1. (10 pts) Write a Lisp function called `my-rotate` that takes a list, pops off the first element and adds it to the end of the list, returning the resulting list. For example:

   `(my-rotate '(a b c))`

   returns `(b c a)`

2. (10 pts) Write a Lisp function called `my-rotate-n` that takes a number n and a list and performs the “my-rotate” function n times. For example:

   `(my-rotate-n 3 '(a b c d))`

   returns `(d a b c)`.

3. (15 pts) Write a Lisp function `first-sat` that takes two lists and a function `foo` as arguments. Function `foo` should take two arguments and return Boolean (t or nil). The result of a call to `first-sat` should be a list containing the first pair of arguments that satisfies (returns true) from `foo`. For example:

   `(first-sat '(1 4 3 5) '(2 5 1 4) #'(lambda (x y) (> x y)))`

   Note that `#'` is the same as `(function …)` so

   `(first-sat '(1 4 3 5) '(2 5 1 4) (function (lambda (x y) (> x y))))`

   returns `(3 1)`.

4. (25 pts) Write a Lisp function `my-remove` that takes an atom and a list as input and returns a list with all instances of the atom removed (including recursive instances). Note that you may not use the Lisp function `remove` or any related function! Implement this using only one function (no helper functions). For example,

   `(my-remove 'b '(a b c d))`

   returns `(a c d)`

   `(my-remove 'b '(((a b) b (c b d e a) (b) (a) c))`

   returns `((a) (c d e a) nil (a) c)`

5. (25 pts) Write a Lisp function called `palindromep` that takes a list as input and returns t if the list is a palindrome and nil otherwise. Do NOT use the built-in reverse function or any related function. For example:

   `(palindromep '(b c c b))`

   returns `t`

   `(palindromep '(c a c))`

   returns `t`

   `(palindromep '(a b c))`
returns nil
(palindromep '(a (b a f) l (b a f) a))
returns t
(palindromep '(a (b a f) l (f a b) a))
returns nil

Note that sublists are not examined (this is shown in the last two examples).