

Assignment 6

Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor).

Define the following Common Lisp functions and macros. Please save your common lisp code in a file named Assignment6.lisp and upload the file to Gradescope when the assignment is complete. A few notes about the assignment:

- Assignment6.lisp is expected to contain only the methods described in this assignment manual. (load “Assignment6.lisp”) should return T and print no other information to the console.
- Parameter names can be arbitrary. I gave the parameters names to identify them in the assignment manual but your implementation does not need to use those specific names.
 - Function and macro names, on the other hand, must be exactly as specified.
- Parameters **must** be in the order they are listed in the assignment manual.
- You may use as many optional parameters as you like, even if no optional parameters are described in the assignment manual.
- Functions are expected to return values, not print values to the console.
 - Functions should not print **any** information to the console when executed.
- Do not use defun inside of a function body.
- Do not define any global variables with defvar. For local variables inside functions, use let and let* blocks.
- Assume all inputs are valid, you do not need to do any parameter validation in your functions.
- You are allowed (and encouraged) to utilize methods defined in Common Lisp which have not been discussed in class. The documentation for Common Lisp can be found at <http://clhs.lisp.se/> (I advise using Google to search the site. Although the information is good, the web design aspect leaves much to be desired)
- You **must** comment your Common Lisp code. The ; character is used to denote comments.
- You **cannot** use any destructive operations in any of your functions. This includes: the setq special operator, setf, delete, nconc, push, or any derivatives of the setq special operator. Destructive operations go against the functional paradigm described in class.
 - A non-destructive alternative to nconc is the append function
- I highly recommend against using looping macros and online compilers for the development of this assignment, historically they have caused significant confusion.
- SBCL can be installed onto almost any modern operating system. You can download and install the SBCL compiler [here](#)
- I do not have a preference on what environment you develop your Common Lisp code on. But Gradescope will be testing it on SBCL version 1.4.12.

1. [5 points] Write a Common Lisp function named `myList` which creates the following list and returns it

```
(4 (7 22) "art" ("math" (8) 99) 100)
```

2. [10 points] Write a Common Lisp function named `leapYear` which takes no required parameters and returns an ordered list containing all leap years from 1800 through 2021. The list of leapyears **must** be calculated, no points will be given for a hard-coded list.
3. [10 points] Write a Common Lisp function named `union-` which takes two list parameters. `union-` returns a single list which contains the separate unique entities from both lists, with no duplication. You are not allowed to use the predefined `union` function for this function.
4. [10 points] Write a Common Lisp function named `avg` with a single parameter named `aList`. `avg` returns the average of all elements in `aList`. Assume all elements in `aList` are numbers. If given an empty list, `avg` should return `NIL`. The `avg` function **must** be tail recursive.
5. [15 points] Write a Common Lisp function named `isType` which takes a single parameter named `dataType`. `isType` will return an anonymous function which takes a single parameter and returns true if the parameter's data type is the data type specified in `dataType`. Otherwise the anonymous function returns false.
6. [15 points] Write a Common Lisp function named `taxCalculator` with three parameters: `limit`, `rate`, and `values`. `limit` and `rate` will be numbers, `values` will be a list. `taxCalculator` returns a list with the same elements and ordering of the values parameter EXCEPT every element which is greater than `limit` is multiplied by `rate`. Assume that all elements of the `values` list are numbers.
BONUS: Make `taxCalculator` tail recursive +5 points

7. [20 points] Write a Common Lisp function named `clean` which takes two parameters: `aFunc` and `aList`. `aFunc` will be a function and `aList` a list. `aFunc` is expected to be a function which takes a single parameter and returns a boolean value. `clean` will return a list which contains all values in `aList` which, when passed to `aFunc`, return true. If `aList` contains sublists, `clean` should create a new sublist with only the values which return true when passed to `aFunc`.
8. [15 points] Define a Common Lisp macro named `threeWayBranch`, which takes three parameters: `x`, `y`, and `z`. All parameters will be lists containing an arbitrary number of sublists where the first sublist is a conditional.

```
( (conditional) (sublist-1) (sublist-2) ... )
```

The `threeWayBranch` macro will behave similarly to the `cond` macro. It will first evaluate the conditional in parameter `x` and if it evaluates to true, execute all subsequent sublists in parameter `x` then return. If the conditional in `x` is `NIL`, the conditional in `y` will be evaluated. If true, the subsequent sublists in parameter `y` will be executed otherwise the conditional in `z` will be evaluated. If the conditional in `z` evaluates to true, execute the remaining sublists in `z`. If no conditionals evaluate to true, `threeWayBranch` will return `NIL`. You are not allowed to use the `cond` macro to implement `threeWayBranch`.