Solutions to SICP

$$\lambda$$

Bernardo Meurer

November 2017

1 Building Abstractions with Procedures

Exercise 1.1. Below is a sequence of expressions. What is the result printed by the interpreter in response to each expression? Assume that the sequence is to be evaluated in the order in which it is presented.

a. `10`

b. `(+ 5 3 4)`

c. `(- 9 1)`

d. `(/ 6 2)`

e. `(+ (* 2 4) (- 4 6))`

f. `(define a 3)`

g. `(define b (+ a 1))`

h. `(+ a b (* a b))`

i. `(= a b)`

j.
```
(if (and (> b a) (< b (* a b)))
    b
    a)
```

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

l. `(+ 2 (if (> b a) b a))`

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

Solution 1.1. a. `10`

b. `12`

c. `8`

d. `3`

e. `6`

f.

g.

h. `19`

i. `#f`

j. `4`

k. `16`

l. `6`

m. `16`

Exercise 1.2. Translate the following expression into prefix form

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

Solution 1.2. `(/ (+ 5 4 (- 2 (- 3 (+ 6 (/ 4 3))))) (* 3 (- 6 2) (- 2 7)))`

Exercise 1.3. Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

Solution 1.3. Using composite procedures:

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

(define (sum-of-squares x y)
    (+ (square x) (square y)))

(define (>= x y)
    (not (< x y)))

(define (foo x y z)
    (cond
        ((and (>= x y) (>= y z)) (sum-of-squares x y))
        ((and (>= y x) (>= x z)) (sum-of-squares x y))
        ((and (>= x y) (>= z y)) (sum-of-squares x z))
        (else (sum-of-squares y z))
    )
)
```

Exercise 1.4. Observe that our model of evaluation allows for combinations whose operators are compound expressions. Use this observation to describe the behavior of the following procedure:

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

Solution 1.4. The `if` expression checks whether `b` is positive or not, and sets the operator for the outer procedure accordingly. So if `(> b 0)` evaluates to false, then we set the operator to `-`, effectively summing the absolute value of `b`.

Exercise 1.5. Ben Bitdiddle has invented a test to determine whether the interpreter he is faced with is using applicative-order evaluation or normal-order evaluation. He defines the following two procedures:

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

(define (test x y)

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

Then he evaluates the expression

```
(test 0 (p))
```

What behavior will Ben observe with an interpreter that uses applicative-order evaluation? What behavior will he observe with an interpreter that uses normal-order evaluation? Explain your answer. (Assume that the evaluation rule for the special form if is the same whether the interpreter is using normal or applicative order: The predicate expression is evaluated first, and the result determines whether to evaluate the consequent or the alternative expression.)

Solution 1.5.