Sample Questions: SICP Section 2.1

Data Abstraction

1. Suppose the rational constructor *make-rat* could be passed both integers *and* rational numbers. How could the constructor identify the type of its arguments in order to simply its result? Choose the most complete answer.

   (A) use *integer?* or *pair?*

   (B) use *integer?*

   (C) use *pair?*

   (D) neither

2. Suppose the rational constructor *make-rat* could be passed both integers *and* rational numbers. How many cases must the constructor handle in order to simply its result?

   (A) 5

   (B) 1

   (C) 4

   (D) 8

   (E) 3

   (F) 6

   (G) 2

   (H) 7

3. Suppose the rational constructor *make-rat* could be passed both integers *and* rational numbers. Define an expression that the constuctor could use to simpify the resultion numerator in the *rational-rational* case. Use the variables *nn* and *dn* to refer to the numerator of the numerator and the denominator of the numerator, respectively. Likewise for *nd* and *dd*.

4. With this implementation for *cons*:

   ```
   (define (cons a b) (lambda (w) (w a b)))
   ```

   which of the following would be a valid implementation of *car*?

   (A) `(define (car c) (lambda (x y) x))`

   (B) `(define (car c) (lambda (c) (c (lambda (x y) x))))`

   (C) `(define (car c) (c (lambda (x y) x)))`

   (D) `(define (car c) (c c))`

5. With this implementation for *cons*, which only works for integers:

   ```
   (define (cons a b) (* (^ 2 a) (^ 3 b)))
   ```

   which of the following would is a valid implementation of *car*? Assume `^` is the exponentiation function and that when given two integers, returns an integer. Assume `(ilog b d)` returns the closest integer to $log_b d$.

   (A) `(define (car x) (ilog x 2))`

   (B) `(define (car x) (if (= (% x 2) 0) 0 (+ 1 (car (/ x 2)))))`

   (C) `(define (car x) (if (= (% x 2) 0) (ilog 3 x) (car (/ x 2))))`

   (D) `(define (car x) (if (!= (% x 3) 0) (ilog 2 x) (car (/ x 3))))`

6. Given the following incrementer and base:

```
(define (inc x) 1)
(define base (lambda (y) (+ 1 y)))
```

and the evaluation of the Church numbers *four*:

```
(define x ((four inc) base))
```

What is the value of x?

(A) 5

(B) 4

(C) 1

(D) `(lambda (y) (+ 1 y))`

7. Which are valid functions for adding two Church numerals?

```
(define (add a b)
    (lambda (f)
       (lambda (x)
           ((((a add-1) b) f) x)
           )
        )
    )

(define (add a b)
    (lambda (f)
        (lambda (x)
            ((a f) ((b f) x))
            )
        )
    )
```

(A) the second

(B) the first

(C) both

(D) neither