Hannah Zhang

1.1 - 1.6

1.1

```
10 - 10
12 - (+ 5 3 4)
8 -- (- 9 1)
3 -- (/ 6 2)
6 -- (+ (* 2 4) (- 4 6))
     (define a 3)
     (define b (+ a 1))
19 - (+ a b (* a b))
#f - (= a b)
4 -- (if (and (> b a) (< b (* a b)))
       b
        a)
16 - (cond ((= a 4) 6)
        ((= b 4) (+ 6 7 a))
        (else 25))
6 -- (+ 2 (if (> b a) b a))
16 - (* (cond ((> a b) a)
          ((< a b) b)
          (else -1))
      (+ a 1))
```

1.2

$$\frac{5+4+\left(2-\left(3-\left(6+\frac{1}{3}\right)\right)\right)}{3(6-2)(2-7)}$$

↓

```
(/ (+ 5 4 (- 2 (- 3 (+ 6 (/ 4 3))))) (* 3 (- 6 2) (- 2 7)))
```

1.3

```
; Takes three numbers and determines the sum of squares of the
two maximum

(define (sumsquare a b c)
  (cond ((and (< a b) (< a c)) (+ (* b b) (* c c)))
        ((and (< b c) (< b a)) (+ (* c c) (* a a)))
```

```
        ((and (< c a) (< c b)) (+ (* a a) (* b b)))
        (else '(the numbers are the same))))
```

OR

```
(define (sum a b c)
  (cond ((and (> a b) (> c b)) (+ (* a a) (* c c)))
     ((and (> b a) (> c a)) (+ (* b b) (* c c)))
     ((and (> a c) (> b c)) (+ (* a a) (* b b)))
     ((= a b) (+ (* a a) (* c c)))
     ((= a c) (+ (* a a) (* b b)))
     ((= b c) (+ (* c c) (* b b)))
     (else '(they are the same numbers))))
```

1.4

- This procedure checks to see it b is greater than 0.
- If so, it adds a to +b which is a + b.
- If not, it adds a to -b which is a + b.
- Therefore, it is equivalent to getting the absolute value
    - Since b is already negative, adding another negative to it will make it positive.

1.5

- Applicative-order: Evaluate all arguments fully, pass the results into a procedure's formal parameters, then apply the procedure.
    - Evaluate the parameters/functions and then apply
- Normal-order: Pass the arguments, as expressions, into a procedure's formal parameters, then apply the procedure, evaluating the parameters only when needed.
    - Fully expand and then reduce (apply functions)
- In Ben’s case, if the interpreter returns 0 then it is normal order
    - Evaluates 0 and p separately. Evaluates 0 first → 0
- If it returns nothing then it is applicative order because it will run in an infinite loop
    - Evaluates both 0 and p and p is a loop

1.6

- When Alyssa attempts to use this new-if to compute square roots it will run an infinite loop
- This is because the new-if uses applicative order rather than normal order which makes it evaluate all the expressions before applying the actual cond statement.
- Since one of the expressions is sqrt-iter, it will run this function in an infinite loop
    - (sqrt-iter 2 4)
    - Returns true, supposed to go to new-if
    - Because it is applicative, evaluates sqrt-iter first
    - Stuck in a loop with (sqrt-iter 2 4)