Ruby 基礎運算與指令Last updated: 2017-10-02 14:02

學習了解與如何輸入 Ruby 的運算指令來讓電腦計算出結果並印出

**這個單元的影片是沒有聲音的！  
Warning: the videos in the unit have no sound!**

本單元學習目標是了解 Ruby 如何進行運算與回傳（列出）結果值並能實際進行操作。跟別的程式語言相比，Ruby 語法對於任何的輸入都會給予結果回傳，所以我們先了解回傳值 (Return value) 的定義：

回傳值 (Return value)

所謂回傳值是透過我們輸入的文字或數字，或是需要計算的變數值等，經由某個程式或函數處理後 , 將結果交給 (return) 原呼叫程式的值。Ruby 語法的回傳值，會以 => 符號來表示，後面接對應回傳的結果值。在 Ruby 語法中，任何一項輸入都會有回傳值 (Return value) 代表結果值或結束。即使是 puts 指令，也會有 nil 這個回傳值代表結束。

nil 又是什麼？

nil 表示一個空值，若用英文就是 "nothing"，中文就是 "無" 的意思，但是在 Ruby 語法裡 nil 是一個存在的值，只是這個值是空的，還是可以用來比較，例如：nil == nil 會回傳 true；同時 nil 也不等於 false，nil == false 會回傳 false。

運算子（ Operator ）

四則運算

| **表示符號** | **中文** | **說明** |
| --- | --- | --- |
| ＋ | 加 | 表示相加，例如：2 + 3 |
| - | 減 | 表示相減，例如：2 - 1 |
| ＊ | 乘 | 表示相乘，程式的乘號以＊表示，而非 X 符號。例如：3 \* 3 |
| / | 相除後取整數 | 表示相除後取整數，例如: 157 / 10 = 15 (後面 .7 刪除不要) |
| % | 相除後取餘數 | 表示相除後取剩下的餘數值，例如: 157 ％ 10 = 7 |

邏輯判斷

| **表示符號** | **中文** | **說明** |
| --- | --- | --- |
| > | 大於 | 表示大於某數值，例如：3 > 2 |
| < | 小於 | 表示小於某數值，例如：4 < 5 |
| >= | 大於等於 | 表示大於等於某數值，例如：3 >= 3 |
| <= | 小於等於 | 表示小於等於某數值，例如：3 <= 3 |
| == | 等於 | 運算的等於符號會是以兩個等於符號來表示精準等於的意思。例如：150 == 150。若是單一個等於符號 (例如：150 = 150)，程式會將右邊的 150 分配 (assign) 給左邊的 150，這樣就會產生語法錯誤，電腦不會判別是分配 (asssign) 還是邏輯判斷，所以會回傳 "syntax error" 語法有問題。 |
| true | 真 | 布林值。若判斷為真時會回傳 true 表示。例如：3 >= 3，會回傳 => true |
| false | 假 | 布林值。若判斷為假時會回傳 false 表示。例如：2 > 3，會回傳 => false |
| ! | 否 | 這個符號會將判斷的結果進行反轉，例如：!true = false，!false = true |
| != | 不等於 | 驚嘆號的邏輯判斷為否，加上等於的符號就是不等於；用來判斷左右數值是否不相等，不相等時會回傳 true。例如：3 != 4 會回傳 => true，3 != 3 會回傳 => false |
| || | 或 | 這個符號用來串接兩個邏輯判斷，其中一個為真，整體判斷就為真，比如說：X 是否是 2 或 11？換成程式碼就會是：X == 2 || X == 11，只要其中一邊（X == 2 或 X == 11）的結果為 true，整個邏輯判斷的結果就是 true。 |
| && | 和 | 這個符號用來串接兩個邏輯判斷，必須兩個為真，整體判斷才為真，比如說：X 是否大於 3 但小於 5？換成程式碼就會是：X > 3 && X < 5，必須兩邊（ X > 3 或 X < 5 ）的結果同時為 true，整個邏輯判斷結果才會是 true。 |

字串 (string) 與插值 (interpolation)

| **表示符號** | **中文** | **說明** |
| --- | --- | --- |
| " " | 字串 (string) | 字串的內容通常用 " " 雙引號夾起來。例如 "2" 是字串而非數字。 |
| #{ } | 插值 (interpolation) | 可以使用雙引號將字串與變數 (variables) 或 Ruby 的語法組合。這稱為「插值」(Interpolation)。 在雙引號內的字串中若有 #{運算式}，則先執行運算式，然後將結果取代 #{運算式}，例如："two plus two is #{2+2}"。注意，插值在單引號內是無效的。 |

puts and print

| **表示符號** | **中文** | **說明 (請參照以下圖片)** |
| --- | --- | --- |
| puts | 印出 | 印出所輸入或經過計算的值，並且會新增斷行，印出 => nil 表示結束 (如下圖) |
| print | 印出 | 印出所輸入或經過計算的值，不會有斷行，接著印出 => nil 表示結束 (如下圖) |

用流程控制、迴圈解題Last updated: 2017-10-11 18:58

此單元你將學會透過 irb 實地操作與應用 for 迴圈來找出奇數與偶數

前言：



*Source：*[*http://bartsblackboard.com/files/2012/03/Barts-earned-a-day-off.png*](http://bartsblackboard.com/files/2012/03/Barts-earned-a-day-off.png)

在這個單元我們會透過影片學習以 irb 操作流程控制 (Control Flow) 與迴圈 (For) 的實際應用，關於這兩個元素的介紹與 Ruby 基本語法，請參閱 [Ruby 基礎運算與指令](https://lighthouse.alphacamp.co/units/250) 與 [資料結構](https://lighthouse.alphacamp.co/units/251) 這兩個單元。

流程控制 (Control Flow): 如果...不然就 (if ... else ... end)

Ruby 的流程控制(Control Flow)中常常會看到 if 判斷式。 if 可以搭配 else 及 elsif (注意少了一個"e") 。我們來仔細看看這兩個條件判斷語句。

if ... end 條件判斷式

if 就是如果的意思，如果條件判斷式為真，則往下執行。例如以下範例：

*# 判斷 2 > 1 嗎？*

if **2** > **1**  *# 如果 2 > 1*

puts true *# 印出 true*

end

請記住，最後要加上 end 來結束 if 判斷。

if ...else... end

if 如果 A 條件為真，則往下執行，不然就 (else) B 條件。讓我們看以下範例：

*# 判斷是男性 (male) 還是女性 (female)？*

gender = "female"  *# 設定性別是女性*

if gender == "male" *# 如果性別是男性*

puts "male"  *# 印出 male*

else

puts "female"  *# 印出 female*

end

以上的範例判斷是男性還是女性。我們先透過=來指定 (assign) 性別 gender 這個變數裡的值是女性 “female”，下一句 if 會判斷 gender 是否等於男性 "male"? 顯然答案是不相等，所以執行 else 的條件，於是印出 "female" 。請記住，最後要加上 end 來結束 if 判斷。

if ...elsif ... else... end

需要判斷的條件如果是超過二種以上，也可以使用 elsif 來增加條件判斷。這邊要留心語法 elsif，中間的確少了一個 "e" 。 讓我們看以下範例

*# 用 BMI 來判斷身體體質狀況？*

bmi = 26  *# 設定 bmi 為 26*

if bmi > **0** && bmi < **18**  *# 如果 bmi > 0 與 < 18 之間*

puts "體重過輕"  *# 印出體重過輕*

elsif bmi >= **18** && bmi < **24**  *# 如果 bmi >= 18 與 < 24 之間*

puts "正常體重"  *# 印出正常體重*

elsif bmi >= **24** && bmi < **27**  *# 如果 bmi >= 24 與 < 27 之間*

puts "體重過重，要開始注意喔！" *# 印出體重過重，要開始注意喔！*

else

puts "肥胖！請立刻執行健康體重管理" *# 印出肥胖！請立刻執行健康體重管理*

end

以上的範例，試著說明整個流程控制的使用，若 bmi 被設定為 26 時，程式會開始往下執行到

elsif bmi >= **24** && bmi < **27**

這一段發現條件式成立(判斷為真)，因此印出結果是「體重過重，要開始注意喔！」注意我們是透過 && 把兩個條件連在一起。&& 代表兩個條件同時要符合。如果只需要符合其中一個條件，可以用前面介紹過的 || 來執行。例如：

if bmi < **18** || bmi > **27** *# 如果 bmi < 18 或是 > 27*

puts "你身體過輕或肥胖，要注意哦！"  *# 你身體過輕或肥胖，要注意哦！*

end

最後請記住，最後要加上 end 來結束 if 判斷。

控制結構 Case

如果 if / else 的條件太多，可考慮用 case 來敘述表達一連串的條件判斷與回應。例如以下範例：輸入不同的名字會得到不同的回覆。

print "Enter your name: "

name = gets.chomp

case name *# 在 name 的結構中*

when "John" *# 當 name 是 John 的時候*

puts "Howdy John!" *# 印出 Howdy John!*

when "Ryan" *# 當 name 是 Ryan 的時候*

puts "Whatz up Ryan!" *# 印出 Whatz up Ryan!*

else *# 其他的名字*

puts "Hi #{name}!" *# 則是印出 Hi <輸入的名字>!*

end

迴圈 (Loop) 基本應用：for 和 times

用 for 這個迴圈主要搭配限定的執行次數來運作。所以 irb 使用 for，就要給定次數 (times)，或者是一個範圍的設定，例如：從 1 數到 10，總共數了 10 次，或是 1.2.3...10 這個範圍。讓我們透過影片來跟著操作。

看完影片後，讓我們對不同迴圈 (loops) 與迭代 (iteration) 做點介紹。注意，在這裡我們只會用一些比較簡單的例子去講解，我們也不會把所有語法工具介紹。去學習一個程式語言的時候，我們建議先熟悉幾個常見的語法工具，然後不斷練習。這樣比一下子去學習所有工具該如何應用更為有效。

迴圈和迭代（Loops and Iteration）實務

下列是 Ruby 不同迴圈工具的介紹。程式碼範例都是在執行類似的操作，就是把印出從 0 到 99 或是從 1 到 100. 注意當使用不同迴圈工具的時候，需然是做類似的事情，但邏輯的寫法會不一樣。

For 迴圈

For 迴圈需要設定一個迴圈的範圍（0 到 99），並指定每次要執行的動作；適合用在把一件事情重複執行，或是針對一系列資料做簡單的運算與判斷。For 迴圈一般會透過一個參數去數迴圈的範圍。在處理一個陣列 (array) 的資料時，我們很多時候會用這個參數來當陣列的索引 (index)。所以 For 迴圈是處理陣列的好工具。

*# 輸出 0 到 99 的數字*

for i in **0.**.**99** *# 讓 i 從 0 到 99 - ".." 會包括最大值 99*

puts i *# 每次操作印出 i，印出 0 到 99*

end

for i in **0.**..**100**  *# 讓 i 從 0 到 99 - "..." 不包括最大值的 100*

puts i  *# 每次操作印出 i，印出 0 到 99*

end

Times 迴圈

Times 迴圈需要設定一個重複的次數，它會不斷進行重複的操作，直到次數到達；但必須另外設定變數進行判斷和控制。

i = **0**

**100**.times do

i = i + **1** *# 每次操作讓 i + 1，從 0 加到 100*

puts i *# 每次操作印出 i，印出 1 到 100*

end

While 和 Until 迴圈

while 和 until 迴圈一定有判斷中止的條件，也必須預先設定好判斷用的變數，一旦條件滿足就會中止迴圈，但 until 只能針對一件事情做多次，無法像其他迴圈一樣執行多件事情 (計算、印出)。

while 適合用來重覆執行沒有明確數量的事情，比如說：設計一個「剪刀、石頭、布的遊戲」如果雙方平手，會一直玩下去，直到分勝負為止。當使用 while 的時候，要非常小心去設計中止的條件，不然會陷入「無限迴圈」的狀況。

until 則適合做簡單的重複計算，比如說：計算出 6 的 5 次方。

i = **0**

while ( i < **100** ) *# 當 i 小於 100，會不斷重複操作*

i += **1** *# 每次操作 i 會加 1*

puts i *# 每次操作印出 i*

end

*# 對 i 做 101 次的 +1 才印出*

i = **0**

i += **1** until i > **100** *# i 會加 100 次，直到 101 才判斷要停下來*

puts i *# 印出為 101*

迭代（Iteration）Each

相較在迴圈裡要進行邏輯判斷，迭代通常用在對陣列 (array) 的所有元素進行計算或處理；適合用在每個元素都必須進行處理的時候，如：所有學生的成績都要加 10 分。 .each 的用法是寫在一個陣列之後。

*# 用 .each 印出陣列（ array ）的所有內容*

nums = [**1**,**2**,**3**,**4**,**5**]

nums.each do |i| *# 讓 i 從 1 到 5*

puts i *# 每次操作輸出 i*

end

實際應用：找出偶數與奇數

假設有一數列是從 1.2.3...100，要從其中挑出哪些是偶數，哪些是奇數，我們在 irb 中該怎麼下指令呢？讓我們跟著影片學習。

在找偶數的影片裏，共在 Terminal 用了三種指令：touch，subl (Ubuntu 請用 atom)，ruby  
注意：這些指令不適用在 irb 的環境！

1）使用 touch 指令創建一個新的名為 even\_number.rb 檔案

**Terminal**

$ touch even\_number.rb

2）使用 subl (Ubuntu 請用 atom) 指令打開 even\_number.rb 檔案

**Terminal**

$ subl even\_number.rb

3）使用 ruby 指令執行 even\_number.rb 的程式碼

**Terminal**

$ ruby even\_number.rb

小技巧：影片最後只跑出 even

在撰寫程式碼的過程中，工程師常會使用一些小技巧來提升視覺上的效果，也方便檢查輸出的結果值是否正確。例如：以下程式碼我們加入了一行 print "n=", n, " :"

for n in **1**..**100**

print "n=",n," :" *# 印出 n 的值*

if n % **2** == **0**

puts "even" *# 如果 n 偶數，印出 even*

else

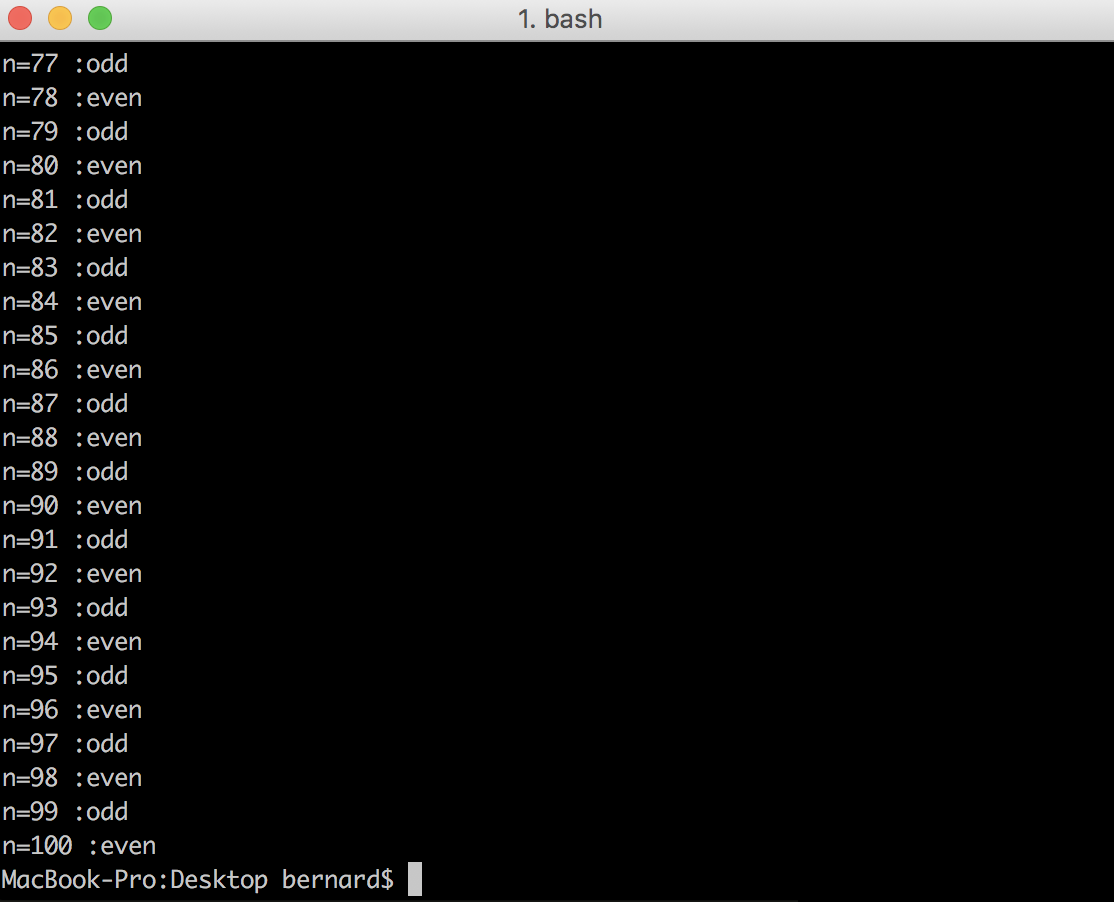
puts "odd" *# 如果 n 奇數，印出 odd*

end

end

其中 "n=" 是字串 (string)，中間的 n 是變數，然後 " :" 也是是字串。電腦會把字串直接顯示，但變數是動態的，電腦會計算並顯示適當的值。

加了這一行程式碼後，輸出的效果就如同下圖，是不是易於閱讀與檢查呢？



找偶數，然後放進一個陣列 (array)：

這邊我們加入使用陣列 (array) 加入幾行程式碼，首先宣告變數 array 是一個陣列 [ ]，然後中間插入 array [i] = n 的程式碼，意即將將偶數的數字存入 array 這個陣列。最後 puts array 就會發現，只會印出偶數。

以下程式碼提供參考：

array = []

i = **0**

for n in **1**..**100**

print "n=",n," :"

if n % **2** == **0**

puts "even"

array [i] = n

i = i+**1**

else

puts "odd"

end

end

puts array