## Exercise1.1

10
12
8
3
6
#<unspecified>
#<unspecified>
19
false
4
16
6
16

## Exercise1.2

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

```
-37/150
```

## Exercise1.3

## Exercise1.4

```
(define (a-plus-abs-b a b)
((if (> b 0) + -) a b))
```

if b > 0, the operator will be +, and the result is (a + b);
else, the operator will be -, and the result is (a - b);

## Exercise1.5

```
(define (p) (p))
```

```
(define (test x y)
(if (= x 0) 0 y))

(test 0 (p))
```

Applicative-order: no output, never stop, cost more than 12% cpu
normal-order: 0
scheme use applicative-order

## Exercise1.6

```
(define (new-if predicate then-clause else-clause)
(cond (predicate then-clause)
(else else-clause)))
;; define a procedure
(define (abs x)
  ;;     cond     action
  (cond ((< x 0) (- x))
        ((= x 0) 0)
        ((> x 0) x)))

(define average (lambda (x y) (/ (+ x y) 2)))

(define (improve guess x)
  (average guess (/ x guess)))

(define (square x) (* x x))
(define (good-enough? guess x)
  (< (abs (- (square guess) x)) 0.001))

(define (sqrt-iter guess x)
(cond ((good-enough? guess x) guess)
(else (sqrt-iter (improve guess x) x))))
;; (new-if (good-enough? guess x)
;; guess
;; (sqrt-iter (improve guess x) x)))

(sqrt-iter 2 5)
```

```
161/72
```

I try to run the code, and find it never stop
when enter new-if, predicate, then-clause and else-clause need to be computed, (sqrt-iter (improve guess x) x) will be excuted whether good enough or not, so it never stop

## Exercise1.7

```
(define average (lambda (x y) (/ (+ x y) 2)))

(define (improve guess x)
  (average guess (/ x guess)))

(define (square x) (* x x))
(define (good-enough? guess x)
  (< (abs (- guess x)) 0.001))

(define (sqrt-iter guess x)
(cond ((good-enough? guess (improve guess x)) guess)
(else (sqrt-iter (improve guess x) x))))

(sqrt-iter 2 0.001)
```

```
0.031642054886792094
```

## Exercise1.8

if $x = y^3$ (the fix point) then $\frac{x/y^2+2y}{3} = y$

```
(define (square x) (* x x))

(define (improve guess x)
  (/ (+ (/ x
           (square guess))
        (* 2 guess))
     3))

(define (good-enough? guess x)
  (< (abs (- guess x)) 0.001))

(define (cube-iter guess x)
(cond ((good-enough? guess (improve guess x)) guess)
(else (cube-iter (improve guess x) x))))

(cube-iter 2 0.001)
```

```
0.10044361382333848
```