Hannah Zhang

2.1, 2.2, 2.3, 2.4, 2.7, 2.8, 2.9

```
; prompt
; comments
; answers
; question
```

2.1

Define a better version of make-rat that handles both positive and negative arguments.

- Rational number is pos, both the num and denom are positive
- Rational number is negative, only the numerator is negative
- Note: numer, denom, print-rat are previously defined

```
; first version given
(define (make-rat n d) (cons n d))
; second version given
(define (make-rat n d)
  (let ((g (gcd n d)))
    (cons (/ n g) (/ d g))))
; the better make-rat based on the first version, with rules
from the question
(define (make-rat n d)
  (cond ((< n 0) (cons n d))
        ((< d 0) (cons (- n) (- d)))
        (else (cons n d))))

; the better make-rat based on the second version with gcd and
with rules from the first question
(define (make-rat n d)
  (let ((g (gcd n d)))
    (cond ((< n 0) (cons (/ n g) (/ d g)))
          ((< d 0) (cons (/ (- n) g) (/ (- d) g)))
          (else (cons (/ n g) (/ d g))))))

; add-rat, sub-rat, mult-rat, div-rat are procedures that when
called, return a list.
; that is why we must use print to make it readable
```

2.2

Define a constructor make-segment and selectors start-segment and end-segment that define the representation of segments in terms of points.

```
; make-segment takes two points and converts it to a list
(define (make-segment x y)
  (cons x y))
; start-segment takes a segment and finds the first point
(define (start-segment segment)
  (car segment))
; end-segment takes a segment and finds the last point
(define (end-segment segment)
  (cdr segment))
; make-point takes two coordinates and converts it to a list
(define (make-point x-coord y-coord)
  (cons x-coord y-coord))
; x-point takes a point and finds the x-coord
(define (x-point point)
  (car point))
; y-point takes a point and finds the y-coord
(define (y-point point)
  (cdr point))
; midpoint takes a segment and finds the middle point
(define (midpoint segment)
  (cons (/ (+ (x-point (start-segment segment)) (x-point
(end-segment segment))) 2 )
        (/ (+ (y-point (start-segment segment)) (y-point
(end-segment segment))) 2 )))
```

To test the functions:

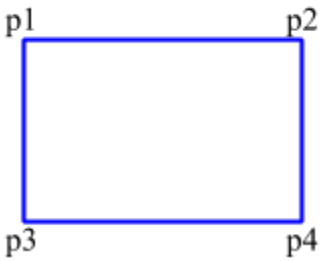
```
(define (print-point p)
  (newline)(display "(")
  (display (x-point p))(display ",")
  (display (y-point p))(display ")"))

(define point1 (make-point 3 4))
(define point2 (make-point 5 4))
(define segment1 (make-segment point1 point2))
```

2.3

Implement a representation for rectangles in a plane.

```
; find the x-coordinate when given a point
(define (x-point point)
  (car point))
; find the y-coordinate when given a point
(define (y-point point)
  (cdr point))
; make a point when given the x-coordinate and y-coordinate
(define (make-point x-coord y-coord)
  (cons x-coord y-coord))
; make a rectangle when given four points
; note where the four points are located below
(define (make-rect p1 p2 p3 p4)
  (list p1 p2 p3 p4))
; helper
(define (square x)
  (* x x))
; finds the length when given p1 and p2 in a rectangle
(define (length rectangle)
  (sqrt (+ (square (- (x-point (car rectangle)) (x-point (cadr
rectangle))))
           (square (- (y-point (car rectangle)) (y-point (cadr
rectangle)))))))
; finds the width when given p1 and p3 in a rectangle
(define (width rectangle)
  (sqrt (+ (square (- (x-point (car rectangle)) (x-point (car
(cdr (cdr rectangle))))))
           (square (- (y-point (car rectangle)) (y-point (car
(cdr (cdr rectangle)))))))))
```



```
; find area when given a rect by multiplying width and length
(define (area rectangle)
  (* (length rectangle) (width rectangle)))
; find area when given a rect in a similar fashion
(define (perimeter rectangle)
  (+ (* 2 (length rectangle)) (* 2 (width rectangle))))
```

To test the functions

```
(define p2 (make-point 0 3))
(define p4 (make-point 5 3))
(define p1 (make-point 0 0))
(define p3 (make-point 5 0))

(define rect1 (make-rect p1 p2 p3 p4))
```

Now implement a different representation for rectangles.

```
; in the previous example, the representation was p1, p2, p3, p4
; now we only need to take two points
```


```
(define (x-point point)
  (car point))

(define (y-point point)
  (cdr point))

(define (make-point x-coord y-coord)
  (cons x-coord y-coord))

; this time, it takes only two points
; for this rectangle, the sides must be aligned with the x and y
axis
(define (make-rect p1 p2)
```

```
  (cons p1 p2))

(define (square x)
  (* x x))
; find length given a rectangle with two points (lined with
axis)
(define (length rectangle)
  (abs (- (x-point (car rectangle)) (x-point (cdr rectangle)))))
; find width given a rectangle with two points (lined with axis)
(define (width rectangle)
  (abs (- (y-point (car rectangle)) (y-point (cdr rectangle)))))
; find area given the rectangle with two points
(define (area rectangle)
  (* (length rectangle) (width rectangle)))
; find perimeter given the rectangle with two points
(define (perimeter rectangle)
  (+ (* 2 (length rectangle)) (* 2 (width rectangle))))

To test functions:
(define p1 (make-point 0 3))
(define p2 (make-point 5 0))

(define rect1 (make-rect p1 p2))
```

2.4

Verify that (car (cons x y)) yields x for any objects x and y

```
(define (cons x y)
  (lambda (m) (m x y)))

(define (car z)
  (z (lambda (p q) p)))
```

Why? → Substitution model

```
> (car (cons 3 4))
> > (cons 3 4)
    (lambda (m) (m 3 4))
    #<procedure> (because no values are given for lambda)
> (car (lambda (m) (m 3 4)))
> > ((lambda (m) (m 3 4)) (lambda (p q) p))
    ; lambda now has an input, it is (lambda (p q) p))
    ; once lambda has an input, it goes from substitution to
evaluation
> ((lambda (p q) p) 3 4)
; p is 3, q is 4
3
```

What is the corresponding definition of cdr?

```
; is the exact same except in the final step, it takes q which
is the last element in the given list
(define (cdr z)
  (z (lambda (p q) q)))

> ((lambda (p q) q) 3 4)
; p is 3, q is 4
4
```

2.7

Define selectors upper-bound and lower-bound to complete the implementation.

```
(define (make-interval a b) (cons a b))
; add-int, mul-int, div-int are given already

; given an interval, determines which one is larger
(define (upper-bound interval)
  (if (> (car interval) (cdr interval))
      (car interval)
      (cdr interval)))
; given an interval, determines which one is smaller
(define (lower-bound interval)
  (if (> (car interval) (cdr interval))
      (cdr interval)
      (car interval)))
```

To test functions:

```
(define int1 (make-interval 3 4))
(define int2 (make-interval 1 5))
```

2.8

Using reasoning analogous to Alyssa's, describe how the difference of two intervals may be computed. Define a corresponding subtraction procedure, called sub-interval.

```
; using add-interval as a template to write sub-interval
; the minimum is the smallest of one interval minus the largest
of the other → x value
; the maximum is the largest of one interval minus the smallest
of the other → y value

(define (sub-interval x y)
  (make-interval (- (lower-bound x) (upper-bound y))
                 (- (upper-bound x) (lower-bound y))))
```

2.9

The *width* of an interval is half of the difference between its upper and lower bounds.

- In other words, the width of one interval is (upper-bound minus lower-bound) divided by two

For some arithmetic operations the width of the result of combining two intervals is a function only of the widths of the argument intervals

- In other words f(a.width, b.width)

whereas for others the width of the combination is not a function of the widths of the argument intervals

- In other words it is NOT f(a.width, b.width)

Show that the width of the sum of two intervals is a function only of the widths of the intervals being added

- Show that width of sum = f(a.width, b.width)
- Find f
- (upper-bound.a + upper-bound.b) - (lower-bound.a + lower-bound.b) = (2 * width)
- (upper-bound.a = center.a + width.a)
  (lower-bound.a = center.a - width.a) → same for b
- ((center.a + width.a) + (center.b + width.b)) - ((center.a - width.a) + (center.b - width.b)) = (2 * width)
- (2width.a + 2width.b) = (2width)
- **width.a + width.b = width** (this is the function)
- This means that the final width is equal to the sums of the widths of the intervals

Now, show that difference is the function of the widths of the intervale being subtracted

- Show that width of difference = f(a.width, b.width)
- Same thing with subtraction except you minus the (lower-bound.a with upper-bound.b) and (upper-bound.a with lower-bound.b)
- In the end, the function is also the same as above

Give examples to show that this is not true for multiplication or division.

- We assume lower-bound a and lower-bound b is the smallest
- We assume upper-bound a and upper-bound b is the largest
- (lower-bound.a * lower-bound.b) - (upper-bound.a * upper-bound.b) all divided by two = width

- (upper-bound.a = center.a + width.a)
  (lower-bound.a = center.a - width.a) → same for b
- ((center.a - width.a) * (center.b - width.b)) - ((center.a + width.a) * (center.b + width.b)) / 2 = width
- Center can not be canceled out
- Therefore we do not get width.a + width.b = width
- Therefore you can not find a function with only width.a and width.b as inputs to be added