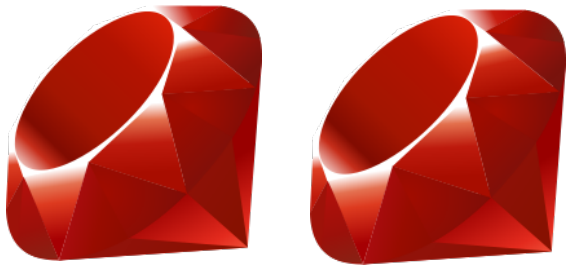

Block Usage

Creating blocks in your methods

- 2 ways to configure a block in your method
  - Explicit
    - Use `&` in front of the last "parameter"
    - Use `call` method to call the block
  - Implicit
    - Use `block_given?` to see if block was passed in
    - Use `yield` to "call" the block

Explicit block example

```
def two_times_explicit (&some_block)
  return "No block" if some_block.nil?
  some_block.call
  some_block.call
end
def two_times_explicit_with_param (some_var, &some_block)
  some_block.call(some_var)
  some_block.call(some_var)
end

puts two_times_explicit # => No block
two_times_explicit { puts "Hello"} # => Hello
                                   # => Hello
two_times_explicit_with_param "Hello" do |param|
  print param + " "
end
# => Hello Hello
```

Implicit block example

```
def two_times_implicit
  return "No block" unless block_given?
  yield
  yield
end
def add_two_values
  return "No block" unless block_given?
  arr = [1, 2]
  ret_val1 = yield arr[0]
  ret_val2 = yield arr[1]
  ret_val1 + ret_val2
end
puts two_times_implicit { print "Hello "} # => Hello
                                          # => Hello
# Preprocess the values before using
puts add_two_values { |val| val * 10} # => 30
```

Blocks as object initializers

- **Problem:** Your object has a lot of attributes.
- It would be nice to initialize them all before the object is used
- Passing everything into constructor is messy
- **Solution:** Use a block to initialize the object

Blocks as initializers - example

```
class Person
  attr_accessor :name, :age, :height
  def initialize
    yield self if block_given?
  end
  def to_s; "#{name}: #{age}, #{height}"; end
end

p1 = Person.new do |p|
  p.name = "Jim"
  p.age = 15
  p.height = "six feet"
end
p2 = Person.new do |p|
  p.name = "Leonardo"
  p.age = 20
  p.height = "seven feet"
end

puts p1 # => Jim: 15, six feet
puts p2 # => Leonardo: 20, seven feet
```

Blocks for transactions

```
class MyFileImpl
  def self.open_and_process(*args)
    file = File.open(*args)
    yield file
    file.close()
  end
end

MyFileImpl.open_and_process("test.txt", "r") do |file_param|
  while line = file_param.gets
    puts line
  end
end

# Line1
# Line2
# etc.
```

lambda - example

```
def makeMultiplierBy5
  a = 5
  lambda { |x| x * a }
end

myMultiplier = makeMultiplierBy5

# … some other code can go here …

puts myMultiplier.call(6) # => 30
```