視覺化程式語言與實習

App Inventor - 米字三角形

Brian Wang 王昱景
brian.wang.frontline@gmail.com

程式拼塊與流程控制

- 執行程式有時需依情況不同而執行不同程式碼，其依據的原則就是「判斷式」
- 程式中用來處理重複工作的功能稱為「迴圈」，迴圈分為固定執行次數迴圈及不固定執行次數迴圈
- 一般程式語言使用「陣列」來解決儲存大量同類型資料的問題，App Inventor 則以 Lists（清單）代替陣列
- Lists 可說是一群性質相同變數的集合，屬於循序性資料結構，Lists 中的資料是一個接著一個存放

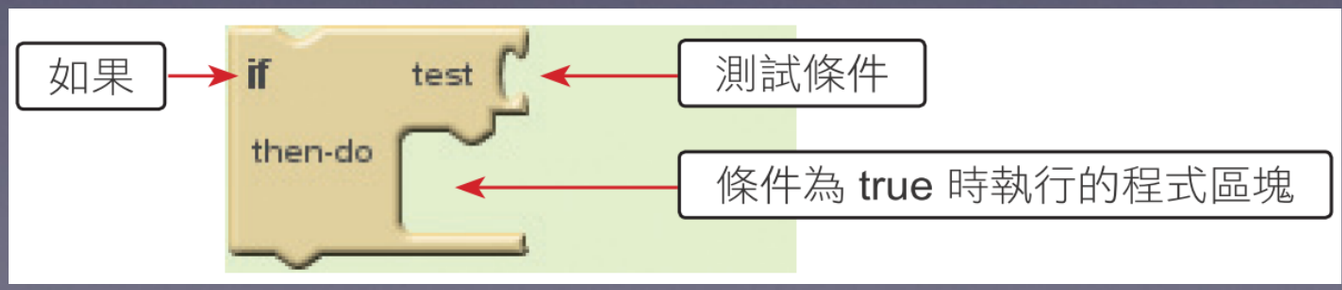
- 判斷式
- 迴圈
- Lists (清單)

判斷式

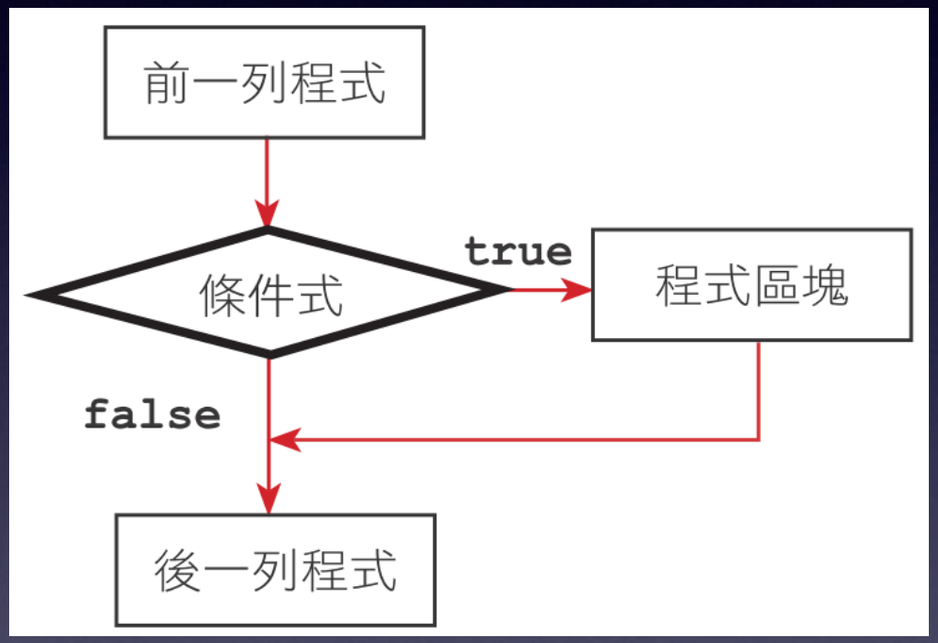
單向判斷式

- 單向判斷式是檢查指定的條件式，當條件式為 true 時，就會執行判斷式內的程式拼塊，若條件式為 false 時，就直接結束單向判斷式拼塊

- 單向判斷式拼塊位於 Built_In 的 Control 指令，拼塊的意義為「如果測試條件的結果為 true，就執行程式區塊。」



- 單向判斷式拼塊的流程圖為：

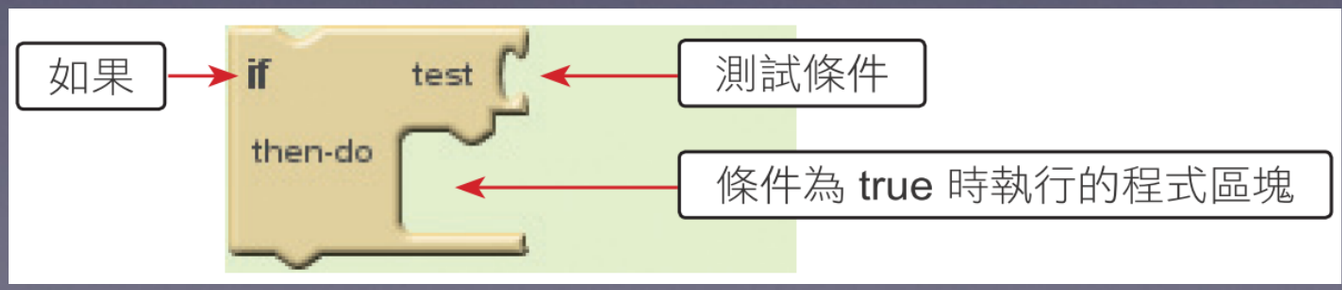


判斷式

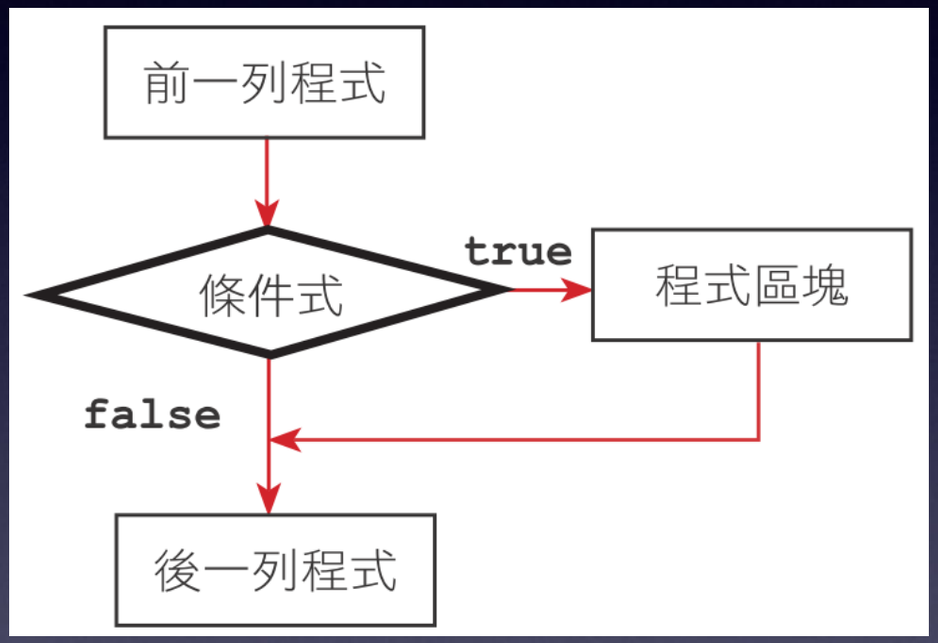
單向判斷式

- 單向判斷式是檢查指定的條件式，當條件式為 true 時，就會執行判斷式內的程式拼塊，若條件式為 false 時，就直接結束單向判斷式拼塊

- 單向判斷式拼塊位於 Built_In 的 Control 指令，拼塊的意義為「如果測試條件的結果為 true，就執行程式區塊。」

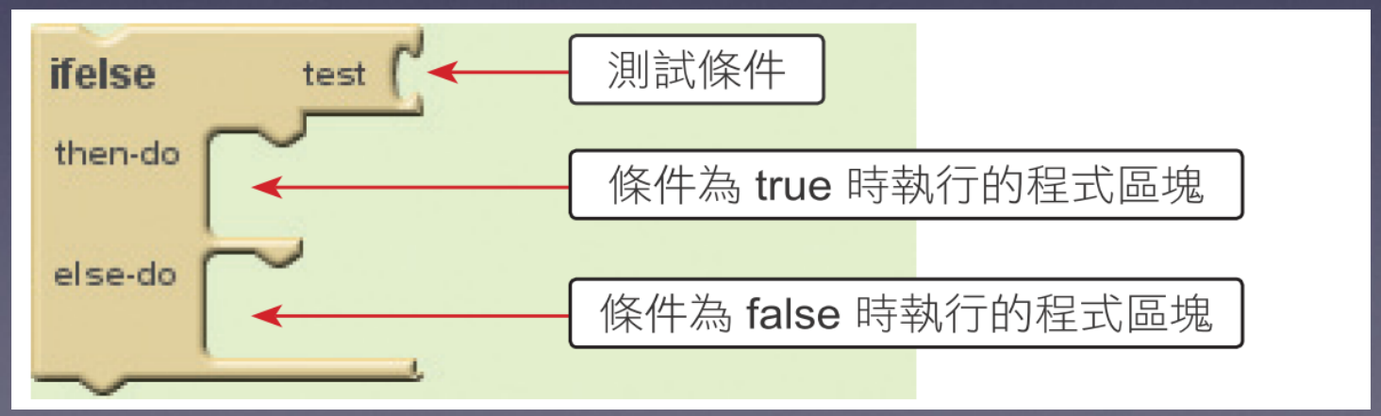


- 單向判斷式拼塊的流程圖為：

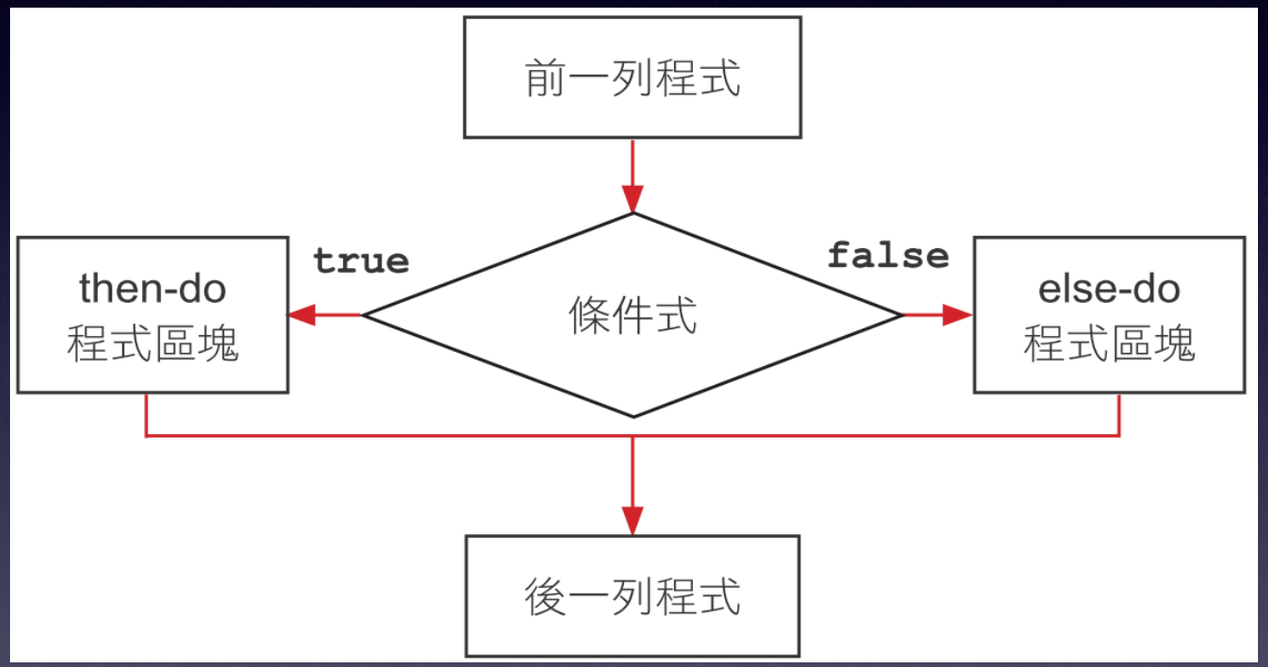


雙向判斷式

- 雙向判斷式拼塊位於 Built_In 的 Control 指令，拼塊的意義為「如果測試條件的結果為 true，就執行 then-do 的程式區塊；若測試條件的結果為 false，就執行 else-do 的程式區塊。」

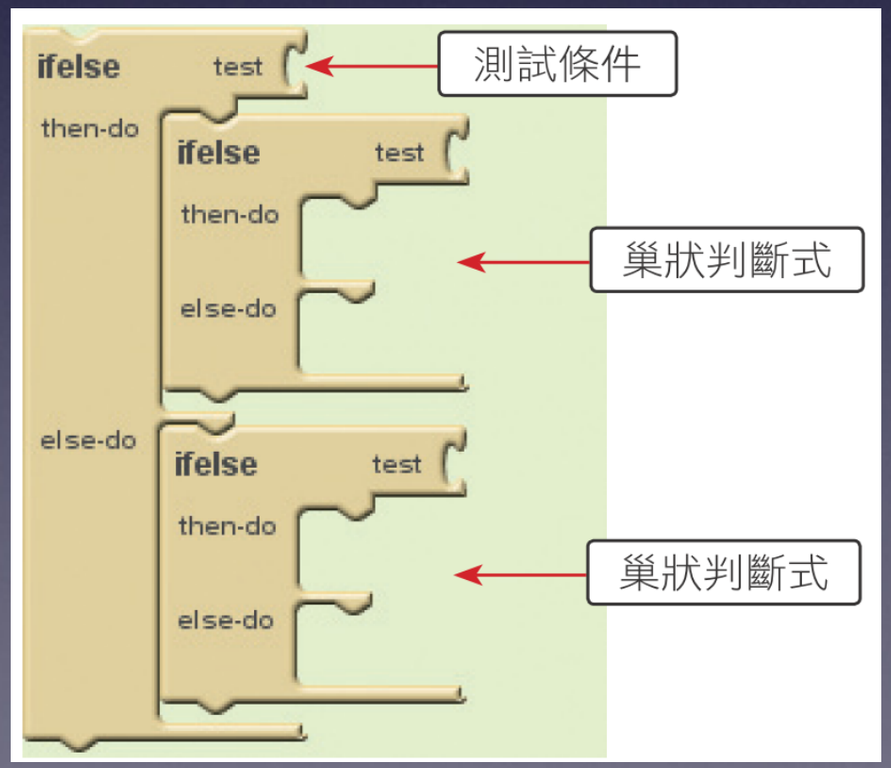


- 雙向判斷式拼塊的流程圖為：



巢狀判斷式

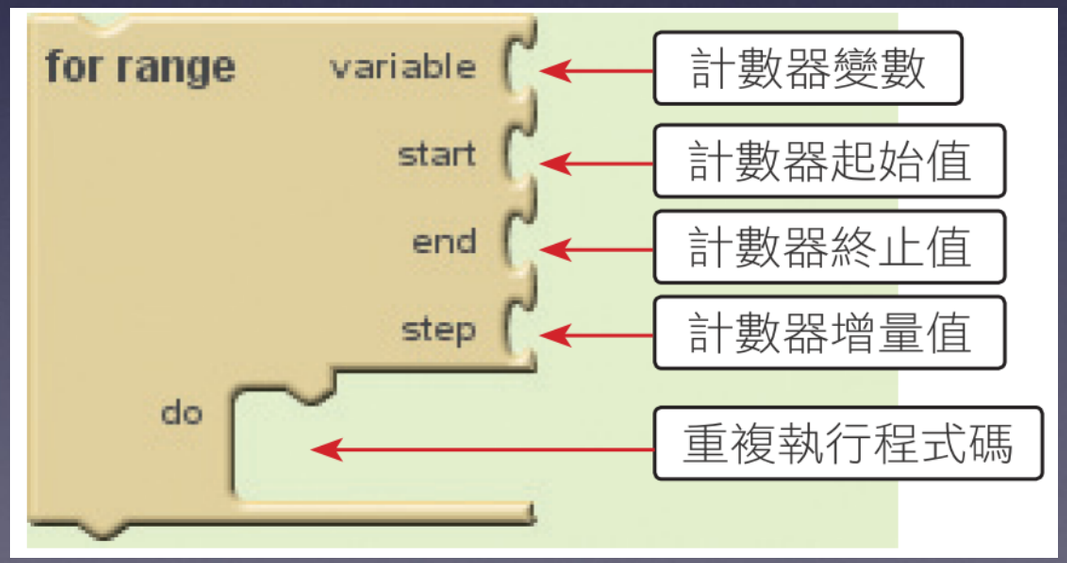
- 巢狀判斷式是在雙向判斷式的 then-do 或 else-do 程式拼塊中再加入判斷式，這樣就可進行無限多次條件判斷



廻圏

for range 迴圈

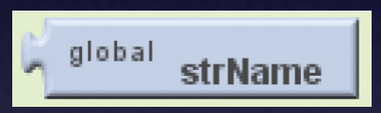
- for range 迴圈是固定執行次數的迴圈，其拼塊位於 Built-In 的 Control 指令：



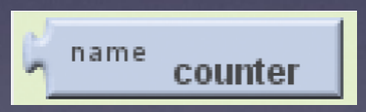
- 拼塊中 variable 是計數器變數名稱，程式中可由此變數取得計數器的數值；迴圈開始時會將計數器數值設定為初始值 (start)，接著比較計數器數值和終止值 (end)，如果計數器數值小於或等於終止值就會執行 do 區塊的程式碼
- 執行完程式區塊後會將計數器數值加上計數器增量值 (step)， 再比較計數器數值和終止值 ，如果計數器數值小於或等於終止值就會執行 do 區塊的程式碼，如此週而復始，直到計數器數值大於終止值才結束迴圈

全域變數與區域變數

- def variable as 拼塊建立的變數是「全域變數」，全域變數在整個應用程式中都可以使用，取得全域變數值的拼塊為 global 變數名稱 拼塊，例如：

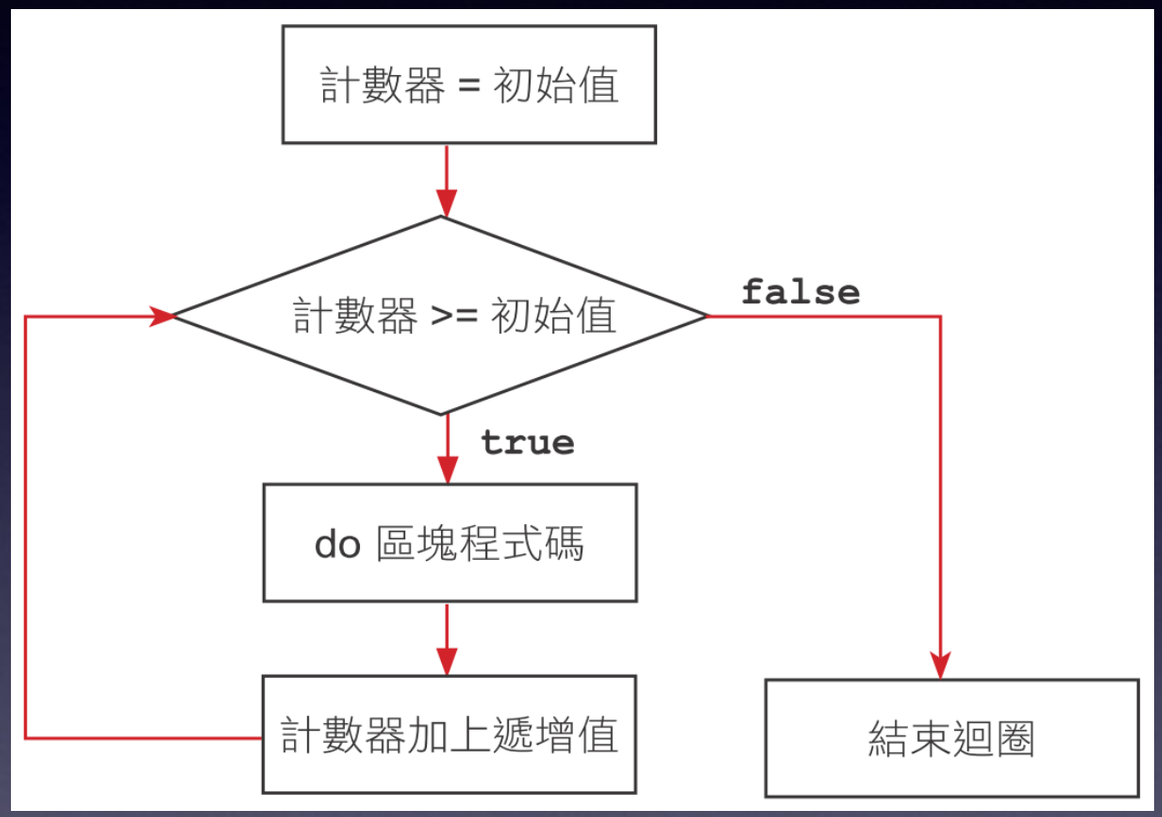


- 迴圈中使用的變數是區域變數 (不能使用全域變數)，此區域變數只能在迴圈中使用，若在迴圈外使用會產生錯誤。區域變數的宣告是使用 Built-In 的 Definition 指令的 name name 拼塊，再修改變數名稱：



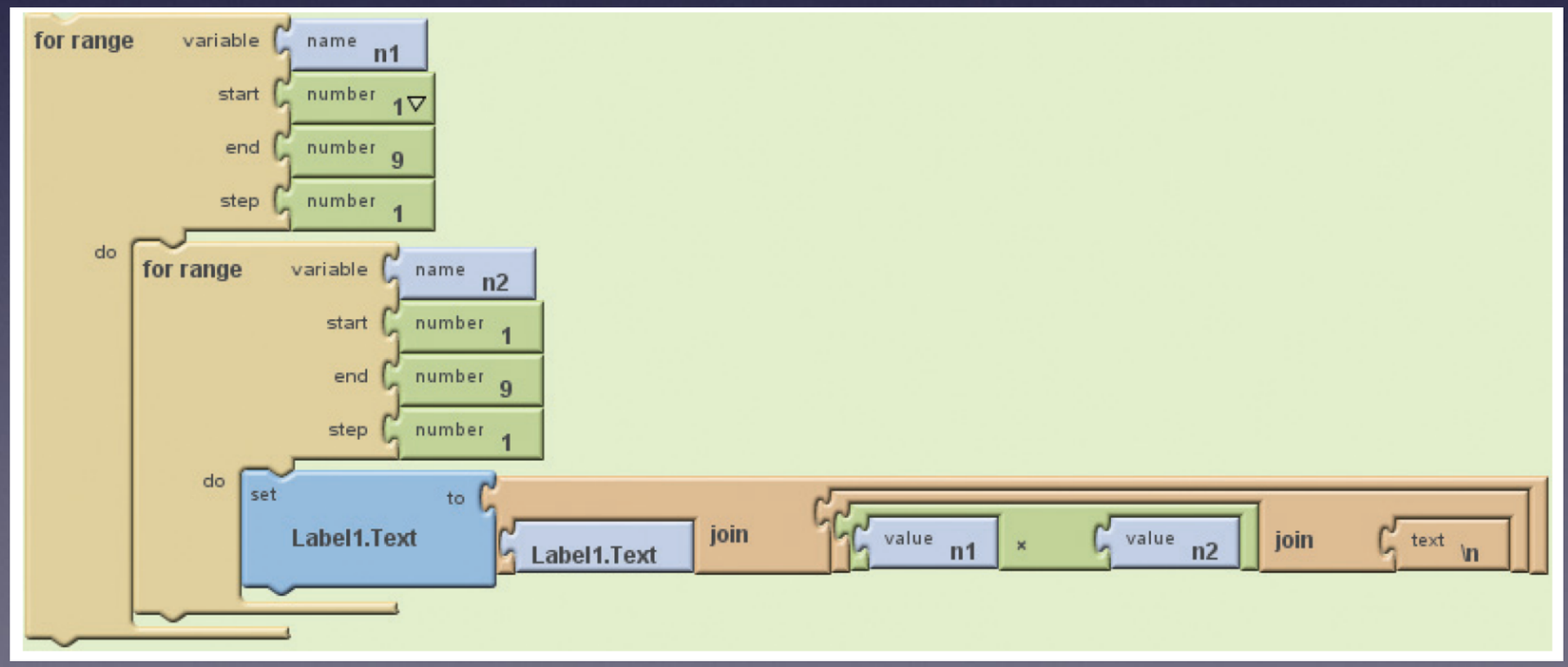
- 取得區域變數值的拼塊為 value 變數名稱 拼塊，例如：

- for range 迴圈的流程圖為：



巢狀迴圈

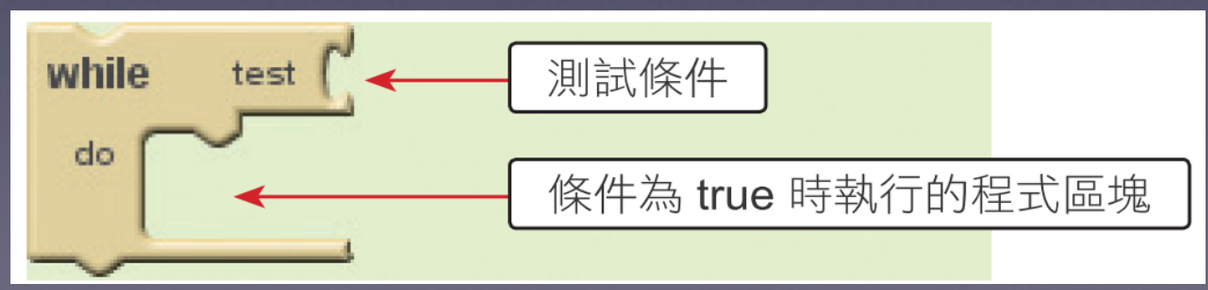
- App Inventor 允許 for range 迴圈之中包含 for range 迴圈，即「巢狀迴圈」。例如使用兩個 1 到 9 的迴圈就可顯示九九乘法表：



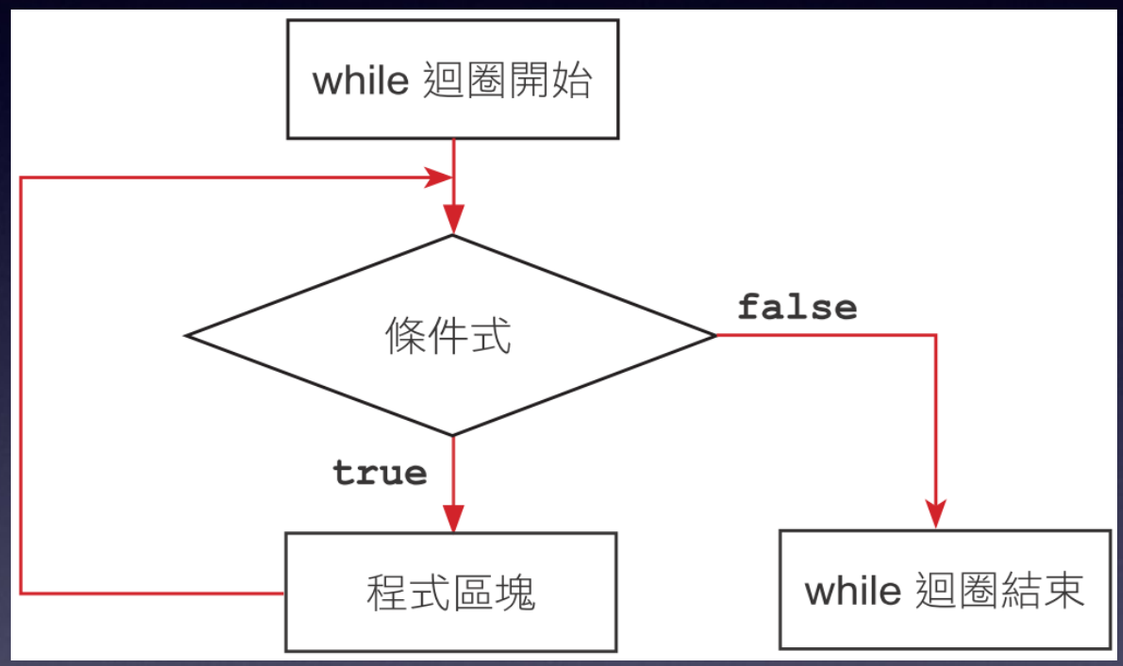
while 迴圈

- while 迴圈是不固定執行次數的迴圈，其原理是檢查條件式是否成立做為執行程式區塊的依據

- while 迴圈拼塊位於 Built-In 的 Control 指令：



- while 迴圈的流程圖為：



Lists (清單)

- 一般程式語言使用「陣列」來解決儲存大量同類型資料的問題，App Inventor 則以 Lists (清單) 代替陣列

- Lists 可說是一群性質相同變數的集合，屬於循序性資料結構，Lists 中的資料是一個接著一個存放

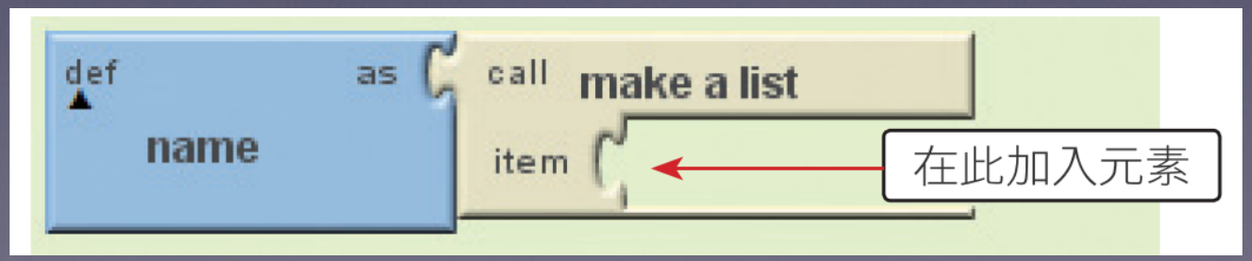
- 宣告 Lists 時需指定一個名稱，做為識別該 Lists 的標誌；Lists 中的每一個資料稱為「元素」，每一個元素相當於一個變數，因為元素是依序儲放，利用元素在 Lists 中的位置編號 (index) 就可輕易存取特定元素

- 可以把 Lists 想成是有許多相同名稱的盒子連續排列在一起，每個盒子有連續且不同的編號
- 使用者可將資料儲存在這些盒子中，如果要存取盒子中的資料時，只需知道盒子的編號即可存取盒子內的資料

宣告 Lists

- 在使用 Lists 之前需先宣告，宣告時要指定 Lists 名稱，以後要使用此名稱來存取這個 Lists，並且要設定初始值

- 宣告 Lists 的方法是先宣告一個變數，再拖曳 Built_In 的 Lists 指令 call make a list item 拼塊到變數的拼塊填入處，例如宣告一個名稱為 name 的 Lists：



取得 Lists 元素值

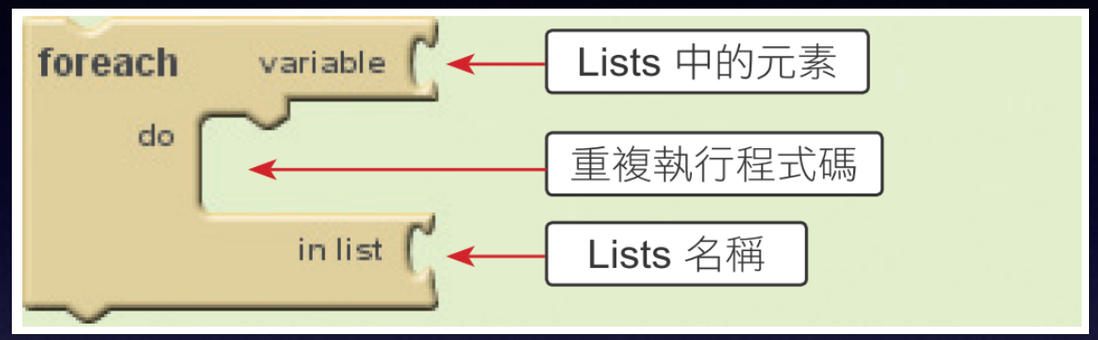
- 宣告 Lists 並設定初始值後，使用 Built_In 的 Lists 指令 call list select list item index 拼塊即可取得 Lists 中指定的元素值

- list 拼塊填入處加入要取得資料的 Lists 名稱，index 拼塊填入處加入元素的 index 值

foreach 迴圈

- foreach 迴圈的功能與 for range 迴圈類似，都是執行固定次數指定的程式區塊
- 不同處在於 for range 迴圈的執行次數是由初始值 (start) 與終止值 (end) 決定，而 foreach 迴圈則是專為 Lists 設計的迴圈，其執行次數是由 Lists 的元素個數決定，foreach 迴圈會依序對 Lists 中每一個元素執行一次程式區塊

- foreach 迴圈位於 Built-In 的 Control 指令：

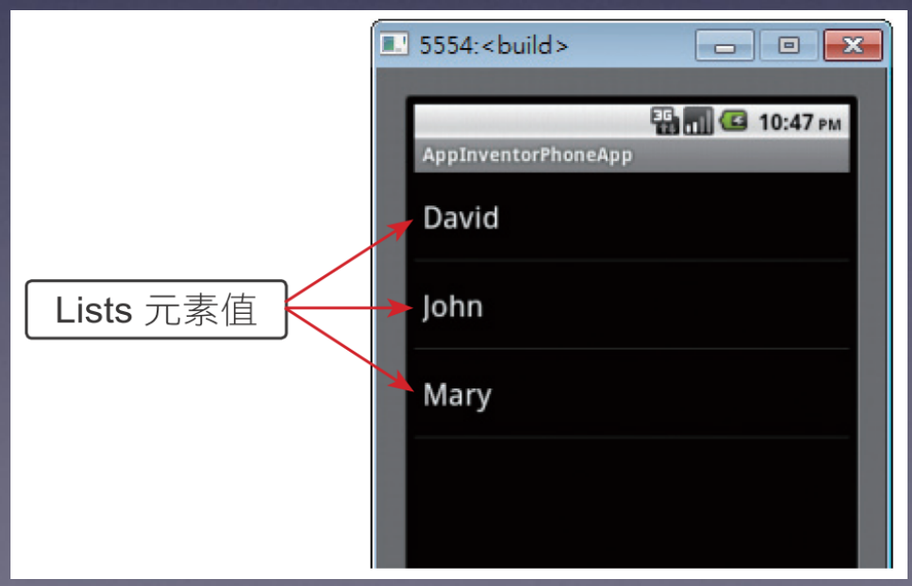


- 拼塊中 variable 是 Lists 中的元素，程式中可由此變數取得元素值，接著執行 do區塊的程式碼

- 執行完程式區塊後 variable 會取得下一個元素值，再執行 do 區塊的程式碼，如此週而復始，直到所有元素都執行程式區塊才結束迴圈

ListPicker 元件

- 以 foreach 迴圈顯示 Lists 元素值，呈現的版面外觀需自行設計，也無法讓使用者點選 Lists 中的元素
- ListPicker 元件可以方便的使用美觀的表列形式顯示 Lists 的元素值，同時可讓使用者在表列中點選，元件會傳回使用者選取的元素值



- ListPicker 元件屬於 Basic 元件，常用屬性有：

屬性	說明
ElementsFromString	設定 Lists 元素值，元素值間以逗號分隔。
Enabled	設定元件是否可用。
FontBold	設定文字是否顯示粗體。
FontItalic	設定文字是否顯示斜體。
FontSize	設定文字大小，預設值為「14」。
FontTypeface	設定文字字形。
Image	設定元件顯示的圖形。
Selection	設定選取的元素。
Shape	設定元件顯示的外觀形狀。
Text	設定顯示的文字。
TextAlignment	設定文字對齊方式 (Left：左、Center：置中、Right：右)。
TextColor	設定文字顏色。
Visible	設定是否在螢幕中顯示元件。

- ListPicker 元件常用的事件及方法有：

事件或方法	說明
AfterPicking 事件	使用者點選 ListPicker 元件的元素後觸發本事件。
BeforePicking 事件	使用者點選 ListPicker 元件後，尚未顯示 ListPicker 元件的元素值前觸發本事件。
Open 方法	顯示 ListPicker 元件的元素值。

- 當使用者點選 ListPicker 元件後，在尚未顯示 ListPicker 元件的元素值前就觸發 BeforePicking 事件， 所以常將設 定 ListPicker 元件的 Lists 來源置於 BeforePicking 事件中

- 而 AfterPicking 事件幾乎是每一個 ListPicker 元件都會使用的事件，因為使用者點選 ListPicker 元件的元素後，需靠 AfterPicking 事件做為後續處理

管理 Lists

新增 Lists 元素

- 在 Lists 增加元素有兩種情況：第一種是將元素加在 Lists 的最後面，拼塊為：

- 第二種是在 Lists 任意位置加入元素，拼塊為：

移除 Lists 元素

- 當 Lists 中元素不再使用時，可將其移除，移除 Lists 元素的拼塊為：

修改 Lists 元素值

- App Inventor 使用新值覆蓋掉舊值的方式來修改元素值，其拼塊為：

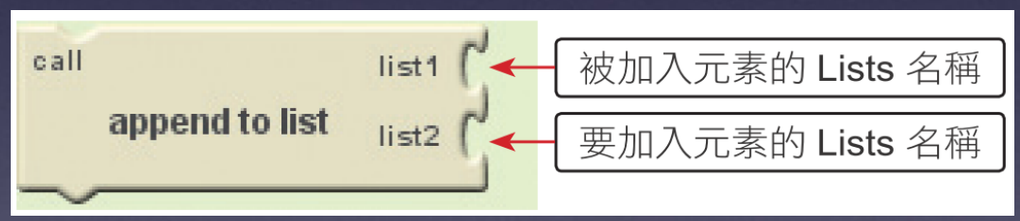
搜尋 Lists 元素

- 如果要尋找某個元素值在 Lists 中的位置，可使用 position in list 拼塊：

結合 Lists

- 有時需要將整個 Lists 中的元素全部加到另一個 Lists 中，也就是將兩個 Lists 結合為一個 Lists。

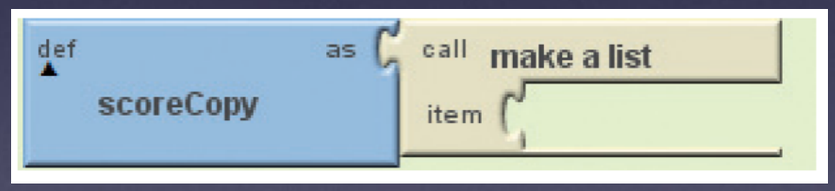
結合 Lists 的拼塊為：



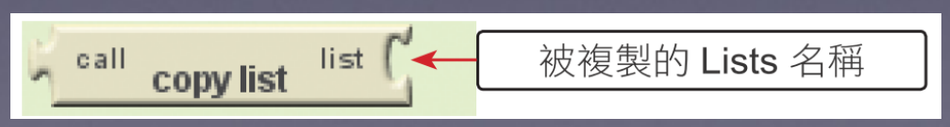
- 會將 list2 的元素加到 list1 的最後

複製 Lists

- Lists 可以複製以建立另一個完全相同的 Lists，複製之前需先宣告一個 Lists 變數，例如宣告一個名稱為 scoreCopy 的 Lists 變數：



- 複製 Lists 的拼塊為：



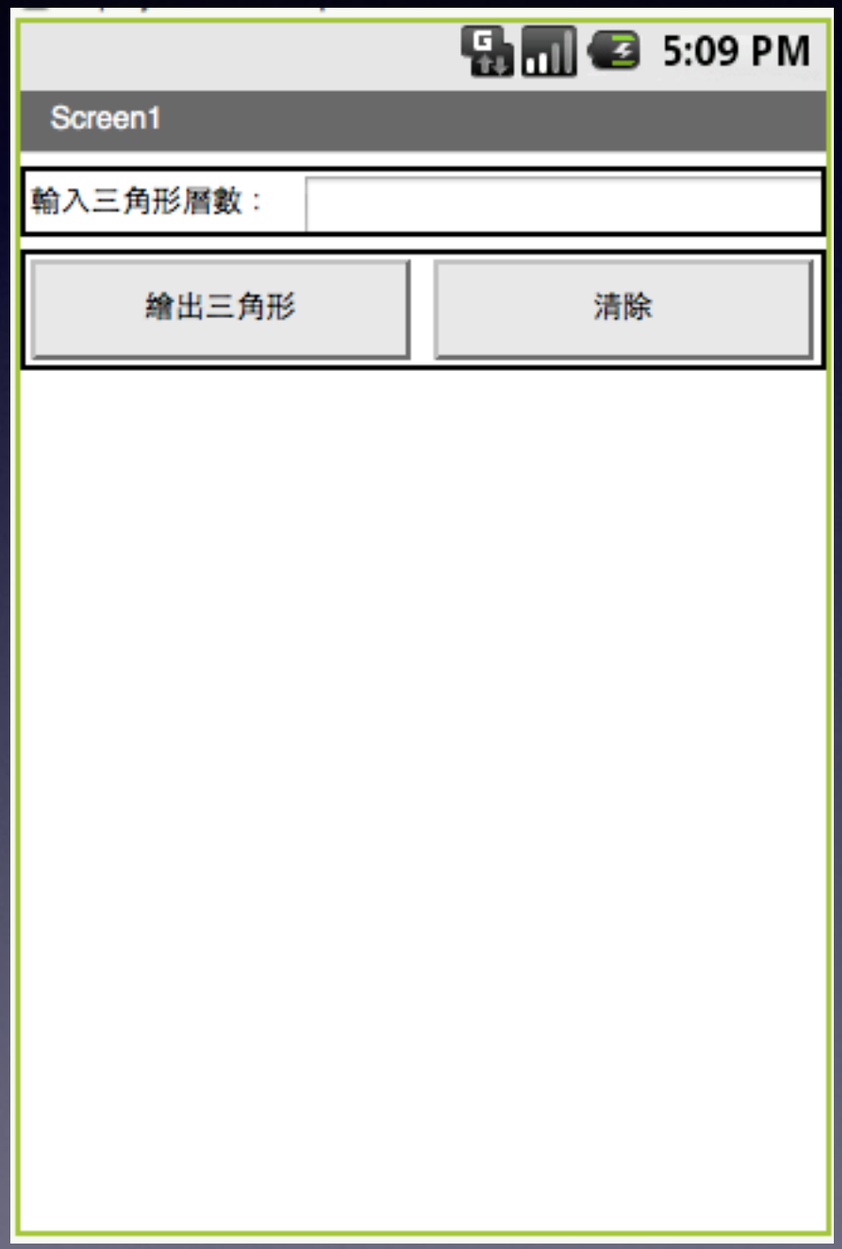
取得元素個數

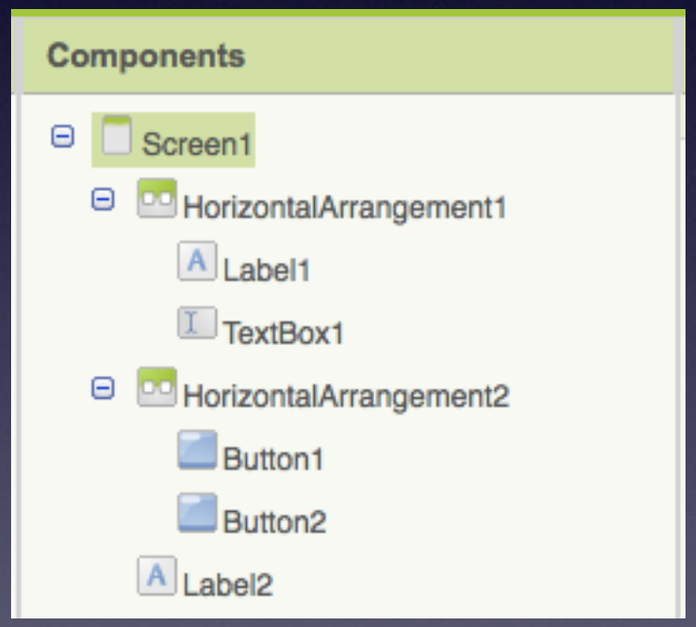
- 要得知 Lists 中有多少元素，可使用下列拼塊：

米字三角形

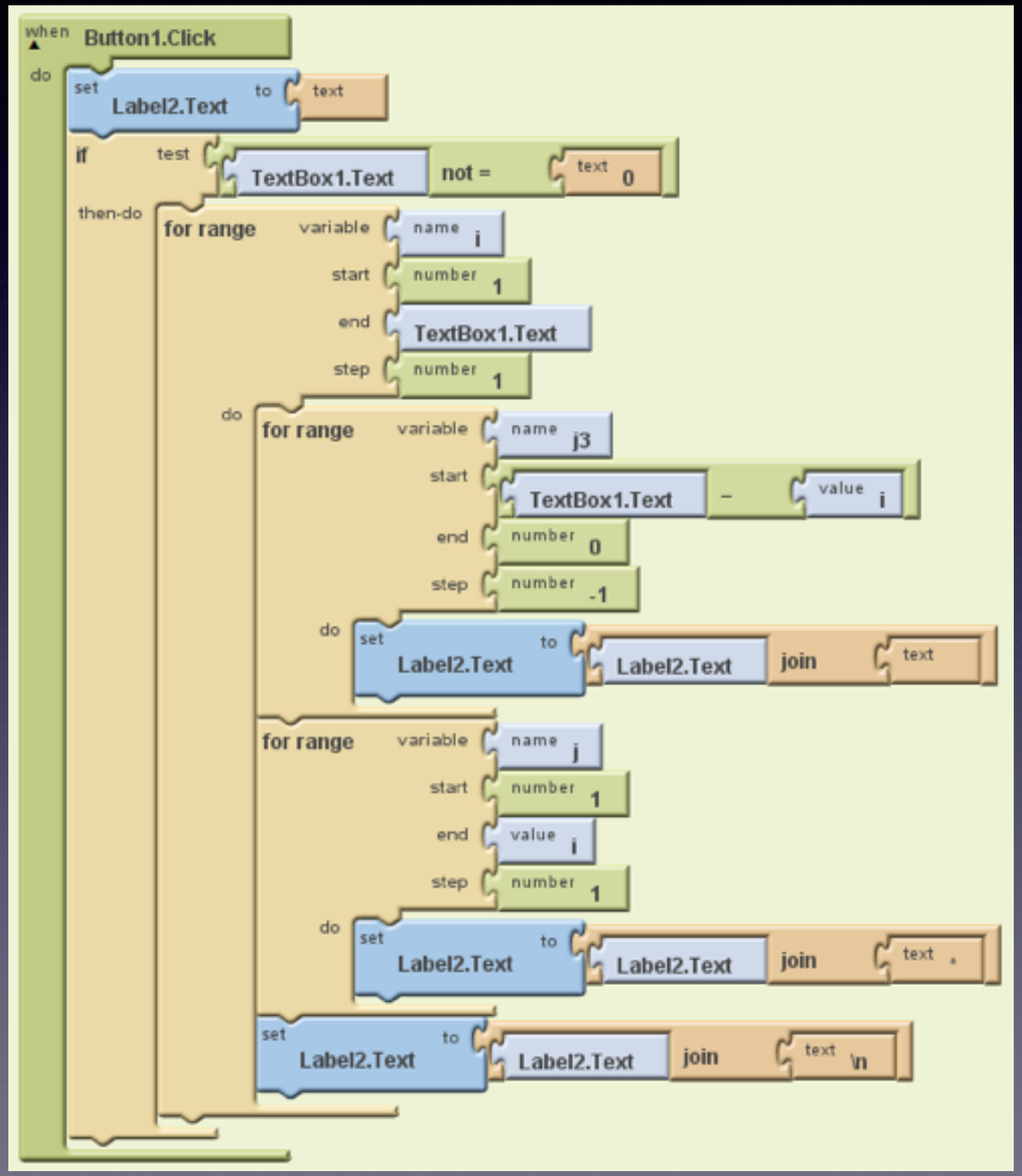
- 使用巢狀迴圈，以文字列印方式排列出文字直角三角形
- 執行時輸入三角形的層數後，按繪出三角形鈕就會以指定的層數列印米字三角形圖案

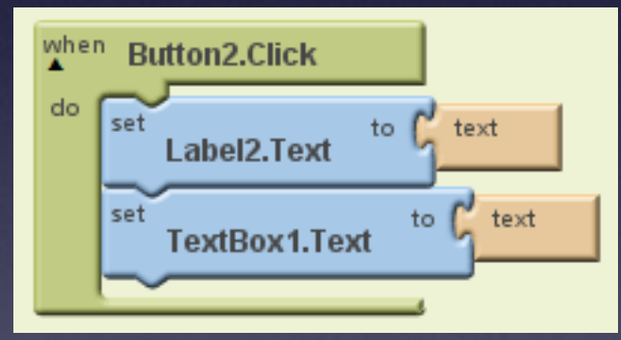
Designer


5:09 PM
Screen1
輸入三角形層數：
繪出三角形
清除


Components
Screen1
HorizontalArrangement1
Label1
TextBox1
HorizontalArrangement2
Button1
Button2
Label2

Blocks


when Button1.Click
do
set Label2.Text to text
if test TextBox1.Text not = text 0
then-do
for range variable name i
start number 1
end TextBox1.Text
step number 1
do
for range variable name j3
start TextBox1.Text – value i
end number 0
step number -1
do
set Label2.Text to Label2.Text join text
for range variable name j
start number 1
end value i
step number 1
do
set Label2.Text to Label2.Text join text
set Label2.Text to Label2.Text join text \n


when Button2.Click
do
set Label2.Text to text
set TextBox1.Text to text