# DATA ABSTRACTION AND SEQUENCES 4

COMPUTER SCIENCE 61AS

## Basics of Pairs and Sequences

1. What is the value of `(car '(1 . 2))`?

2. What is the value of `(cdr '(1))`?

3. Suppose I enter `(cons '(1 . 2) '(3 . 4))` into the interpreter. What will Racket print?

## Practice with Pairs and Sequences

1. What will Racket print if each of these are entered into the interpreter? If it is an error, write 'error'. Draw the box and pointer diagram for each one if possible.

   a. `(cons (cons (cons 1 4) 5) (cons 3 (list 4)))`

b. `(append (cons 1 (list 3)) (list (cons 1 2)))`

c. `(list (append (list 2 4) (list (list (list 6))) '(3)) '(1))`

## Basics of Data Abstraction

1. Why do we define constructors and selectors rather than just telling people use `car` and `cdr`?

2. I want to create an ADT representing a point in 2D space that keeps track of its x and y coordinates. Define a set of possible constructor/selectors called `make-point`, `point-x`, and `point-y`.

## Practice with Data Abstraction

1. Suppose we want to write a procedure which, given a list of grades (numbers between 0 and 100), finds both the minimum and the maximum grade. Since in Racket we can only return one thing, well invent a new ADT  a num-pair. Assuming you have already defined the constructor, `make-num-pair`, and the selectors, `first-num` and `second-num`, write a procedure `minmax` which takes in a list of numbers and returns a `num-pair` whose `first-num` is the minimum element and whose `second-num` is the maximum element. You may assume that the input list has at least one number.

2. Now lets actually write the constructors and selectors. But lets test 3 different types!

   a. Write constructors and selectors which represent a num-pair as a pair/list.

```
> (pair? (make-num-pair 50 60))
#t
> (second-num (minmax (89 94 83 95 91 50)))
95
```

   b. Write constructors and selectors which represent a num-pair as a number. (Hint: Remember that since the numbers represent grades, they must be in the range 0-100, inclusive.)

```
> (number? (make-num-pair 50 60))
#t
> (first-num (minmax (89 94 83 95 91 50)))
50
```

   c. Write constructors and selectors which represent a num-pair as a procedure.

```
> (procedure? (make-num-pair 50 60))
#t
> (second-num (minmax (89 94 83 95 91 50)))
95
```