Lazy Evaluation

Here are some new Scheme expressions:

- (delay exp) returns an object called a *promise,* without evaluating exp.
- (force promise) evaluates the promised expression and returns its value.

If a promised expression has been evaluated once, forcing it again returns its value without re-evaluating it.

For example

```
> (define foo
     (delay
          (begin
             (display "Oh, goody I'm being evaluated!\n")
             2)))
> (force foo)  => Oh goody I'm being evaluated!
                  2
> (force foo) => 2
> (force foo) => 2
```

Another example:

```
> (define d  (+ z 3)) => error: z is undefined
> (define d (delay (+ z 3)))
> (define z 5)
> (force d) => 8
> (define z 23)
> (force d) => 8
```

Now, how could we implement delay and force?

The only place in standard Scheme where we can give an expression without immediately evaluating it is in the body of a lambda expression.

Try this:

```
> (define d (lambda () (+ x 5)))
> (define x 23)
> (d)
```

A lambda expression with no arguments is a wrapper that delays evaluation; such a lambda expression is sometimes called a *thunk*.

To avoid re-evaluating the delayed expression, we can store the expression's value in an internal environment and just return it when we need it --

```
          (delay exp)
is equivalent to
          (let ( [thunk (lambda () exp)]
                 [value 0]
                 [evaluated? #f] )
                   (lambda ( )
                          (if (not evaluated)
                                 (begin
                                        (display "evaluating\n")
                                        (set! value (thunk))
                                        (set! evaluated? #t)
                                        value))
                                 value)))
```

We can then define (force promise) as (promise)

Note that (delay exp) cannot be defined as a procedure, since

```
        (f exp)
```

always evaluates exp.

delay is created as a type of expression through define-syntax.