

Ruby 簡介

About Me

- 盧韋仁(Lu Wei Jen)
- 目前任職於和多 (Handlino.com)
- Facebook: <http://www.facebook.com/weijenlu>
- Twitter: weijenlu
- 經驗：
 - Rubyconf Taiwan 2010 講者
 - Rubyconf Taiwan 2011 講者
 - JavaTwo 2012 講者

RVM

- Ruby 被很多語言實作：RMI、JRuby、Rubinius、MacRuby、IronRuby ...
- 每一個又有很多version：Ruby 1.8.6、1.8.7、1.9.2、1.9.3、2.0.0
- RVM是一個不錯的管理方式，無論在開發的時候，或是Production Site。
- <https://rvm.io/>

Interactive Ruby(irb)

- irb是Ruby的指令互動環境。
- 執行方式：
 - 在console中輸入： **irb**
 - 你的第一個ruby指令：
 - **puts "Hello World!"**

執行Ruby

- 利用編輯器，建立helloruby.rb
 - `puts "Hello World!"`
- 執行：`ruby helloruby.rb`

程式的說明(I)

- "Hello World" 是一個字串(String)物件。
- 在Ruby的世界中，全部都是物件，包含 nil、true、false。

程式的說明(II)

- 方法 (method)
 - ◆ puts 是一個方法。
 - ◆ 方法是用來傳遞訊息給物件。
 - ◆ 在Ruby中，method之後，並不強制需要接小括號。所以下列兩行敘述 (statement) 是相同的：

```
puts("Hello World Ruby!")  
puts "Hello World Ruby!"
```

字串 (String) |

- 字串可以用單引號(')或雙引號(")來建立：
- 'Hello World' 與 "Hello World" 是相同的。

字符串 (String) II

- 但當字符串中有使用跳脫字元時，就會有不相同之處。
- ex:

```
puts "Hello \nWorld"
```

```
=> Hello  
World
```

```
puts 'Hello \nWorld'
```

```
=> Hello \nWorld
```

字串 (String) III

- 單引號只允許兩個跳脫字元
 - \' - 單引號
 - \\ - 單斜線

字符串 (String) IV

- 雙引號
 - 雙引號允許了比單引號更多的跳脫字元。他也允許你嵌入變數或者Ruby程式碼在一個字符串內。
 - 嵌入Ruby變數：`a = 10; puts "a is #{a}"`
 - 嵌入程式碼：`puts "3 × 5 = #{3 * 5}"`

Output method: print

```
rb(main):004:0> print "Hello World"  
Hello World=> nil
```

- `print`在`stdout`輸出字串後，不會補上一個換行字元。
- `print`回傳(`return`) `nil`。

Output method: puts

```
irb(main):006:0> puts "Hello", "World"
Hello
World
=> nil
```

- puts 在Stdout輸出字串後，一定會換行。
- puts 回傳 nil

Output method: p

```
irb(main):003:0> p "Hello World"  
"Hello World"  
=> "Hello World"
```

- p 在Stdout輸出字串後，會換行。
- p 回傳該物件

數值

- 要用Ruby顯示數值是很簡單的，直接打出數字就好了，Ruby會用相對應的類別去接受。
- 例如你輸入100，就是內容為100的Fixnum物件。
- 如果你輸入3.14，就是內容為3.14的Float物件。

四則運算(I)

- 範例

```
print("1 + 1 = ", 1+1, "\n")
print("2 - 3 = ", 2-3, "\n")
print("5 * 10 = ", 5*10, "\n")
print("100 / 4 = ", 100/4, "\n")
print("25 % 3 = ", 25 % 3, "\n")
```

變數(I)

- 所謂的變數，是將某個物件指定一個人類比較容易了解的名稱。
- 指定變數方式：
 - 變數名 = 物件
 - ex: num = 100
 - 取個容易懂的變數名稱

變數(II)

- 範例

```
x = 10  
y = 20  
z = 30  
area = (x * y + y * z + z * x) * 2  
volume = x * y * z  
puts("Area = #{area}")  
puts("Volume = #{volume}")
```

註解 (I)

- 程式碼中的註解，並不會被當做程式的一部分。也就是說，在程式執行的時候，並不會影響執行結果。
- 註解可以拿來幹嘛？
 - 標明程式的名稱、作者、散佈條件等資訊。
 - 說明程式碼的意義。

註解 (II)

- Ruby 的註解是使用「#」符號表示。
 - `x = 100 # assign 100 to x`
- 如果很長篇的註解可以用「=begin」、「=end」。

條件判斷

條件

- 條件一般是指運算結果為true或false的運算式：
- 在Ruby中，只有運算結果為false 與nil會視為false，其他都視為true。

```
2 > 3    #=> false  
2 < 3    #=> true  
2 == 2   #=> true  
2 >= 3   #=> false  
2 <= 3   #=> true
```

邏輯運算子(I)

- 邏輯運算子`&&`、`||`、`!`。
- 也可以用關鍵字`and`、`or`、`not`。

邏輯運算子(II)

x = 10

x > 1 && x < 10 #=> false

x > 1 and x < 10 #=> false

x > 1 || x < 10 #=> true

x > 1 or x < 10 #=> true

!true #=> false

!false #=> true

!nil #=> true

!"abc" #=> false

!100 #=> false

if

- 我們會希望可以依據某些條件來決定程式要執行的內容。
- if 的語法：

```
if 條件  
    要執行的動作  
end
```

else (I)

- 當條件成立與不成立的時候，要執行不同內容的話，可以使用else。
- 語法：

if 條件

 條件成立時要執行的動作

else

 條件不成立時要執行的動作

end

else (II)

- (範例) basic/bigger_smaller.rb

```
a = 20
if a >= 10
  puts "bigger"
else
  puts "smaller"
end
```

if + elseif + else

- 語法：

```
if condition1  
    statement1  
elseif condition2  
    statement2  
else  
    statement3  
end
```

unless 敘述 (I)

- unless敘述是與if完全相反的敘述。
- 語法：

```
unless condition  
    statement1  
else  
    statement2  
end
```

unless 敘述 (II)

- (範例) basic/bigger_smaller3.rb

```
a = 13
unless a < 10
    puts "bigger"
else
    puts "smaller"
end
```

case 敘述(I)

- 當條件很多的時候，用case敘述會更簡單好懂。
- 語法：
case [要被比較的物件]
when value1
 statement1
when value2
 statement2
else
 statement3
end

case 敘述(I)

- 範例

```
x = 83
case x
when 90..100
  puts "A"
when 80..90
  puts "B"
when 70..80
  puts "C"
else
  puts "D"
end
```

單行的if 與 unless

```
if a > b  
  puts "a is bigger than b"  
end
```

```
puts "a is bigger than b" if a > b  
puts "a is bigger than b" unless a < b
```

- 寫在後面看起來很精簡，但是需要考慮到可讀性。

迴圈(Loop)

times (l)

- 如果要重複執行的次數是確定的時候，可以用times。
- 語法：

```
重複次數.times {  
    要重複執行的動作  
}
```

times (II)

- (範例)times.rb

```
10.times { |i| puts i }
```

- 輸出的結果是0到9，因為i是從0開始的。
- 在範例中的「{ ~ }」也可以改用「do ~ end」。
- 慣例是單行用「{}」，多行用「do ~ end」

for (l)

- 語法：

for 變數 in 開始的數值..結束的數值 do
 要重複執行的動作
end

```
sum = 0
for i in 1..5 do
    sum = sum + i
end
puts sum
```

for (II)

- 範圍，不是只有整數，也可以是字串。
- 範例

```
sum = ""  
for c in 'a'..'z'  
    sum += c  
end  
puts sum
```

for (III)

- (1..5) 與 ('a'..'z') 是Range物件。
- Range指的是一組值，一個序列，可能是數字，也可能是文字，只要有順序的開頭和結尾，就可以使用Range。

while

- 語法：

while 條件 [do]

 要重複執行的動作

end

i = 1

while i <= 10 do

 puts i

 i += 1

end

until

- 語法：

until 條件 [do]

要重複執行的動作

end

i = 1

until i > 10 do

 puts i

 i += 1

end

each ()

- each是集合物件用來逐項取得資料，並加以處理的方法。
- 常見的集合物件有Array, Hash, Range。
- 語法：

```
集合物件.each do |變數|
  要重複執行的動作
end
```

each (II)

```
sum = 0
(1..5).each do |i|
  sum += i
end

puts sum
```

迴圈的控制

- 在迴圈的途中，有時會需要跳離迴圈，或是直接跳到下一圈。Ruby提供了三個命令做這些控制。
- `break`: 停止動作，馬上跳出迴圈。
- `next`: 直接跳到迴圈的下一圈。
- `redo`: 以相同的條件重新進行這一圈。

方法(method)

定義方法

- 語法：

```
def 方法名( 參數1, 參數2 ... )  
    想要做的動作  
end
```

```
def hello(name) # 定義method  
    puts "Hello #{name}!"  
end
```

```
hello("Wei Jen") #呼叫method
```

參數的預設值 (I)

- 參數可以指定預設值。當有多個參數時不可以跳著指定預設值，有指定預設值的參數必須放在最後。

參數的預設值 (II)

- 範例：

```
def hello(name, language = "Ruby")
  puts "Hello #{name}!,
        I am using #{language}"
end
hello("Wei Jen", "Java")
hello("Wei Jen")
```

方法的回傳值(I)

- 在方法中，我們可以用**return**來指定回傳值。
- 範例

```
def volume(x, y, z)
    return x * y * z
end
p volume(2, 3, 4)
p volume(10, 20, 30)
```

方法的回傳值(II)

- 如果沒有寫return，那方法中，最後被執行的statement的回傳值，會是該方法的回傳值。
- 範例

```
def volume(x, y, z)
    x * y * z
end
p volume(2, 3, 4)
p volume(10, 20, 30)
```

使用其他檔案

讀入其他檔案(I)

- 我們常會需要使用其他檔案中的方法或類別，我們可以利用require方法，來將其他檔案載入。
- (範例)hello.rb

```
def hello
    puts "Hello Ruby!"
end
```

讀入其他檔案(II)

- (範例)call_hello.rb

```
require 'hello'
```

```
hello #執行在hello.rb中的方法
```

類別與物件

什麼是類別(Class) |

- 類別是用來表示物件(Object)的「類型」，有點像是「模子」。
- 類別決定物件的行為模式。
 - 100是屬於Fixnum類別的物件，可以進行四則運算。
 - [1, 2, 3]是屬於Array類別的物件，沒有定義乘、除的行為。

什麼是類別(Class) II

- 我們可以用Object#class方法來看某物件所屬的類別。

```
puts [1, 2, 3].class  #=> Array  
puts 100.class        #=> Fixnum
```

Ruby 文件中表示method的慣例：

實體方法： Object#class

類別方法： Object.superclass

什麼是類別(Class) III

- 我們可以利用`instance_of?`方法來判斷某個物件是否屬於某個類別。

```
puts [1, 2, 3].instance_of?(Array)
puts [1, 2, 3].instance_of?(Fixnum)
```

繼承(inheritance) |

- 擴充既有的類別，定義出新的類別。

很久以前，有一個人定義了車子（父類別）要有四個輪子、一個引擎、一個車殼。

Benz的工程師利用原有的車子的模子，做出了S350的模子（子類別）。

然後你們公司股票飆上1000元，所以你賣了5張股票，去買一台S350（物件）。

繼承(inheritance) II

- 繼承可以做到下列事：
 - 保留父類別的所有功能，並追加新功能。
 - 重新定義原有功能，使相同名稱的方法有不同的行為。
 - Ruby中所有類別都是Object類別的子類別，Object的父類別是BasicObject，BasicObject是唯一沒有父類別的類別。

繼承(inheritance) III

- 我們可以用Object#is_a?方法來檢查有沒有繼承的關係。
- 範例

```
a = [1, 2, 3]
puts a.is_a?(Array)      #=> true
puts a.is_a?(Fixnum)     #=> false
puts a.is_a?(Object)     #=> true
puts a.is_a?(BasicObject) #=> true
```

繼承(inheritance) IV

- 我們可以用Class.superclass來知道該類別的父類別。
- 範例

```
a = [1, 2, 3]
```

```
p a.class
```

```
p a.class.superclass
```

```
p a.class.superclass.superclass
```

```
p a.class.superclass.superclass.superclass
```

class

- 語法：

class 類別名

 類別定義

end

- 類別名一定要以大寫字母開始。

定義自己的類別(I)

```
class HelloWorld
  def initialize(myname = "Ruby")
    @name = myname
  end

  def hello
    puts "Hello World, I am #{@name}"
  end
end
```

定義自己的類別(II)

```
bob = HelloWorld.new("Bob")
alice = HelloWorld.new("Alice")
ruby = HelloWorld.new

bob.hello    #=> Hello World, I am Bob
alice.hello  #=> Hello World, I am Alice
ruby.hello   #=> Hello World, I am Ruby
```

initialize方法

- 當我們呼叫new方法建立物件時，這個initialize方法會被呼叫。同時傳給new的參數，都會傳給initialize方法

實體變數 (Instance Variable)

- 以@開始的變數稱為實體變數。
- 與區域變數不同，實體變數的值在離開方法後仍然存在。
- 在每一個物件(實體)中，實體變數會各自不同。

實體方法 (Instance Method)

- 實體變數可以被實體方法存取。
- 在這個範例中，hello方法，可以從物件外部呼叫。所以可以呼叫 bob.hello

存取方法 (Access Method)

- Ruby不允許物件外部直接存取實體變數，所以要存取物件內部的資料，都需要定義方法來操作。

```
def name
  return @name
end
```

```
def name=(new_name)
  @name = new_name
end
```

存取方法 (Access Method)

- Ruby不允許物件外部直接存取實體變數，所以要存取物件內部的資料，都需要定義方法來操作。

```
def name
  return @name
end
```

```
def name=(new_name)
  @name = new_name
end
```

但是，如果每個實體變數都要這樣寫，我寫Java就好了。

存取方法 (Access Method)

- Ruby 提供了快速定義存取方法的方式
 - 只定義用來讀取的方法
 - `attr_reader :name`
 - 只定義用來寫入的方法
 - `attr_writer :name`
 - 同時定義讀取與寫入的方法
 - `attr_accessor :name`

類別方法 (Class Method)

- 當接收者不是物件而是類別時，這個方法稱為類別方法。
- 例如：

```
ruby = HelloWorld.new
f = File.open("some_file")
t = Time.now
```

定義類別方法 (I)

- 語法：

```
def self.方法名  
    方法內容  
end
```

- 範例

```
class HelloWorld  
    def self.hello(name)  
        puts "Hello #{name}" #name是區域變數  
    end  
end
```

定義類別方法 (II)

- 語法：

```
def 類別名.方法名  
    方法內容  
end
```

- 範例

```
def HelloWorld.hello(name)  
    puts "Hello #{name}"  
end
```

定義類別方法 (III)

- 語法：

```
class << 類別名
  def 類別名.方法名
    方法內容
  end
end
```

- 範例

```
class << HelloWorld
  def HelloWorld.hello(name)
    puts "Hello #{name}"
  end
end
```

常數(Constant)

- 類別內可以定義常數。常數名稱的慣例是第一個字母大寫。
- 範例：

```
class HelloWorld
    Version = 1.0
end
```
- 從外部讀取的方式：
 - `p HelloWorld::Version #=> 1.0`

類別變數 (Class Variable)

- 以@@開始的變數是類別變數。
- 類別變數是該類別所有實體共用的變數。很類似常數可是變數值是可以被修改的。（不會有警告訊息）
- 要少用。

擴充類別

- 我們可以在原本已經定義好的類別中新增方法。危險動作，請小心服用。

```
class String
  # 將自己(字串)以空白字元分解
  def count_word
    ary = self.split(/\s+/)
    return ary.size
  end
end
```

繼承的實作方式(I)

- 語法：

```
class 類別名 < 父類別  
    類別內容  
end
```

繼承的實作方式(II)

```
class WeekArray < Array
  def [](i)      # 實作[] method 的方式
    idx = i % 7
    super(idx)   # 會使用父類別同名的方法
  end
end
wa = WeekArray['Mon', 'Tue', 'Wed',
               'Thu', 'Fri', 'Sat', 'Sun']
p wa[2]          #=> Wed
p wa[10]         #=> Thu
```

限制方法的呼叫()

- 預設所有的方法都是public。
- Ruby提供了三種限制層級：

public	將方法公開為外部可以使用的實體方法
private	將方法限制為只有內部可以使用（不允許接在接收者物件後面呼叫）
protected	將方法限制為只有內部可以使用，但是在同一個類別內可作為實體方法使用。

限制方法的呼叫(II)

```
class AccTest
  def call_priv
    priv
  end

  private
  def priv
    puts "priv is a private method"
  end
end

at = AccTest.new
at.call_priv
at.priv
```

#=>會有錯誤訊息

限制方法的呼叫(III)

```
class AccText
  def call_prot(at)
    at.prot
  end
  protected
  def prot
    puts "This is a protected method"
  end
end

a = AccText.new
b = AccText.new
a.call_prot(b)
a.prot
```

#=>會有錯誤訊息

模組(Module)

模組簡介

- 模組提供一種方式讓你將方法、類別與常數集合在一起。
- 優點：
 - 模組讓你可以定義命名空間(namespace)，讓你避免命名衝突。
 - 模組提供了mixin的功能。

命名空間 (Namespace)

- 命名空間是為了讓方法、常數、類別名稱不互相衝突而設計的機制。

```
module Trig
  class Math
    def self.count
      ...
    end
  end

  def Trig.count
    ...
  end
end
```

Mixin (I)

- 模組是不能被**實體化**(instance)的，就是不能單獨成為一個物件。因為模組並不是一個類別。
- 但是你可以在定義類別的時候，加入一個模組。這時，模組中的實體方法都會成為該類別中的實體方法。

Mixin (II)

```
module Debug
  def who_am_i
    "#{@self.class.name}( id:
    #{@self.object_id})"
  end
end
```

- `self.class.name` 會將類別名稱轉成字串
- 每個物件都有自己的唯一的id，你可以利用`Object#object_id`來取得。

Mixin (III)

```
class HelloModule
  include Debug # 將模組包含進來
end

a = HelloModule.new
puts a.who_am_i
#=> HelloModule( id: 2000 )
```

關於include ()

- Ruby的include statement並不會複製一組module的實體方法給類別。而是只有一個module，而所有include這個module的類別都參照(reference)到這個module。
- 而在module中所定義的實體變數，則會在每一個物件實體化時，各自建立一份。

關於include (II)

- 建議：
 - 在模組中的實體變數只有該模組的實體方法使用。
 - 該模組的實體方法，也只使用在該模組中定義的實體變數，不要使用到類別中的實體變數。

Array

建立陣列

- 用[]來建立陣列

```
num = [1, 2, 3]
```

```
mix = [1, "a", Time.now]
```

- 用Array.new來建立陣列

```
a = Array.new      #=> []
```

```
a = Array.new(3)    #=> [nil, nil, nil]
```

```
a = Array.new(3, 0)  #=> [0, 0, 0]
```

取得元素

- 索引以0開始計算。

```
a = [1, 'a', Date.today]
```

```
p a[0]      => 1
p a[-1]     => 2013-05-20
p a[1..2]   => ["a", 2013-05-20]
p a[0,2]    => [1, "a"]
```

改寫元素

```
a = [1, 'a', Time.now]
```

```
a[2] = "b"  
p a  #=> [1, "a", "b"]
```

```
a[1..2] = ["A", "B"]  
p a  #=> [1, "A", "B"]
```

```
a[1, 2] = ["C", "D"]  
p a  #=> [1, "C", "D"]
```

相加、交集與聯集

- 相加 : $\text{array} = \text{array1} + \text{array2}$
- 交集 : $\text{array} = \text{array1} \& \text{array2}$
- 聯集 : $\text{array} = \text{array1} | \text{array2}$

Array#each

- 陣列可以使用each方法依序取得每一個元素，並進行所需的處理。

```
word = ""  
ary = %w(a b c d e)  
  
ary.each do |el|  
  word += el  
end  
p word #=>"abcde"
```

Array#each_with_index

- Array可以使用each_with_index方法來同時取得元素與索引值。

```
ary = %w(a b c d e)
```

```
ary.each_with_index do |ele, index|
  puts "#{index}: #{ele}"
end
```

Hash

雜湊(Hash)的操作 (I)

- 雜湊是一群由key/value pair所組成的集合物件。
- key與value可以是任意物件。
- 建立：

```
hash = { :a => "Mon", :b=>"Tue", :c => "Wed"}
```

- 取得value：

```
p hash[:a] #=> "Mon"
```

湊(Hash)的操作 (II)

- 修改value

```
hash[:a] = "Today"
```

- 刪除value

```
hash[:a] = nil
```

```
hash.delete(:a) # 這兩個其實是有點不同的喔
```

湊(Hash)的操作 (III)

- 檢查某個key是否存在

`hash.include?(:b)`

`hash.key?(:a)`

#這裡可以解釋剛剛刪除value的部份的不同處

- 檢查某個value是否存在

`hash.value?("Tue")`

湊(Hash)的操作 (IV)

- 查詢雜湊大小

hash.size

hash.length

- each:

```
hash.each do |key, value|
  puts "#{key}: #{value}"
end
```

Exception

例外處理(I)

- 語法：

begin

一般的處理動作

rescue

例外發生時的處理方式

end

例外處理 (I)

- 語法：

begin

一般的處理動作

rescue

例外發生時的處理方式

end

begin ~ end 內部定義的區域變數，無法在
begin ~ end 外部使用。

例外處理 (II)

```
begin
  File.open("no\file")
rescue
  puts "Some error happened"
  puts $!      #最後發生的例外(例外物件)
  puts $@      #最後例外所發生的位置相關資訊
end
```

例外處理 (III)

- 語法：

begin

一般的處理動作

rescue => 用來存放例外物件的變數

例外發生時的處理方式

end

例外處理 (IV)

```
begin
  File.open("no\file")
rescue => ex
  puts ex.class      #=> Errno::ENOENT
  puts ex.message
  #被呼叫的順序與所屬檔案的行數
  puts ex.backtrace
end
```

善後 (I)

- 如果有些動作無論是否發生例外都需要確實完成，則寫在ensure區塊中。
- 語法：`begin`

一般的處理動作

`rescue =>變數`

例外發生時的處理方式

`ensure`

無論有無發生例外，都會執行的處理

`end`

善後 (II)

```
begin
  File.open("no/file")
rescue => ex
  puts ex.message
ensure
  # 無論檔案是否開啟成功，這行都會印出來
  puts "File closed"
end
```

重試 (I)

- 在rescue區塊內，可以使用**retry**敘述重新執行begin區塊內的動作。
- 如果例外一直沒被排除（例如檔案就是一直打不開）會變成無窮迴圈，所以建議要設定重試次數。

重試 (II)

```
try_count = 0
begin
  File.open("no/file")
rescue => ex
  if try_count < 3
    try_count += 1
    retry
  end
  puts try_count
  puts ex.message
end
```

練習

- 19 * 19乘法表
- 請建立MyTime的類別儲存時間資料
 - 這個類別擁有三個成員變數hour,minute,second儲存小時、分和秒的資料。
 - 建立建構子Time.new
 - 建立讀取時間資料的方法
 - 建立printTime()方法顯示時間資料，validateTime()方法可以檢查時間資料。

練習

- 建立一個MyArray類別，類別中有一個實體變數ary是一個陣列，預設有10個數字。並提供sort方法，可以將ary的內容排序，並印出。
- 建立一個Hash，並把原本的key變成value，把value變成key。（請把每一個value也都設成唯一）