**Question 1:**

* When the referred or the ‘called’ variable is an argument of a function, Call-by-reference refers to and is able to modify the original value, whereas Call-by-value can only manipulate the copy of the variable within the function. Call-by-value preserves the original values and prevents unwanted modifications on them but keeps generating copies in the memory at each passing and requires all arguments to be variables. On the contrary, Call-by-reference allows changing the value of the actual variable and does not generate copies by always referring to the same memory address. Thus, Call-by-reference is slightly more advantageous in terms of computational complexity and memory efficiency; however, it all depends on the implementation.
* Call-by-need computes the value of an argument the first time it is needed to be computed and stores it in a cache, whereas Call-by-name computes the arguments each time they are called within a function. Therefore, Call-by-name might end up giving an error in a recursive function if the parameter is not computed and stored in the previous recursion stage. Yet, they are both lazy evaluation methods and both are memory efficient with low computational complexity.

**Question 2:**

We introduced ‘thunk’ data types to act as caches that store the unevaluated variables and to be evaluated (as the thunk) when needed. In order to evaluate variables, **var-exp (var)** in the *interp.scm* should be changed in a way such that after finding the location of the variable (line 2) and dereferencing it to obtain its value (line 3), it should check if the value is an expressed value or a thunk (line 4 & 6) and decides to evaluate the expressed value (line 5) or the thunk (line 7-9).

Also, an additional method for evaluating the value of a thunk should be implemented in the interpreter, which evaluates the saved expression *exp* in the saved environment *env*, which are the stored parameters of the thunk.