Ruby Block, Proc and Lambda

Block

你常使用Block

```
ary = [1, 2, 3, 4, 5]
ary.collect! do |n|
    n ** 2
end
p ary
```

做了什麼事？

1. 呼叫了ary的collect!方法，並傳入一個block做參數。
2. collect! 會傳入一個變數，在block中使用，然後乘以平方。
3. ary中的每個數字都被乘以平方了。

自定迴圈Method (I)

```
class Array
  def iterate!
    self.each_with_index do |n, i|
      self[i] = yield(n)
    end
    self
  end
end
```

- 我們利用了each_with_index來取array的element與index，並將element傳給yield
- yield才是執行block的method。

自定迴圈Method (II)

```ruby
ary = [1, 2, 3, 4]
ary.iterate! do |n|
  n ** 2
end

p ary
```

Block

- 不一定需要命名
- 利用yield method執行block。

- 在定義block的時候，利用|n|來傳遞參數。
- yield會回傳block中最後一個statement的回傳值。

把Block視為Proc

```
class Array
  def iterate!(&code)
    self.each_with_index do |n, i|
      self[i] = code.call(n)
    end
    self
  end
end
```

- Block可以被命名，甚至當做參數傳給其他方法。
- 事實上 ，block就是Proc的物件。
- 把block當做參數前面要加&。

Proc

重複使用block？(I)

```
# array_proc.rb
class Array
  def iterate!(code)
    self.each_with_index do |n, i|
      self[i] = code.call(n)
    end
  end
end
```

- 傳遞Proc參數時，不需要&開頭。

重複使用block？(II)

```
require "array_proc"
array_1 = [1, 2, 3, 4]
array_2 = [2, 3, 4, 5]

square = Proc.new do |n|
  n ** 2
end

array_1.iterate!(square)
array_2.iterate!(square)
puts array_1.inspect
puts array_2.inspect
```

傳遞多個block參數

```
def callbacks(procs)
  procs[:starting].call
  puts "Still going"
  procs[:finishing].call
end

callbacks(:starting => Proc.new {
                         puts "Starting" },
          :finishing => Proc.new {
                         puts "Finishing" })
```

跟我們傳遞一般參數相同

lambda

lambda也可以當Proc

```
require 'array_proc'

array = [1, 2, 3, 4]

array.iterate!(lambda { |n| n ** 2 })

puts array.inspect
```

lambda v.s. Proc (I)

- lambda會檢查參數個數，如果錯誤會拋出Exception。

```
def args(code)
  one, two = 1, 2
  code.call(one, two)
end

args(Proc.new{|a, b, c| puts "#{a}, #{b}, #{c}"})

args(lambda{|a, b, c| puts "#{a}, #{b}, #{c}"})
```

lambda v.s. Proc (II)

- 當block裡面有return時，執行方式會不同。Proc會離開他所屬的method，而lambda只是離開自己的block。

- 為什麼？ANS: 設計理念
  - Proc是procedure，像是插了一段code在method中。
  - lambda是method，method呼叫lambda就像是呼叫另一個method。

lambda v.s. Proc (III)

```ruby
def proc_return
  Proc.new { return "Proc.new"}.call
  #這裡就不會被執行了
  return "proc_return method finished"
end

def lambda_return
  lambda { return "lambda" }.call
  #這裡會被執行了
  return "lambda_return method finished"
end

puts proc_return
puts lambda_return
```