1 Scheme

Questions

1.1 What will Scheme output? Draw the box and pointer whenever the expression evaluates to some pair or list.

> `(or 'false (/ 1 0) 'true)`

> `(1 2 3)`

> `(cons 2 '())`

> `(cons 1 (cons 2 '()))`

> `(cadar '((1 2) 3 (4 5)))`

> `(caddr '((1 2) 3 (4 5)))`

> `(cddar '((1 2) 3 (4 5)))`

> `(cddr '((1 2) 3 (4 5)))`

1.2 Spot the bug(s). Test out the code and your fixes in the scheme interpreter!

(https://scheme.cs61a.org/)

```scheme
(define (sum-every-other lst)
  (cond ((null? lst) lst)
        (else (+ (cdr lst)
                  (sum-every-other (car lst)))))
)
```

1.3 Define `append`, which takes in two lists and concatenates them together.

> `(append '(1 2 3) '(4 5 6))`

`(1 2 3 4 5 6)`
1.4 Define reverse. You may use append in your definition.

> (reverse '(1 2 3))
(3 2 1)

1.5 Define reverse without using append. (Hint: use a helper function and cons)

1.6 Define add-to-all, which takes in an item and a list of lists, and adds that item to the front of each nested list.

> (add-to-all 'foo '(((1 2) (3 4) (5 6))))
(((foo 1 2) (foo 3 4) (foo 5 6))

1.7 Define map, which takes in a function and a list, and applies that function to each item in the list.

> (map (lambda (x) (+ x 1)) '(1 2 3))
(2 3 4)

1.8 Define add-to-all using one call to map. (Hint: consider using a lambda expression!)

1.9 Define sublists. (Hint: use add-to-all)

> (sublists '(1 2 3))
(() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))

1.10 Define sixty-ones, a function that takes in a list and returns the number of times that 1 follows 6 in the list.

> (sixty-ones '(4 6 1 6 0 1))
1
> (sixty-ones '(1 6 1 4 6 1 6 0 1))
2
> (sixty-ones '(6 1 6 1 4 6 1 6 0 1))
3
1.11 Define **no-elevens**, a function that takes in a number n, and returns a list of all distinct length-n lists of 1s and 6s that do not contain two consecutive 1s.

```scheme
> (no-elevens 2)
((6 6) (6 1) (1 6))
> (no-elevens 3)
((6 6 6) (6 6 1) (6 1 6) (1 6 6) (1 6 1))
> (no-elevens 4)
((6 6 6 6) (6 6 6 1) (6 6 1 6) (6 1 6 6) (6 1 6 1) (1 6 6 6) (1 6 6 1) (1 6 1 6))
```
2 Exceptions

Questions

2.1 How do we raise exceptions in Python?

2.2 How do we handle raised exceptions? And why would we need to do so?