Chapter 6

(6.5, 3,6, 6.6)

Iterators, Closures, and Recursion

Iterators

A mechanism for iterating over a *collection* of objects

Java and other languages define a standard *interface* for iterator objects

In Ruby, the `each` method is an iterator (and several others)

Ruby Iterators

```
[1,2,3,4,5].each {|i| puts i}

for i in [1,2,3,4,5]
   puts i
end

hash = {:a => 1, :b => 2, :c => 3}

for k, v in hash
   puts "#{k} : #{v}"
end

hash.each {|k,v| puts "#{k} : #{v}"}
```

Ruby Iterators

```
1.upto(5) {|x| print x}

5.times {|x| print x}

0.step(Math::PI/2, 0.1) do |x|
   printf "%.2f\n", Math.sin(x)
end

[1,2,3,4].map {|x| x*x}

(1..20).select {|x| x%2 == 0}

(1..20).reject {|x| x%2 == 0}
```

Iterators and yield

```
def fibs_up_to (n)
  f1, f2 = 1, 1
  while f1 <= n
    yield f1   #invokes the block
    f1, f2 = f2, f1 + f2
  end
end

fibs_up_to(100) {|f| print f, " "}
```

Closures

An object that is both an invocable function and a variable binding for that function

- Code wrapped up as an object

Like a method: Can take parameters and return a value

Like an object: Can make a reference to it and assign to a variable

Lambda – A Ruby Closure

An object that represents a block

```
is_positive = lambda {|x| x >  0}

is_positive.call 6
is_positive[-4]
```

Ruby Closure

```
def f(closure)
    puts
    result = closure.call
    puts "Closure returned: #{result}"
    "Value from f"
end

puts "f returned: " +
                f(lambda { "From lambda" })
```

Ruby Closure

```
def multiplier(n)
   lambda {|data| data.map {|x| x * n} }
end

doubler = multiplier(2)
puts doubler[[1,2,3,4]]

tripler = multiplier(3)
puts tripler[[1,2,3,4]]
```

Like each but returns an array

Recursion

An alternative to iteration

Sometimes more intuitive, sometimes less

Fundamental to functional languages like Scheme

Tail recursion

No computation follows recursive call

```
int gcd (int a, int b) { /* assume a, b > 0 */
 if (a == b) return a;
 else if (a > b) return gcd(a - b, b);
 else return gcd (a, b – a);
}
```

```
start:
  if (a == b) return a;
  else if (a > b) {
     a = a-b; goto start;
  } else {
     b = b – a; goto start;
  }
```