Your first assignment will be to program two different versions of insertion sort in Lisp. Here is a pseudocode version of insertion sort, which is written assuming an array/vector of numbers is being sorted. You should use this as a model for your code—i.e., your code should use this algorithm and have this basic structure:

**INSERTION-SORT(A)**

FOR i <- 2 to length[A] DO
    key <- A[i]
    j <- i-1
    WHILE j>0 AND key<A[j] DO
        j <- j-1
    A[j+1] <- key
RETURN A

Lisp versions of Insertion-Sort to be handed in:

1. A version that works with an (one-dimensional) array of numbers as input. (A one-dimensional array is also called a vector in Lisp.) This version should be a single function named `isort`. The numbers may be of arbitrary and possibly mixed types. You should use array-specific accessor functions. Pay attention to the fact that in pseudocode we typically use 1-based array indices, but Lisp uses 0-based indices.

2. A generic function version, which will work with an (one-dimensional) array of objects that are of arbitrary types. This version should be a single function named `isort-generic`. This will require that you introduce a functional argument. You should start with your code to version (1) above, and make the necessary changes. The first thing to do is add a second parameter: the comparison predicate for the type(s) of objects in the array that is being passed. In the pseudocode version, this comparison is written as `<` because the code assumes numbers. More generally, the comparison predicate tells whether its first argument should precede the second in the sorted sequence. So, introduce the predicate parameter and call it `precedp`. Assume that the comparison predicate will take 2 arguments and return true (T or non-NIL) if the first argument should precede the second argument, or else false (NIL).

As a first step, you can test your code using the same test(s) that you used for version (1) by using `#'<` as the functional argument (this will pass the less-than predicate as the functional argument). You are to also write your own precedes predicate that works for `conses` whose two elements are numbers. Your predicate should be named `cons<`. Cons1 is `cons<` cons2 if (1) its car is `<` cons2’s car; or (2) the cars are equal and cons1’s cdr is `<` cons2’s cdr. Make sure you test your cons comparison predicate.
Submit your definitions in a single file named `lab1.lisp`. The file should contain 3 things:
1. the definition of `isort`;
2. the definition of `isort-generic`;
3. the definition of `cons<`;

Of course you must be certain your code compiles, and you should avoid compiler warnings. Finally, make sure your code adheres to good Lisp style or you will lose points. Good style includes: proper/typical indenting, typical spacing around parens in functions calls, avoiding placing parens on lines alone, avoiding explicit return statements, using built-in functions when available, using standard Lisp approaches, etc.