Tutorial 3

Some simple examples:

represent complex numbers as lists:

a + bi is represented as (a b)

```
;; addition of two complex numbers represented as a list
;; with two elements
```

Want:

given (a1 b1) and (a2 b2), get (a1+a2 b1+b2)

```
(define cadd
  (lambda (c1 c2)
    (list (+ (car c1) (car c2))
          (+ (cadr c1) (cadr c2)))))
```

;; version using **map**

```
(define cadd2
 (lambda (c1 c2)
   (map + c1 c2)))
```

;; complex number multiplication

recall:

(a1 + b1*i) * (a2 + b2*i) = a1*a2 + b1*a2*i + a1*b2*i + b1*b2*i^2
= (a1*a2 – b1*b2) + (a1*b2 + a2*b1)*i

Want:

given (a1 b1) and (a2 b2), get (a1*a2 – b1*b2 a1*b2 + a2*b1)

```
(define cmult
  (lambda (c1 c2)
    (list
        (– (* (car c1) (car c2)) (* (cadr c1) (cadr c2)))
        (+ (* (car c1) (cadr c2)) (* (cadr c1) (car c2))))))
```

```
;; using apply and map
(define cmult2
  (lambda (c1 c2)
    (list (apply - (map * c1 c2))
          (apply - (map * c1 (reverse c2))))))
```

Car-cdr recursion:

```
; append

(define (append x y)
        (cond ((null? x)  y)
              (else  (cons (car x)
                           (append (cdr x) y)))))


;; Replace a with b in list lst

(define replace
  (lambda (a b lst)
    (if (null? lst)
        ()
        (cons
            (if (eq? a (car lst))
                b
                (car lst))
          (replace a b (cdr lst))))))
```

```
;; using map.
(define replace2
  (lambda (a b lst)
    (map     (lambda (x) (if (eq? a x) b x))     lst)))
```

Fibonacci numbers:

Fib (0) = 1
Fib (1) = 1
Fib (i) = Fib (i-1) + Fib (i-2)

Recall:

; (fib n) returns the nth Fibonacci number
; Pre: n is a non-negative integer

```
(define fib
   (lambda (n)
      (cond ((= n 0) 1)
            ((= n 1) 1)
            (else (+ (fib (- n 1)) (fib (- n 2)))))))
```

Time complexity? exponential in n!

Solution: use an **accumulator** variable

```
; (fast-fib p1 p2 i n) returns the n^th Fibonacci number
; Pre: n>=0 is an integer
;      0<=i<=n is an integer
;      p1 is the (i-1)^st Fibonacci number (or 0 if i=0)
;      p2 is the i^th Fibonacci number

(define fast-fib
      (lambda (p1 p2 i n)
            (if  (= i n)
                 p2
                 (fast-fib  p2  (+ p1 p2)   (+ i 1)   n ))))
```

Note: p2 is an accumulator

```
; (fib n) returns the n^th Fibonacci number
; Pre: n is a non-negative integer

(define fib
      (lambda (n)
          (fast-fin 0 1 0 n)))
```

Time complexity? linear