Login

CPS Refresher

To refresh our memory, let's CPS the factorial procedure. Here is the definition of `fact` in direct style.

```
(define fact
  (lambda (n)
    (cond
      [(zero? n) 1]
      [else (* n (fact (sub1 n)))])))
```

Here is `fact-cps`, which is the CPSed version of `fact`.

```
(define fact-cps
  (lambda (n k)
    (cond
      [(zero? n) (k 1)]
      [else (fact-cps (sub1 n) (lambda (v)
                                 (k (* n v))))])))
```

Since `fact-cps` consumes two arguments instead of one, we might find it convenient to also define a "driver" procedure called `fact`.

```
(define fact
  (lambda (n)
    (fact-cps n (empty-k))))
```

We could use the identity procedure `(lambda (v) v)` as the initial continuation, but we want to signal an error if the initial continuation is invoked more than once. Therefore, we will use the continuation returned from calling the following `empty-k` procedure:

```
(define empty-k
  (lambda ()
    (let ((once-only #f))
      (lambda (v)
        (if once-only
          (error 'empty-k "You can only invoke the empty continuation once")
          (begin (set! once-only #t) v))))))
```

Just for fun, and to demonstrate how `empty-k` behaves, let's write an incorrect definition of `fact-cps` that invokes the continuation `k` multiple times.

```
(define fact-cps
  (lambda (n k)
    (cond
      [(zero? n) (k 1)]
      [else (fact-cps (sub1 n) (lambda (v) (k (* n (k v)))))])))
```

When we test our new version, we get an error:

```
Exception in empty-k: k invoked in non-tail position
Type (debug) to enter the debugger.
```

Here is one more example, using the Fibonacci procedure. Notice this definition of Fibonacci is slightly different than that which we used in class, but it is of no particular relevance. First we define `fib` in direct style.

```
(define fib
  (lambda (n)
    (cond
      [(zero? n) 1]
      [(= n 1) 1]
      [else (+ (fib (sub1 n)) (fib (sub1 (sub1 n))))])))
```

Now we transform `fib` into continuation-passing style.

```
(define fib-cps
  (lambda (n k)
    (cond
      [(zero? n) (k 1)]
      [(= n 1) (k 1)]
      [else (fib-cps (sub1 n) (lambda (v)
                                (fib-cps (sub1 (sub1 n)) (lambda (w)
                                                           (k (+ v w)))))))])))
```

Once again, we can define a driver procedure if we like.

```
(define fib
  (lambda (n)
    (fib-cps n (empty-k))))
```