Chapter 1: Building Abstractions with Procedures

This chapter is an introduction to functional programming, and more concretely to lisp programming.

Excercise 1.1

- `10`
- `(+ 5 3 4)` ⟶ 12
- `(- 9 1)` ⟶ 8
- `(/ 6 2)` ⟶ 3
- `(+ (* 2 4) (- 4 6))` ⟶ 6
- `(define a 3)` ⟶ Stores 3 into var *a*
- `(define b (+ a 1))` ⟶ Stores 4 `(+ 3 1)` into var *b*
- `(+ a b (* a b))` ⟶ 19
- `(= a b)` ⟶ NIL

-
  ```
  (if (and (> b a) (< b (* a b)))
      b
      a)
  ```

  ↳ 4

-
  ```
  (cond ((= a 4) 6)
        ((= b 4) (+ 6 7 a))
        (else 25)))
  ```

  ↳ 16
- `(+ 2 (if (> b a) b a))` ⟶ 6

-
  ```
  (* (cond ((> a b) a)
           ((< a b) b)
           (else -1))
     (+ a 1))
  ```

  ↳ 16

Excercise 1.2

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

Excercise 1.3

```
(define ex1.3 (x y z)
      (cond ((> x y)
              (if (> y z)
                  (+ (* x x) (* y y))
                  (+ (* x x) (* z z))))
            (t
              (if (> x z)
                  (+ (* y y) (* x x))
                  (+ (* y y) (* z z))))))
```

Excercise 1.4

The function `a-plus-abs-b` utilizes the if condition to change the operation to a sum if b is positive or a substraction otherwise, acting as $|b|$.

Mathematically:

$$\text{a-plus-abs-b}(a, b) = \begin{cases} a+b \text{ if } b>0 \\ a-b \text{ if } b<0 \end{cases} \equiv a + |b|$$

Excercise 1.5

With an applicative oreder evaluation, the test function will not run properly because `(p)` will loop on itself, continiously running `(test 0 (p))`. Using normal order evaluation, because y is not utilized on the `test` function, the `if` clause will be executed and resolve to 0.

Excercise 1.6