MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.037—Structure and Interpretation of Computer Programs
IAP 2019

Scheme Basics

********** SOLUTIONS *********

Getting Started

For each Scheme expression below, what value results when the expression is evaluated?

```
42  => 42

"Hello World" => "Hello World"

(8 + 9)  => Error, expected procedure, given 8

(+ 8 9)  => 17

(define a 10) => undefined

a => 10

b => error

(define b a) => undefined

b => 10

(* a b) => 100
```

Nested Expressions

For each Scheme expression below, what value results when the expression is evaluated?

```
(* (- 8 4) (+ 1 10))  => 44

(define foo 100) => undefined

(define bar (* 10 foo)) => undefined

(+ (- (- 2010 (/ bar foo)) (* foo (- (/ bar foo) 3))) 37) => 1337
```

Hello, λ

For each Scheme expression below, what value results when the expression is evaluated?

```
(lambda (x) (/ x 1024)) => procedure

((lambda (x) (/ x 1024)) 4096) => 4

(lambda () 1) => procedure

((lambda () 1)) => 1

(((lambda () 1))) => Error, expected procedure, given 1

((lambda () 1) 5) => Error, too many arguments (expected: 0, got: 1)

(lambda (y z) (+ z y)) => procedure

((lambda (y z) (+ z y)) 5 4) => 9

((lambda (y z) (+ z y)) x 7) => error, x undefined
```

What's in a name?

Assume that you've already evaluated the following Scheme expressions:

```
(define x 1)
(define y -1)
(define foo (lambda (a b) (+ a b)))
(define bar (lambda (x) x))
(define baz (lambda () 1))
(define quux (lambda (p) (foo p 5)))
```

Alright, now to what value do each of these Scheme expressions evaluate?

```
x  => 1

foo => procedure

(foo 1 2) => 3

(foo 1) => error, too few arguments (expected: 2, got: 1)

(foo)   => error, too few arguments (expected: 2, got: 0)
```

```
(baz)   => 1

(bar 10) => 10

(quux (foo (baz) (bar y))) => 5
```

Short and sweet: Syntactic Sugar

For each Scheme expression below, write an equivalent Scheme expression that doesn't explicitly use `lambda`.

```
(define foo (lambda (a b) (+ a b)))  => (define (foo a b)
                                          (+ a b))

(define bar (lambda () 1)) => (define (bar)
                                 1)
```

Sum of all its parts

Write a procedure named `sum-numbers` which takes as input two integers, M and N, and returns the sum of all the numbers on the interval [M,N].

```
(define sum-numbers-rec
  (lambda (m n)
    (if (> m n)
      0
      (+ m (sum-numbers-rec (+ m 1) n)))))


(define sum-numbers-iter
  (lambda (m n)
    (sum-helper m n 0))

(define sum-helper
  (lambda (m n sum)
    (if (> m n)
      sum
      (sum-helper (+ m 1) n (+ m sum)))))
```

Fibonacci

```
(define (fib n)
  (if (= n 0)
```

```
      0
      (if (= n 1)
          1
          (+ (fib (- n 1)) (fib (- n 2))))))
```

For the iterative one, definitely show the table as per the lecture before writing code:

F_{x-2}	F_{x-1}	n
0	1	7
1	1	6
1	2	5
2	3	4
3	5	3
5	8	2
8	13	1

```
(define (fib-iter n)
  (if (= n 0)
      0
      (fib-helper 0 1 n)))

(define (fib-helper Fx-2 Fx-1 n)
  (if (= n 1)
      Fx-1
      (fib-helper Fx-1 (+ Fx-2 Fx-1) (- n 1))))
```

Feel the power

```
(define (my-expt-rec x y)
  (if (= y 0)
      1
      (* x (my-expt-rec x (- y 1)))))

(define (my-expt-iter x y)
  (define (expt-helper y answer)
    (if (= 0 y)
        answer
        (expt-helper (- y 1) (* x answer))))
  (expt-helper y 1))
```

fast-expt

For this one, first make the error of writing

```
(* (fast-expt x (/ y 2)) (fast-expt x (/ y 2)))
```

and then discuss the difference between that and the version below.

```
(define (fast-expt x y)
  (if (= y 0)
      1
      (if (even? y)
          (square (fast-expt x (/ y 2)))
          (* x (fast-expt x (- y 1))))))
```