An Expression Evaluator with Variables

In this assignment, your task is to create an expression evaluator in Scheme for arithmetic expressions containing variables.

Part 1: Association Lists

An association list is a list of pairs, where a pair is a 2-element list of the form `(key value)`. For example:

```
(define pets
   '((cat 1) (dog 1) (fish 1) (cat 2) (fish 2))
)
```

`pets` is an association list containing 5 pairs. Notice that both `cat` and `fish` appear twice as keys: association lists allow keys to occur more than once.

Implement and test the following functions:

- `(is-pair? x)` returns `#t` if `x` is a 2-element list, and `#f` otherwise. For example, `(is-pair? '(fish 1))` and `(is-pair? '(5 4))` return `#t`, but `(is-pair? '(fish 1 2))` and `(is-pair? 5)` return `#f`.

  The input `x` could be any Scheme value, not necessarily a list.

- `(is-alist? x)` returns `#t` if, and only if, `x` is an association list, i.e. a list of pairs. The empty list is an association list. For example, `(is-alist? pets)` and `(is-alist? '())` return `#t`, but `(is-alist? '((cat 1) (dog 1) (hamster)))` returns `#f`.

  The input `x` could be any Scheme value, not necessarily a list.

- `(get-all-pairs key lst)` returns the list of all pairs in `lst` that have `key` as their first element. If no pairs match, then the empty list is returned. For example, `(get-all-pairs 'cat pets)` returns `((cat 1) (cat 2))`, and `(get-all-pairs 'bird pets)` returns `'()`.

  If `lst` is not an association list, then use the `error` function to stop the program, e.g. by calling something like `(error "lst is not an association list")`.

- `(get-first-pair key lst)` returns the first pair in `lst` that has `key` as its first element. If no pair matches, then `'()` is returned. For example, `(get-all-pairs 'cat pets)` returns `(cat 1)`, while `(get-all-pairs 'bird pets)` returns `'()`.

  If `lst` is not an association list, then use the `error` function to stop the program, e.g. by calling something like `(error "lst is not an association list")`.

  One way to implement `get-first-pair` is by calling `(car (get-all-pairs key lst))`. However, that's inefficient for large lists, so implement it an a more efficient manner.

- `(del-all-pairs key lst)` returns an association list that is the same as `lst` but all pairs in `lst` whose first element is `key` have been removed. For example, `(del-all-pairs`

`'cat pets)` returns `((dog 1) (fish 1) (fish 2))`.

If `lst` is not an association list, then use the `error` function to stop the program, e.g. by calling something like `(error "lst is not an association list")`.

- `(del-first-pair key lst)` returns a list that is the same as `lst` but the first pair in `lst` whose first element is `key` has been removed. For example, `(del-first-pair 'cat pets)` returns `((dog 1) (fish 1) (cat 2) (fish 2))`.

  If `lst` is not an association list, then use the `error` function to stop the program, e.g. by calling something like `(error "lst is not an association list")`.

No special function is needed for adding a pair. Just use `cons`, e.g.:

```
=> (cons '(frog 2) pets)
((frog 2) (cat 1) (dog 1) (fish 1) (cat 2) (fish 2))
```

Implement all these functions as described with the *exact* headers: they will be tested individually.

> Note: Scheme already provides support for association lists with built-in functions such as `alist?`, `assq`, `assv`, etc. However, do not use any of those functions in this assignment. Implement your association list using only basic Scheme functions.

Part 2: An Expression Evaluator

In this part, your task is to implement a function called `myeval` that evaluates an infix arithmetic expression that might (or might not) have variables. The values for variables are stored in an association list that is passed to the evaluator.

Here are some examples:

```
=> (myeval '(2 + (3 * x))      ;; the expression
           '((x -1) (y 4))     ;; the environment
   )
-1

=> (myeval '(2 + (3 * 1))      ;; the expression
           '((x -1) (y 4))     ;; the environment
   )
5

=> (myeval '((m * a) - 0.1)    ;; the expression
           '((m 2.5) (a 2))    ;; the environment
   )
4.9

=> (myeval '(4 * (s * s))      ;; the expression
           '((q 3) (r 4))      ;; the environment
   )
;unknown variable              ;; call error if expression can't be evaluated
```

To make it easier for us to mark, your evaluator must be called exactly like this:

```
(myeval expr environment)
```

`expr` is an arithmetic expression (as defined below), and `environment` is an association list (see part 1) of the variables and their values that can appear in `expr`.

Here is an EBNF grammar for the expressions that `myeval` can evaluate:

```
expr = "(" expr "+" expr ")"
           | "(" expr "-" expr ")"
           | "(" expr "*" expr ")"
           | "(" expr "/" expr ")"
           | "(" expr "**" expr ")"  ;; e.g. (2 ** 3) is 8, (3 ** 3) is 27
           | var
           | number
  number = a Scheme number
  var    = a Scheme symbol
```

If you call `myeval` on an expression not generated by this grammar, or if the expression contains a variable not in the environment, then it should end with a call to the `error` function.

Marking Scheme

Part 1

- 6 marks for correctly implementing each of the given association list functions as described in Part 1. This includes both the behaviour of the functions, and the names and parameters of the functions.
- 2 marks for coding style. This includes consistent and sensible indentation, good variable names, good use of Scheme features, etc.

Part 2

- 10 marks for correctly evaluating all the expressions specified by the grammar.

  All invalid expressions (i.e. expressions *not* specified by the grammar) should result in a call to `error`.

  The evaluation function must have exactly the interface as described above. Otherwise, you will probably get 0 for this part.

- 2 marks for coding style. This includes consistent and sensible indentation, good variable names, good use of Scheme features, etc.

The maximum score for this assignment is 20 marks. There are no bonus marks.

What to Submit

Put all the code for this assignment into a single file called `proj2.scm`, and submit it on Canvas.

Of course, all the code you submit should be written by you.