CS 61A Challenge Problems:
Basic Scheme

Solutions at https://alextseng.net/teaching/cs61a/

Alex Tseng

---

1 Constructing Pairs and Lists

For each of the following lines of code, determine what Scheme will print out and draw the corresponding box-and-pointer diagram.

```
(define a '(1 2 3))
(define b '(4 5 6))

(cons a b) ; 1


(cons 10 20) ; 2


(cons '(a b) '(a)) ; 3


(cons (list a b) (list a)) ; 4


(list a b) ; 5


(list (list (list a)) b) ; 6


(append a b) ; 7


(append (cons a 'foo) (list b)) ; 8


(append a b 42) ; 9
```

Make sure you understand the *mechanisms* of `cons`, `list`, and `append`, as they are the 3 main ways of creating pairs and lists. Also make sure you understand `car`, `cdr`, and everything in between. Remember, read the letters right to left: `(caddr a) ---> 3`. There can be up to 4 letters between the `c` and `r`.

2 Functions on Lists

(a) Write a function `last` that takes in a list and returns the last element in the list.
`(last '(1 2 3 4 (5 6) 7)) ---> 7`
`(last '(1 2 3 (4 5)) ---> (4 5))`

(b) Write a function `double` that takes in a list and returns a list with every element duplicated. Assume that every element in the list is a single token, and not another list.
`(double '(1 2 3 4)) ---> (1 1 2 2 3 3 4 4)`

(c) *Challenge* You may be familiar with the function that reverses a shallow list. That is, if the list has elements that are also lists, those inner lists are not reversed themselves. Write a function `deep-reverse` that reverses all elements of the list, including sublists.
`(deep-rev '(1 2 (3 4 5) ((6)) 7 (8 9) 10)) ---> (10 (9 8) 7 ((6)) (5 4 3) 2 1)`

(d) *Challenge* Write a function `flatten` that flattens a list, bringing all elements in sublists to the top level.
`(flatten '(a b (c d) (((e)) f) (g (h (i) j k) l))) ---> (a b c d e f g h i j k l)`

3 Iteration to Recursion in Scheme

(a) Write a function `prime` that tests if a number is prime. You may find the function `mod` useful, which is the equivalent to the `%` operator to find the remainder in Python.
Hint: consider writing a helper function

(b) *Challenge* Write a function `fibo` that returns the `n`th Fibonacci number in $\Theta(n)$ time (so no tree recursion).