

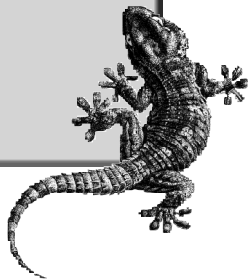
---

# 第四章

# 控制結構

本章內容：

- \* 敘述區段
- \* if/unless
- \* while/until
- \* do {} while/until
- \* for
- \* foreach
- \* 習題



## 敘述區段

敘述區段 (statement block) 是由一對大括號，把一連串敘述包起來所構成，看起來應該像這樣：

來應該像這樣：

```
{  
    第一行敘述 ;  
    第二行敘述 ;  
    ...  
    最後一行敘述 ;  
}
```

Perl 會依序執行其中的敘述。稍後我們會介紹改變執行順序的方法，請耐心看下去。

文法上，敘述區段可以放在任何可出現單一敘述之處；反之不成立。

## 第四章

---

敘述區段內最後一個敘述的分號可以省略。為了方便以後加入新的敘述，除非只有一行，否則我們還是會加上分號。讀者可以比較下面這二種寫法：

```
if($ready)      { $hungry++;}
if($tired) {
    $sleep= {$hungry+1}*2;
}
```

## if/unless

if 敘述可以選擇程式的「走向」。你還可以加上對應的 else 敘述，像這樣：

```
if(some_expression) {
    true_statement1;      # 條件成立的情況下
    true_statement2;      # 所要執行的敘述
    true_statement3;
} else {
    false_statement1;     # 條件不成立的情況下
    false_statement2;     # 所該處理的事
    false_statement3;
}
```

(如果你對 C 或 Java 語言很熟，應該知道括號是少不了的。)

如果 if 括號內的敘述 (some\_expression) 為真，第一個區段會被執行 (true\_statementX)。若為假，會執行第二個區段 (false\_statementX)。

不過，究竟真假要如何判定呢？Perl 的判定方法其實有點奇怪，不過結果仍然是我們直覺上所預期的。控制敘述會在純量環境下以字串取值；如果敘述是純量，會被轉換成字串（註）。若該字串是“0”或空字串（長度為0），此敘述之值會被認定為假。除此之外其它情形都是真。這種判斷方法雖然聽起來很好玩，不過對系統來說非常方便，因為只要分辨空字串、“0”和非空字串即可，不必另外設計一套機制判定數字的真假。下面是一些例子：

---

註 其實系統內部的作法不是這樣，不過「看起來」卻是如此。

```
0                # 轉換成 "0", 故為假
1-1              # 同上
1                # 轉換成 "1", 故為真
""               # 空字串, 假
"1"              # 非 "" 亦非 "0", 故真
"00"             # 同上 ( 有點奇怪, 請注意 )
"0.000"          # 同上
undef            # 取值為 "", 故假
```

實際上說起來，值的真假判定是非常直覺的，不要被這些理論嚇到了。

下面是 if 敘述的完整使用例：

```
print "how old are you? ";
$a = <STDIN>;
chomp($a);
if ($a < 18) {
    print "So, you're not old enough to vote, eh?\n";
} else {
    print "Old enough! Cool! So go vote!\n";
    $voter++; # count the voters for later
}
```

else 區段在語法上可以省略，像這樣：

```
print "how old are you? ";
$a = <STDIN>;
chomp($a);
if ($a < 18) {
    print "So, you're not old enough to vote, eh?\n";
}
```

有時，我們「只要」else 區段，不過這在語意上有點彆扭，所以 Perl 提供了 unless 敘述，使用方法如下：

## 第四章

---

```
print "how old are you? ";
$a = <STDIN>;
chomp($a);
unless ($a < 18) {
    print "Old enough! Cool! So go vote!\n";
    $voter++;
}
```

unless 也可以有 else 區段，意思和「正面來說」的 if 一樣。如果你想做多重選擇，可以用 elsif 敘述，使用方法如下：

```
if (some_expression_one) {
    one_true_statement_1;
    one_true_statement_2;
    one_true_statement_3;
} elsif (some_expression_two) {
    two_true_statement_1;
    two_true_statement_2;
    two_true_statement_3;
} elsif (some_expression_three) {
    three_true_statement_1;
    three_true_statement_2;
    three_true_statement_3;
} else {
    all_false_statement_1;
    all_false_statement_2;
    all_false_statement_3;
}
```

若某判斷敘述為真，對應區段裡的每個敘述會被依序執行，其它分支判斷及其區段則會忽略。如果所有判斷敘述均為假，就會執行 else 的區段（如果有的話）。你可以依需要增減 elsif 的數目。

## while/until

迴圈是程式語言的要件之一（註）。在 Perl 中，你可以用 while 敘述製造迴圈，用法如下：

```
while(some_expression) {
    statement_1;
    statement_2;
    statement_3;
}
```

程式執行到 while 敘述時，會檢查控制敘述（some\_expression），若其值為真，會執行迴圈本體一次。如此不停反覆下去，直到控制敘述的值為假，Perl 才繼續執行 while 迴圈的下一個敘述。請看下面的例子：

```
print "how old are you?";
$a = <STDIN>;
chomp($a);
while($a >0) {
    print"At one time, you were $a years old.\n";
    $a--;
}
```

某些時候，我們說「直到某事為真」會比「當某事為假」來得順口。Perl 提供的 until 敘述正可以做到這件事：

```
until (some_expression) {
    statement_1;
    statement_2;
    statement_3;
}
```

無論 while 或 until，一旦控制敘述之值讓迴圈終止，就會完全跳過迴圈內的敘述。以上面輸入年齡的程式來說，若使用者鍵入小於 0 的數，那麼迴圈一次都不會執行。

---

註 所以 HTML 不能算是一種程式語言。

## 第四章

---

Perl 允許我們製造無窮迴圈；這不但是合法的，有時也是因為實際應用的需要，所以不會被當成是錯誤。例如，你可以寫一個程式，如果不產生錯誤，就讓迴圈不停重覆，並在迴圈後面寫處理錯誤的程式。這是 `daemon`（註1）的典型寫法，可讓它不停執行直到系統當機。

### do {} while/until

前面所介紹的 `while/until` 會在進入迴圈之前檢查條件敘述；如果該條件一開始即為假，裡面的敘述一次都不會被執行到。但有些情況會要先做一次，再形判斷。

以這種情況來說，就必須用到 `do{} while` 敘述了。此敘述的用法和 `while` 很像（註2），只是它至少會執行一次迴圈本體，然後才檢查條件：

```
do {
    statement_1;
    statement_2;
    statement_3;
} while (some_expression);
```

Perl 會先執行 `do` 區段裡的敘述，最後再檢查條件。若條件式為假，迴圈即結束；否則迴圈會再進行一次。

你也可以用和 `while` 類似的想法把 `do{} while` 換成 `do{} until`。請看下面的例子：

```
$stops = 0;
do {
    $stops++;
    print "Next stop? ";
    chomp($location = <STDIN>);
} until $stops > 5 || $location eq 'home';
```

---

註1 熟悉 Unix 系統的讀者為很清楚它是甚麼，就功能而言它有點像 NT 裡的“服務”

註2 其實並不「很像」；這個問題在第九章會詳細討論。

## for

Perl 另外還有一個迴圈敘述 `--for`，用法和 C 或 Java 的幾乎沒什麼兩樣。請看：

```
for ( 起始敘述 ; 測試敘述 ; 步進敘述 ){  
    statement_1;  
    statement_2;  
    statement_3;  
}
```

上面的迴圈可以用 `while` 改寫如下：

```
起始敘述 ;  
while( 測試敘述 )  
{  
    statement_1;  
    statement_2;  
    statement_3;  
    步進敘述 ;  
}
```

首先會執行起始敘述。一般可在這裡為變數設值，不過這不是硬性規定；你甚至可以什麼都不寫 -- 但分號還是要寫。測試敘述之值若為真，迴圈本體會被執行一次，接著執行步進敘述。下面這段程式會印出 1 到 10 的數：

```
for ($i=1; $i<=10; $i++){  
    print "$i ";  
}
```

一開始 `$i` 的值是 1，隨後程式會檢查它是不是比 10 小。由於結果為真，迴圈本體、（`print` 敘述）會被執行一次，之後步進敘述會讓 `$i` 的值加 1。這動作一直持續下去，直到 `$i` 等於 11 為止。

## 第四章

---

# foreach

另外一個迴圈敘述是 `foreach`。它可接受一個串列，將其中的資料一次一個設給某純量變數，並執行隨後的區段敘述。其用法如下：

```
foreach $i (@some_list) {  
    statement_1;  
    statement_2;  
    statement_3;  
}
```

注意，一旦迴圈結束，純量變數的值會自動回復到迴圈之前的狀態；delete it！對 `foreach` 迴圈來說，這就好像把它們當成區域變數 (local variable) 一樣。

下面是 `foreach` 的使用例：

```
@a = (1,2,3,4,5);  
foreach $b (reverse @a) {  
    print $b;  
}
```

上面程式的執行結果是 54321。注意 `foreach` 所用的串列可以是任何運算結果為串列的敘述，不一定要陣列變數。

甚至，你可以不寫純量變數，讓 Perl 用預設的變數，名為 “`$_`”。在 Perl 中，許多運算都會用 `$_` 做為預設的變數；任何這種地方都可以用一般的變數取代 `$_`。舉例來說，如果不指定任何值給 `print`，會印出 `$_` 的內容；所以上個程式可以改寫如下：

```
@a = (1,2,3,4,5);  
foreach (reverse @a) {  
    print;  
}
```

是否覺得這樣比較容易呢？等到你學會更多「偷用」`$_` 的敘述、算符、或函式，就能把程式寫得再簡潔一點。

如果在 `foreach` 敘述中的括號內，以真實純量變數取代串列（或會傳回串列的函式），則該變數事實上會成為其他串列變數的別名，不僅僅只有複製值而已。而且，如果你改變了純量變數，你同時也改變了該純量變數所對應的元素值。舉例說明：

```
@a = ( 3,5,7,9 ) ;
foreach $one (@a) {
    $one += 3 ;
    $x = 17 ;
    @a = (3,5,7,9) ;
    @b = (10,20,30) ;
    foreach $one ( @a, @b, $x ){
        $one *= 3 ;
    }

    # x 現在的值為 51
    # @a 現在的值為 (9,15,21,27)
    # @b 現在的值為 (30,60,90)
}
# @a = (9,15,21,27)
```

注意改變 `$one` 就會動到 `@a` 中對應的元素。這是特別的設計，不要把它當成 bug。

## 第四章

---

### 習題

1. 寫一程式要使用者輸入現在的氣溫，如果超過 72 就印出 “ too hot ”，否則印出 “ too cold ”。
2. 改良以上程式，氣溫超過 75 才印 “ too hot ”，低於 68 印 “ too cold ”，介於其間則印 “ just right! ”。
3. 不停讀入數字（每個一行）直到使用者輸入 999 為止，然後把這些數字的和印出來（注意不含 999）。舉例來說，如果你輸入 1、2、3、999，程式應該印出 6（1+2+3）。
4. 讀入字串（每個一行），並依輸入的相反順序把它們印出來，不得利用 reverse 函式。提示：<STDIN> 在陣列環境下能讀入分行的字串。
5. 印出從 0 到 32 的數字，及這些數字的平方值。你可以用一串列儲存 0 到 32 這些值，然後做計算；另外還有不需用到串列的方法。這二種方法都做做看，然後加以比較。（為了好看起見，你可以用

```
printf ("%5g %8g\n", $a, $b);
```

把 \$a 當成 5 位數、\$b 當成 8 位數印出來。）