroom ((dimensions (10 14))(features (bath washbasin toilet))))
 (bedrooms ((bedroom1 ((dimensions (15 21))(features (double-bed washbasin))))))))
 (garden (pond lawn shed)))
;; next house
((name blue-house)
 (type terrace)
 (cost 210000)
 (rooms ((kitchen ((dimensions (20 12))(appliances (cooker fridge dishwasher))))
 (bathroom ((dimensions (10 14))(features (bath washbasin toilet))))
 (bedrooms ((bedroom1 ((dimensions (8 20))(features (chandelier))))
    (bedroom2 ((dimensions (8 20))(features (chandelier))))
    (bedroom3 ((dimensions (8 20))(features (chandelier))))
    (bedroom4 ((dimensions (15 21))(features (double-bed washbasin))))))))
 (garden (pond lawn shed)))

.............. AND SO ON

((name green-house)
 (type glass))))
```

This is just an example structure and you should choose one of your own that elaborates on it. To speed up the process, download the `house-db.lisp` file on blackboard in the tutorial area and add a property

or two, if you feel so inclined. The essential thing to remember is that the structure should represent the important descriptive attributes of a house and make it easy for you to search the database of houses for them.

7.1 Tasks

The tasks should be using functions you have created in previous tutorials such as get-rooms and get-room. The idea is to construct functions that work on accessing parts of the house that you are interested in and then combining them to do carry out useful functions for the knowledge-based system. For this tutorial, the functions will be used to select properties that match a user's requirements (e.g. number of bedrooms, houses with a garden).

TASK Set up a global variable in your program file that contains a list of individual properties (houses) held by the Estate/Letting Agent (ELA). In other words, create a database of houses along the lines of the example given above . . . and make it more interesting.

TASK Define a function that returns a list of all houses with a given number of rooms.

TASK Define a function that returns a list of all houses with a given number of bedrooms.

TASK Define a function that returns a list of all houses falling within a given price or rental range.

TASK In the previous lab, you were asked to rate houses based on the number of rooms they contain. Try producing a more sophisticated rating system based on more than one attribute. For example, rate them based on the number of rooms and the cost, with a weighted combination of the two.