

UNIT 4



pythonTM

If-else, Loop

張傑帆 Chang, Jie-Fan



OUTLINE

- 判斷
 - 單向判斷 (if...)
 - 雙向判斷 (if...else)
 - 多向判斷 (if...elif...else)
 - 巢狀判斷
- 迴圈
 - for...in 迴圈
 - while 迴圈
 - 巢狀迴圈
 - 強制結束 : break
 - 強制回頭 : continue



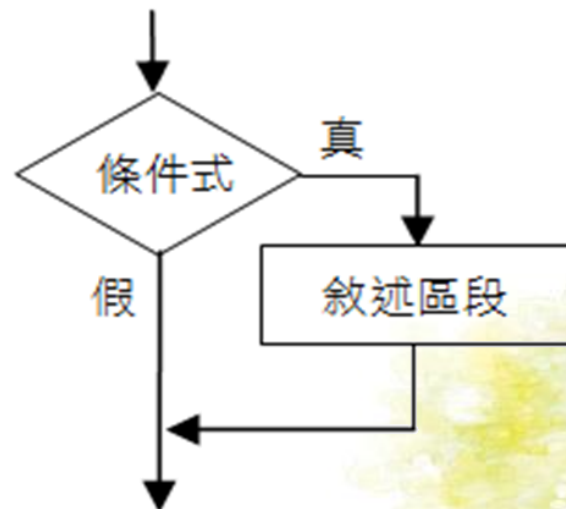
判斷

- **Python** 是依照甚麼樣的方式在執行程式碼
- 在沒有特殊情形之下，會從程式碼的第一行開始執行，**依序向下**
- **特殊情形**有以下四種：
 - 判斷式
 - 迴圈
 - 函數
 - 例外



判斷式

- if 判斷式
- 當程式執行到if 判斷式時
- 判斷現在的情形是否符合撰寫所設定的條件
- 所以可能會因條件不符合，而發生跳過不執行某些區塊程式碼的情形

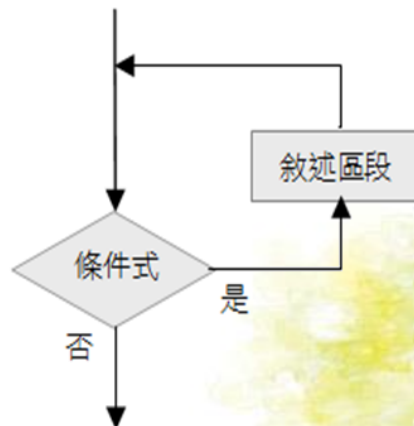


判斷

■ 迴圈：

迴圈內的程式碼是可能要重複執行的程式碼，有for迴圈跟while迴圈兩種類型

■ 當程式碼執行到迴圈時，會根據程式撰寫時所設定的條件作判斷，若符合則重複執行迴圈裡面的程式碼



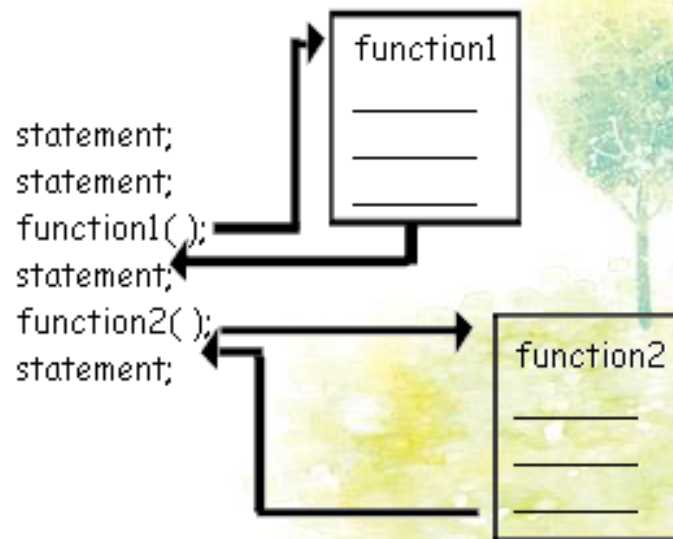
判斷

■ 函數：

函數是為了完成特定功能而被撰寫的程式碼，有可以被重複呼叫的特性。

當程式執行時遇到呼叫函數，會去尋找此函數的完整程式碼，然後利用程式呼叫函數時所傳入的參數，進而執行該函數的程式碼。

■ 在這種情形下，通常會產生跳躍式執行程式碼的可能性。



判斷

- 例外：
- 這裡所說的例外就是錯誤，例如縮排不一致、格式發生衝突等等。
- 出現這種情形如果沒有妥善處理，程式的執行就會強制中斷。



判斷

- 以上四種特殊情形會改變從上至下逐行檢查並執行的動作
- 可以完成更複雜的程式結構與功能，讓所撰寫出來的程式能完成更複雜的動作
 - 判斷式
 - 迴圈
 - 函數
 - 例外



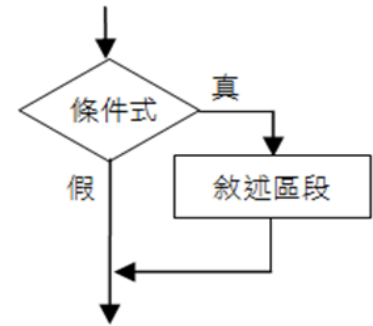
單向判斷 (IF...)

- **if** 判斷式是程式語言中最基本的判斷式
- 只要是一支功能完整的程式就少不了**if** 判斷式
- 在這裡先以單層**if** 來做講解：

if 判斷條件：

縮排

→ 執行的程式碼...



- 在**if** 後面加上條件判斷式，後面加上「**:**」符號
- 下一行開始一定要**縮排**，直到在此條件下欲執行的程式碼結束為止。
- **Python** 是以縮排來區分程式碼的區塊，如果縮排沒有對齊，程式碼在執行上會出現錯誤！



縮排

- Python 在程式區塊中**強制使用縮排**，而不使用{}
- 令程式碼更簡潔，增加可讀性
- 在 Java/C/C++ 中縮排可做可不做
- 但在Python中，**一定要**做縮排
- 如果縮排**沒有對齊**，那程式碼是無法執行的
Python 會跟你說出現預期以外的縮排行為

hello3.py

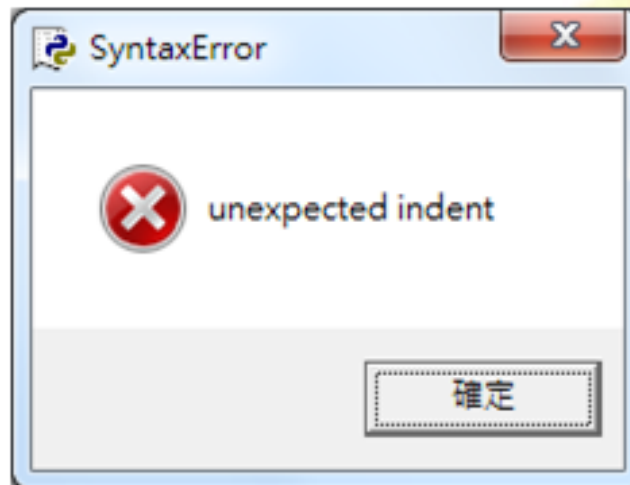
```
1  # Prints a helpful message.
2  def hello():
3      ➡ print("Hello, world!")
4      ➡ print("How are you?")
5
6  # main (calls hello twice)
7  hello()
8  hello()
```



縮排

■ 示範程式碼

```
01 print('Hello, World') # 預設的print 方式
02 print('Hello, World', end=' ')
    # 將預設的結尾換行符號替換成半型空白，
03    # 下一次輸出便會接在這一次輸出之後
04 → print('Hello, ', 'World', sep='@')
    # 用@ 替換半型空白當作隔開項目之間的字元
```



縮排

- 還有，在Python 裡面，Tab 鍵也可以拿來縮排
- 就算看起來縮排一致了，但是空白鍵的縮排與Tab 鍵的縮排計算方式是不一樣的，所以
- 縮排時儘量不要混用空白鍵和Tab 鍵！



單向判斷 (IF...)

- 從流程圖可以清楚了解單向if 判斷式在執行時會是怎樣的情形。以下用實例進一步說明：
- 示範程式碼

```
01 age = input(' 請輸入您的年齡\n')
02
03 if(int(age) >= 20):
04     print(" 您已經成年了，可以投票了喔")
```

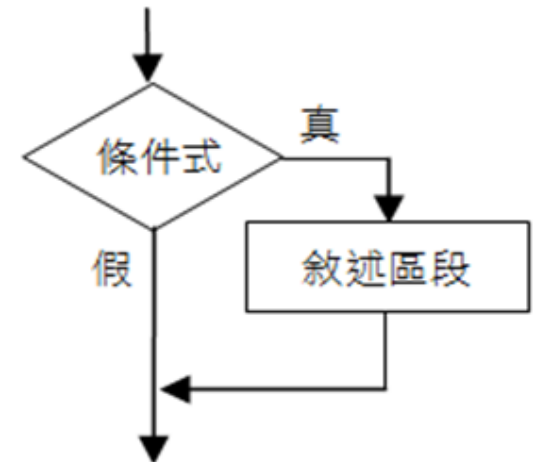
>>>

請輸入您的年齡

23

您已經成年了，可以投票了喔

>>>



單向判斷 (IF...)

■ 示範程式碼

```
01 if True:
```

```
02     print('這行程式碼一定會被執行')
```

■ 執行結果

```
>>>
```

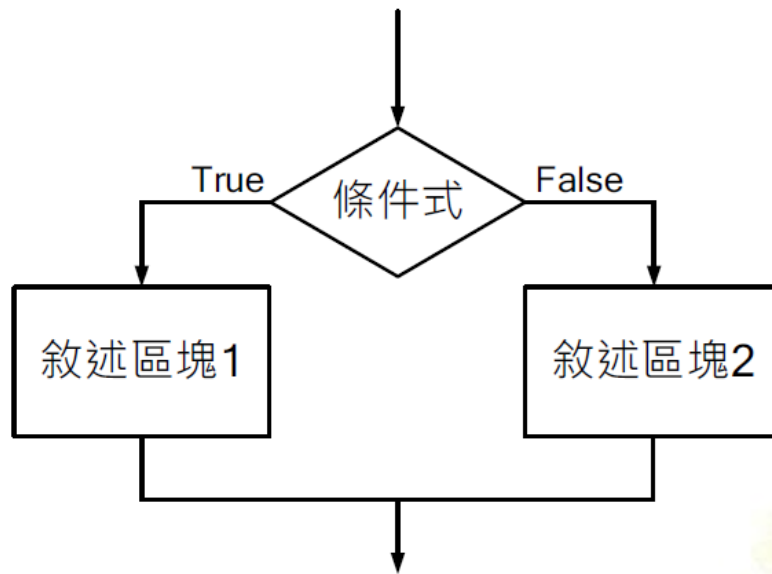
```
這行程式碼一定會被執行
```

```
>>>
```



雙向判斷 (IF...ELSE)

- 在日常生活領域中，常出現「假如～則～，否則～」
- 此種決策模式有兩種解決問題的方案，故稱為雙向判斷
- 這種時候就使用 **else** 來呈現沒有達成條件的情形
- 一定要記得在 **else** 後面也要加上「:」符號。



if 判斷條件:

執行的程式碼1...

else:

執行的程式碼2...



雙向判斷 (IF...ELSE)

■ 示範程式碼

```
01 age = input(' 請輸入您的年齡\n')
02
03 if (int(age) >= 20):                # 當輸入年齡大於等於18
04     print(" 您已經成年了，可以投票了喔")
05 else:                              # 當輸入年齡小於18
06     print(" 雖然您還未成年，但是可以關心政治喔")
```

```
>>>
```

```
請輸入您的年齡
```

```
15
```

```
雖然您還未成年，但是可以關心政治喔
```

```
>>>
```



小練習

- 令使用者輸入成績
- 並判斷其成績是否及格

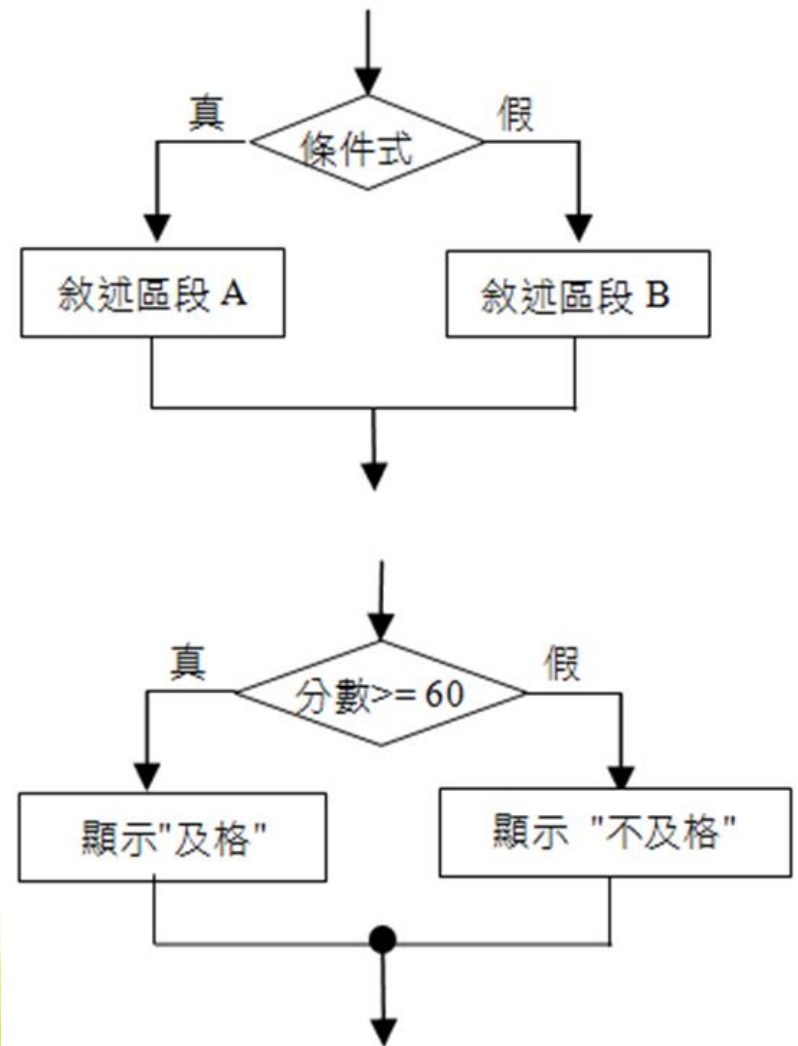
請輸入成績：59
不及格

>>> =====

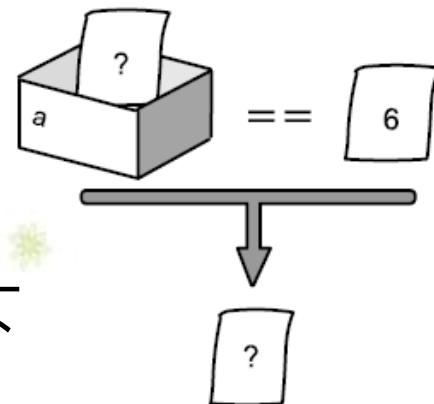
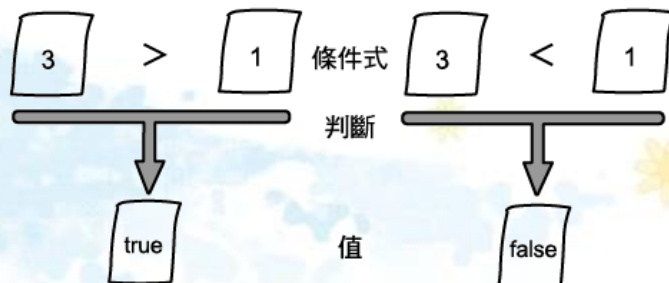
>>>

請輸入成績：60
及格

>>>



IF-ELSE



- 邏輯判斷可以使用的運算符號如下

運算符號	意義
>	大於
<	小於
>=	大於或等於
<=	小於或等於
==	等於 ※ 和「=」(指定運算子) 不同!
!=	不等於

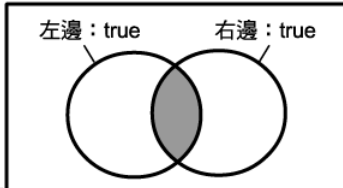
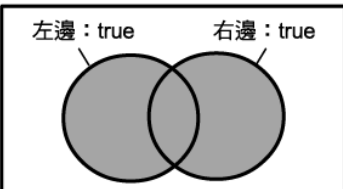

- 用來連結邏輯判斷的運算子有

運算子	意義
and	" 而 且", 所有的條件都要成立, 整個判斷才會成立
or	" 或 ", 只要有任何一個條件成立, 整個判斷就會成立
not	" 非 ", 條件不成立時, 整個判斷就會成立



運算子	說明	使用例	結果
== (相等)	判斷此運算子左右兩邊運算式的值是否相等。	18 == 18	1(真)
		18 == 35	0(假)
		3+2 == 1+4	1(真)
!= (不相等)	判斷此運算子左右兩邊運算式的值是否不相等。	17 != 18	1(真)
		56 != 56	0(假)
		12*3 != 3*12	0(假)
< (小於)	判斷此運算子左邊運算式的值是否小於右邊運算式的值。	17 < 18	1(真)
		42 < 30	0(假)
		2 < 10-7	1(真)
> (大於)	判斷此運算子左邊運算式的值是否大於右邊運算式的值。	19 > 18	1(真)
		26 > 36	0(假)
		12*3 > 12*2	1(真)
<= (小於等於)	判斷此運算子左邊運算式的值是否小於等於右邊運算式的值。	17 <= 18	1(真)
		18 <= 18	1(真)
		10+3 <= 12	0(假)
>= (大於等於)	判斷此運算子左邊運算式的值是否大於等於右邊運算式的值。	17 >= 18	0(假)
		18 >= 18	1(真)
		12*3 >= 35	1(真)



邏輯運算子	說明	真值表															
(AND, 且)	<p>此運算子左右兩邊的運算式結果若不為零值，結果為1(真)；否則為零值(假)。</p> 	<table><tr><th>運算式 1</th><th>運算式 2</th><th>結果</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	運算式 1	運算式 2	結果	1	1	1	1	0	0	0	1	0	0	0	0
運算式 1	運算式 2	結果															
1	1	1															
1	0	0															
0	1	0															
0	0	0															
(OR, 或)	<p>此運算子左右兩邊的運算式結果只要其中有一個不為零值，結果就是1；兩個都為零結果才是零值</p> 	<table><tr><th>運算式 1</th><th>運算式 2</th><th>結果</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	運算式 1	運算式 2	結果	1	1	1	1	0	1	0	1	1	0	0	0
運算式 1	運算式 2	結果															
1	1	1															
1	0	1															
0	1	1															
0	0	0															
(NOT)	<p>此運算子是單一的運算，主要是將敘述結果相反，即 $1 \Rightarrow 0$，$0 \Rightarrow 1$。</p> 	<table><tr><th>運算式</th><th>結果</th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	運算式	結果	1	0	0	1									
運算式	結果																
1	0																
0	1																



邏輯運算子使用範例

- $5 > 3$ and $3 == 4$

- (結果為false)

- $a == 6$ or $a \geq 12$

- 如果變數a的值等於6

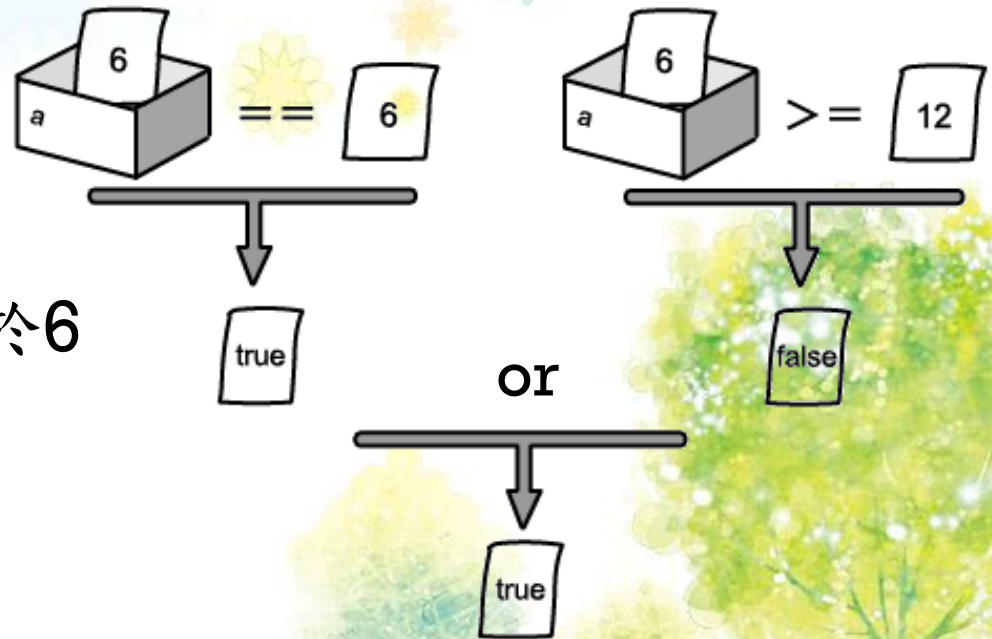
- 或是大於等於12

- 結果就會是true

- $\text{not } (a == 6)$

- 當變數a等於6以外的其他值時

- 結果就會是true



邏輯運算子

- 邏輯運算子又稱為布林運算子
- 顧名思義就是跟邏輯有關
- 其更進一步所代表意義如下所示

運算子	例子	意義
and	a and b	若a為假則回傳a；若a為真則回傳b
or	a or b	若a為假則回傳b；若a為真則回傳a
not	not a	若a為假則回傳True；若a為真則回傳False



邏輯運算子

■ 示範程式碼

01	a, b = 0, (1, 2)	# 給予a, b 不同的值
02	print(a or b)	# 印出a or b 的結果
03	print(a and b)	# 印出a and b 的結果
04	print(not a)	# 印出not a 的結果
05	print(not a and b)	# 印出not a and b 的結果

■ 執行結果

```
>>>  
(1, 2)  
0  
True  
(1, 2)  
>>>
```



邏輯運算子

■ 邏輯運算條件式的真值表

條件式 運算子	a, b皆為真	a為假b為真	a為真b為假	a, b皆為假
a and b	b	a	b	a
a or b	a	b	a	b
not a	False	True	False	True
not b	False	False	True	True



多向判斷 (IF...ELIF...ELSE)

- 在實際使用上，常出現不只一種條件需要做判斷的情形
- 使用 **elif** 來解決，是「**else if**」的縮寫，可以有許多個

if 判斷條件1:

執行的程式碼1...

elif 判斷條件2:

執行的程式碼2...

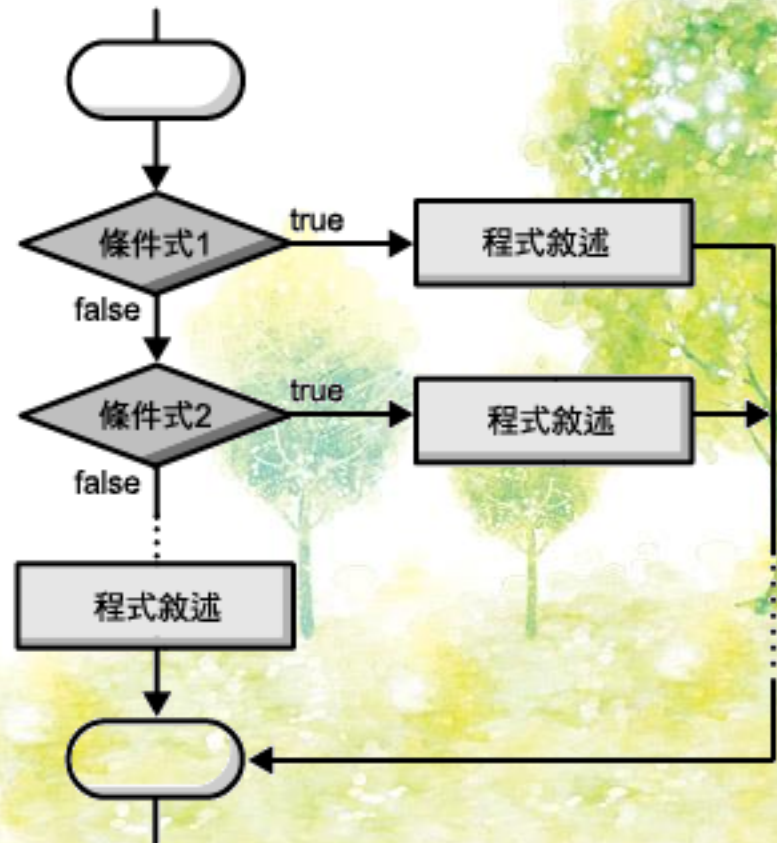
elif 判斷條件3:

執行的程式碼3...

...

else:

執行的程式碼N...



多向判斷 (IF...ELIF...ELSE)

■ 示範程式碼

```
01 age = input(' 請輸入您的年齡\n')
02
03 if int(age) >= 20:                # 當輸入年齡大於等於20
04     print(" 您在民法及刑法上都已成年")
05 elif int(age) < 20 and int(age) >= 18:
06                                     # 當輸入年齡小於20 且大於等於18
07     print(" 您在刑法上已成年，在民法上則還是未成年")
08 else:                            # 當輸入年齡小於18
09     print(" 您在民法及刑法上都還未成年")
```



多向判斷 (IF...ELIF...ELSE)

■ 執行結果

>>>

請輸入您的年齡

18

您在刑法上已成年，在民法上則還是未成年

>>> ===== RESTART =====

>>>

請輸入您的年齡

20

您在民法及刑法上都已成年

>>> ===== RESTART =====

>>>

請輸入您的年齡

19

您在刑法上已成年，在民法上則還是未成年

NT >>>



小練習

■ BMI計算機

- 寫一程式，依所輸入之**身高**、**體重**計算BMI
- BMI值計算公式： $BMI = \text{體重(公斤)} / \text{身高}^2 (\text{公尺}^2)$
- 例如：

一個52公斤的人，身高是155公分，則BMI為：

$$52(\text{公斤}) / 1.55^2 (\text{公尺}^2) = 21.64$$

並判斷其範圍：

Underweight 過輕： $BMI < 18.5$

Normal 正常： $18.5 \leq BMI < 24$

Overweight 過重： $24 \leq BMI < 27$

Obese Class I (Moderately obese) 輕度xx： $27 \leq BMI < 30$

Obese Class II (Severely obese) 中度xx： $30 \leq BMI < 35$

Obese Class III (Very severely obese) 重度xx： $BMI \geq 35$

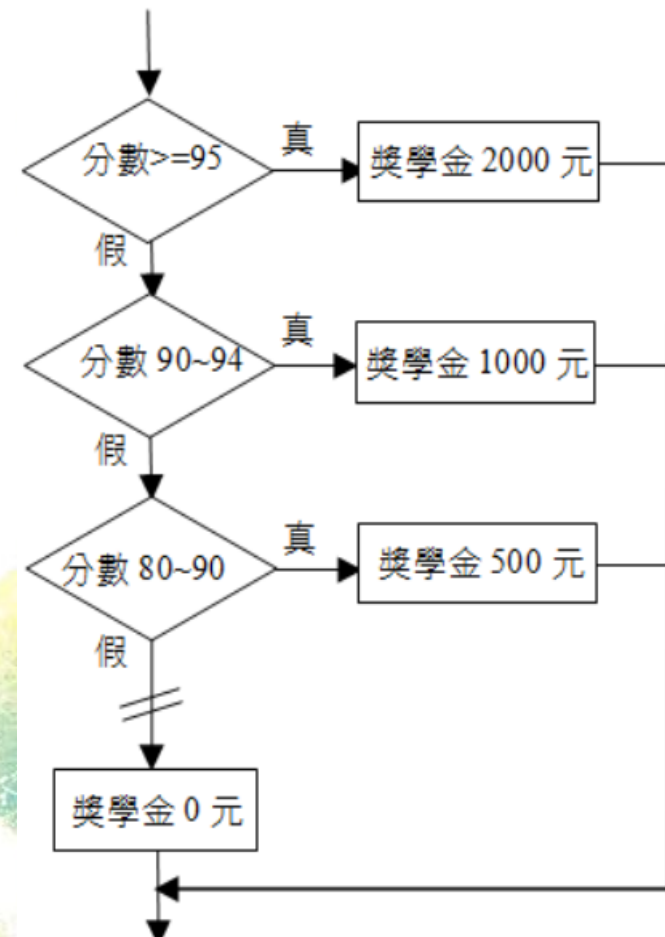
```
>>>
155
52
21.64
Normal
```



小練習

- 將獎學金額判斷程式
- 若你的分數達95分以上
 - 得獎金 2000 元
- 若是90分以上到94分
 - 得獎金 1000 元
- 若是80分以上到89分
 - 得獎金 500 元
- 若不到80分則無獎金
 - 得獎金 0 元
- Hint: 可用 **and** 連接或範圍關係運算子如：
 - $5 < \text{integer} \leq 10$

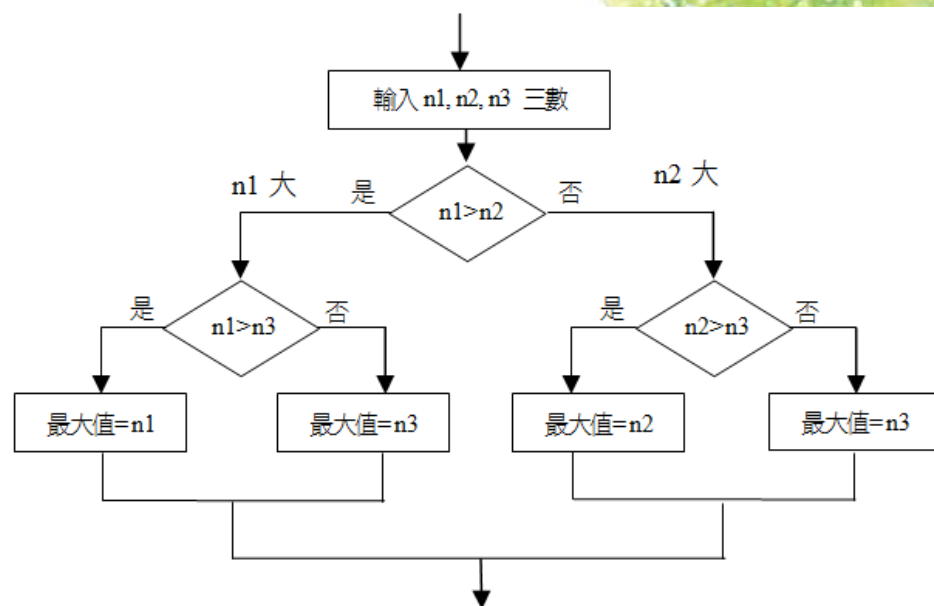
```
請輸入成績：95
獎金 2000 元
>>> =====
>>>
請輸入成績：90
獎金 1000 元
>>> =====
>>>
請輸入成績：80
獎金 500 元
>>> =====
>>>
請輸入成績：79
獎金 0 元
>>> |
```



巢狀判斷

- 有時只有一個條件是沒辦法完成精確的判斷
- 巢狀判斷就是進行一次條件判斷結束後再進行另一次條件判斷，而這兩個條件是有關聯的

```
if 判斷條件1:  
    if 判斷條件:  
        執行的程式碼1a...  
    else:  
        執行的程式碼1b...  
else:  
    if 判斷條件:  
        執行的程式碼2a...  
    else:  
        執行的程式碼2b...
```



巢狀判斷

- 比賽的參賽資格要年滿十歲，並且限制已經參賽四次的選手不得參加第五次
- 示範程式碼：

```
01 age = input(" 請輸入您的年齡\n")
02
03 if int(age) >= 10:
04     times = input(" 請問這是第幾次參加此大賽\n")
05     if int(times) < 5:
06         print(" 恭喜您，您符合本大賽的參賽資格。")
07     else:
08         print(" 很抱歉，您參加太多次了，請把機會留給別人。")
09 else:
10     print(" 很抱歉，您的年齡不符合大賽的規定，請過幾年再來。")
```



■ 執行結果

>>>

請輸入您的年齡

9

很抱歉，您的年齡不符合大賽的規定，請過幾年再來。

>>> ===== RESTART =====

>>>

請輸入您的年齡

12

請問這是第幾次參加此大賽

3

恭喜您，您符合本大賽的參賽資格。

>>> ===== RESTART =====

>>>

請輸入您的年齡

15

請問這是第幾次參加此大賽

6

很抱歉，您參加太多次了，請把機會留給別人。

>>>



小練習

- 請輸入3個數字並找到其中最大的數

輸入數字1: 99

輸入數字2: 5

輸入數字3: 3

最大的數字為: 99

>>> =====

>>>

輸入數字1: 5

輸入數字2: 99

輸入數字3: 65

最大的數字為: 99

>>> =====

>>>

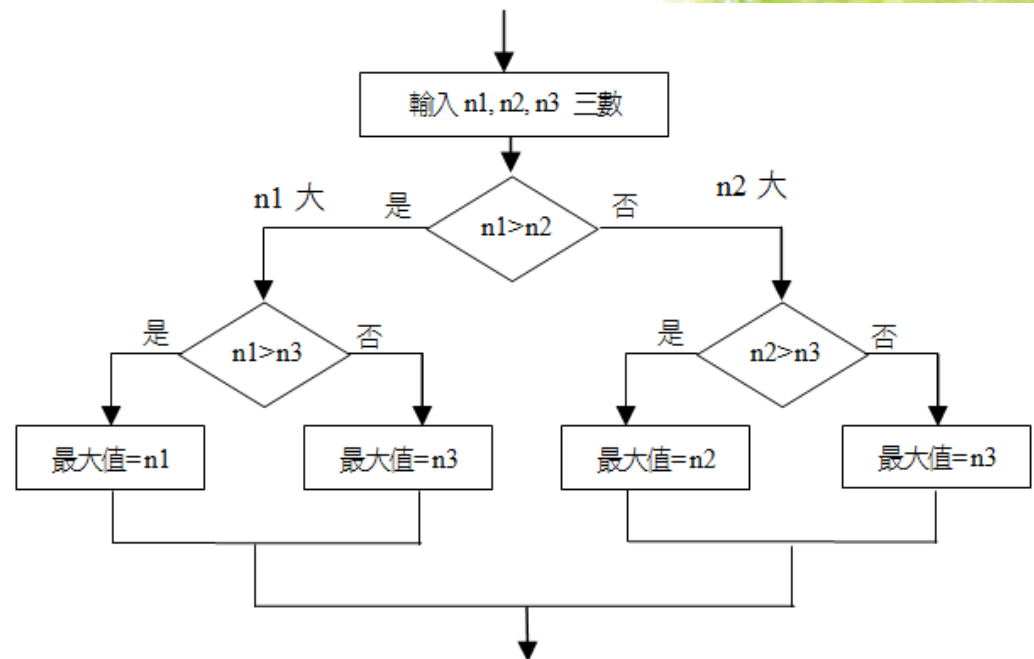
輸入數字1: 45

輸入數字2: 89

輸入數字3: 99

最大的數字為: 99

>>>



IF ELSE 縮排的恩怨情仇

- **if else** 用 **縮排** 表示其程式碼範圍
- 請務必記得改成多行時要加上 **縮排** 不然....
- 就會變成爆肝工程師！

```
n = eval(input("Please enter date (1~7):"))  
  
if (n==6 or n==7):  
    print("Yes, holiday")  
else: # 1-5  
    print("work day")  
print("Go to work.")
```

```
Please enter date (1~7):1  
work day  
Go to work.  
>>>
```



IMPORT

- 除了內建功能外，Python 的標準模組庫 (standard library) 還有許多已經定義好，並且測試無誤的模組(module)
- 模組就是已經寫好的 Python 程式檔案
我們在需要的時候使用關鍵字 (keyword) **import** 到我們自己的程式中就可以使用相關定義，同樣的，使用標準模組庫中的模組也要 **import**



亂數 RANDOM NUMBERS

`import random`

`random.randint(min, max)`

- 回傳在範圍[min, max] 中的一亂數

`choice(sequence)`

- 回傳一容器中隨機選擇的數值
- (此容器可為range, string, list, ...)

```
>>> import random
>>> random.randint(1, 5)
2
>>> random.randint(1, 5)
5
>>> random.choice(range(4, 20, 2))
16
>>> random.choice("hello")
'e'
```



回家作業

- 猜數字遊戲
- 讓系統隨機生成一數字介於1到5之間
- 使用者輸入一數字猜此數字是什麼
- 猜對的話輸出"猜對了"
- 猜錯則反之輸出"猜錯了"

>>>

3

請猜一1到5的號碼: 3

你猜對了! 答案正是 3

>>> =====

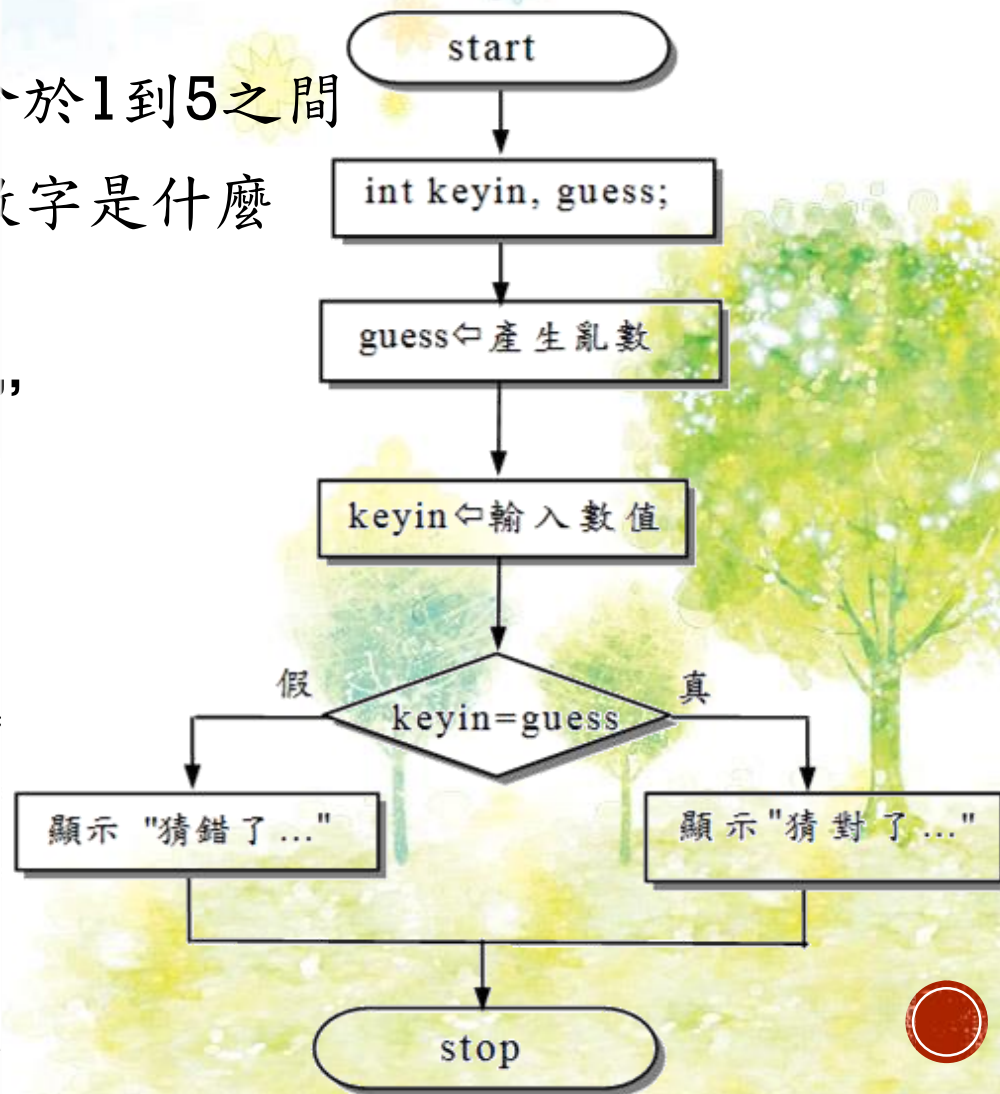
>>>

5

請猜一1到5的號碼: 4

你猜錯了喔~其實是 5

>>> |



印出10行HI PYTHON

■ ?

```
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");  
print("Hi, Python");
```

■ 那100行呢？



重複結構簡介

- Python允許將需要連續重複執行的敘述區段改用「**迴圈敘述**」，不但可**縮短**程式的長度，而且程式**易維護**及**增加**程式**可讀性**
- 此種程式架構稱為「**重複結構**」或「**迴圈**」(Loop)
- Python提供二種迴圈敘述：**for**、**while**敘述
- 若迴圈的**次數可以預知**，**for**敘述是最好的選擇
- 若迴圈**次數無法確定**，則可使用**while**敘述來達成



FOR...IN 迴圈

- **for...in**迴圈，簡稱為**for迴圈**，通常拿來處理有次序性且已知應該執行次數的問題
- 在執行**for**迴圈時需要下列三個條件：
 - 設定控制變數的初始值
 - 設定控制變數的終止值，就是迴圈的結束條件
 - 設定控制變數值的變動方向與量，就是迴圈的迭代



FOR...IN 迴圈

- for迴圈裡面的expression
 - 變數（如常用的*i*）
 - 變數序列（iterable迭代）如list、tuple與range()函數
- else是選擇性設定，後面的程式碼會在for迴圈中止之後被執行。

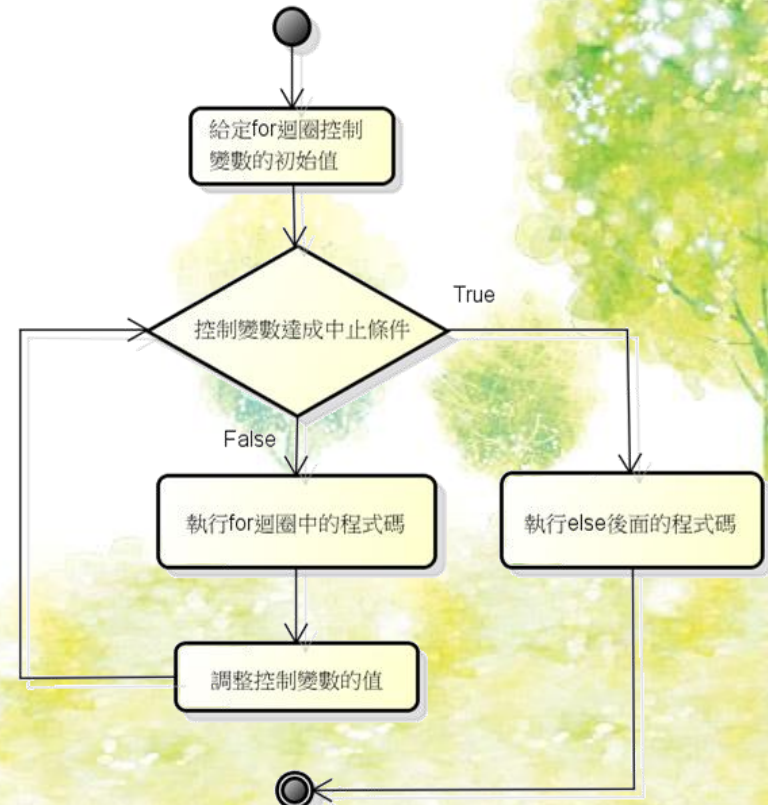
for expression in iterable:

欲執行的程式碼1...

else:

欲執行的程式碼2...

```
for i in [1,2,3,4,5]:  
    print(i)  
else:  
    print('finish')
```



RANGE()

- 一個在for迴圈中常被使用的函數——**range()**
- 顧名思義，**range**就是範圍的意思。

函數	描述
<code>range([start], stop[, step])</code>	建立整數序列

- 如上表所示，**range()** 所必須要給予的傳入值是
 - **stop**，也就是停止條件
 - **start**是起始值，預設為0
 - **step**是迭代數字，預設為1(可為負數)。
- **start**跟**step**都是選擇性選項屬性
- 三者所需要的**資料型態**都是**int**
- **通常**會將**range()****轉型**成其他容器資料型態如**list**、**tuple**等



RANGE()

```
01 print(range(10)) # 直接印出range() 會印出range 物件，使用意義不大
02
03 print(list(range(10))) # 將range 物件轉型為list
04 print(list(range(3, 15, 4)))
    # 設定起始值為3，終止值為15，迭代數字為4
05 print(list(range(10, -14, -3)))
    # 設定起始值為10，終止值為-14，迭代數字為-3
06
07 print(tuple(range(-7, 24, 5))) # 也可以將range 物件轉型為tuple
```

```
>>>
```

```
range(0, 10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[3, 7, 11]
```

```
[10, 7, 4, 1, -2, -5, -8, -11]
```

```
(-7, -2, 3, 8, 13, 18, 23)
```

```
>>>
```



FOR...IN 迴圈

- for 迴圈通常拿來執行已知執行次數的程式碼，例如：

```
01 sum = 0                # 將sum 給定初始值為0
02 for i in range(101):   # 設定for 迴圈的執行條件，讓i 從0 迭代到100
03     sum += i           # 將i 的值加到sum
04 else:
05     print(' 將1 到100 加總，所得總和為', sum)
                                # 在迴圈執行結束時將sum 值印出
```

```
>>>
```

```
將1 到100 加總，所得總和為 5050
```

```
>>>
```

- range() 終止值的結果並不會被輸出，所以range的範圍要+1



小練習

- 修改上述程式
- 使使用者輸入一數字 n
- 算出1加到 n 的結果
- ※ if判斷無法連續執行 只能再按一次F5

```
>>>
請輸入一數字n : 5
1加到 5 的總和為 : 15
>>> =====
>>>
請輸入一數字n : 10
1加到 10 的總和為 : 55
>>> |
```



小練習

- 修改上述程式
- 使使用者輸入一數字 n
- 算出 n 階乘的結果

```
請輸入一數字n : 3
3 的階乘為 : 6
>>> =====
>>>
請輸入一數字n : 5
5 的階乘為 : 120
>>>
```



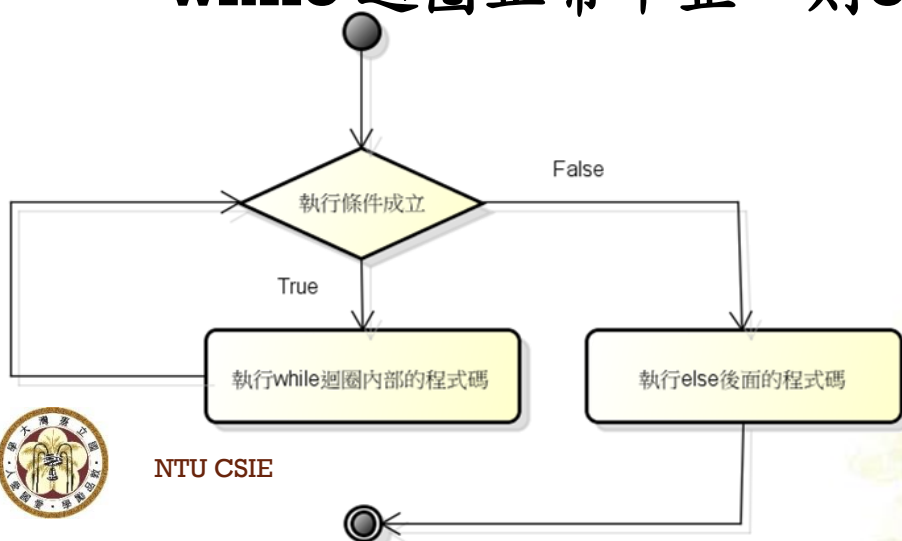
練習

- 輸入一個大於0的整數n
印出" $1+2+3+\dots+n = \text{結果}$ "
- 例如: 輸入5, 印出 $1+2+3+4+5 = 15$
- hint : 想在中間的數字加上「+」號該怎麼做呢？



WHILE 迴圈

- **while**迴圈就比較適合拿來處理沒有次序性，或者不知道總共需要執行幾次的問題 (但有次序性的也可只是較適合)
- 而**while**迴圈的結構如下：
- **boolean_expression** (布林) 是**while** 迴圈執行的條件
- **while** 迴圈只有終止條件的敘述，因此控制變數的初始值跟調整方式就要在其他地方設定。
- **while** 迴圈正常中止，則**else:** 後面的程式碼就會被執行



while boolean_expression:

欲執行的程式碼1...

else:

欲執行的程式碼2...

```
a = 0
while a < 10:
    print(a)
    a += 1
```



WHILE 迴圈

01	sum = 0	# 設定sum 為0
02	count = 0	# 設定count 也就是起始條件為0
03	while count < 100:	# 當count<100 時執行以下的程式碼
04	count += 1	# 每次執行時count+1
05	sum += count	# 將count 的值加總到sum
06	else:	# 當while 迴圈結束時
07	print('1+2+3+...+%d=%d'%(count, sum))	# 利用格式化輸出

>>>

1+2+3+...+100=5050

>>>



WHILE 迴圈

- 只能以while 迴圈解決的問題來當作例子：

```
01 sum = score = 0           # 將sum 和score 初始值設為0
02 count = -1                # 將count 初始值設為-1
03
04 while score != -1:        # 當score 不等於-1 時
05     count += 1            # 計數器+1
06     sum += score          # 將分數加總
07
08     # 讓使用者輸入分數，並告知輸入-1 則結束並計算平均
09     score = eval(input('輸入分數，如果輸入-1 則結束並計算平均'))
10 else:
11     # 以格式化輸出，控制輸出到小數點後第二位
12     print('平均：%0.2f'%(sum / count))
```



WHILE 迴圈

■ 執行結果

>>>

輸入分數，如果輸入-1 則結束並計算平均75

輸入分數，如果輸入-1 則結束並計算平均93

輸入分數，如果輸入-1 則結束並計算平均91

輸入分數，如果輸入-1 則結束並計算平均87

輸入分數，如果輸入-1 則結束並計算平均-1

平均：86.50

>>>

- 上面的例子是讓使用者輸入分數之後，再進行加總和平均
- 因為要讓使用者動態決定要輸入幾筆資料，因此使用 **while** 迴圈來做處理會較適合。



小練習

- 質數判斷程式
- 令使用者輸入數字 n
- 並判斷 n 是否為質數

>>>

請輸入一個數字：2

2 是質數

>>>

>>>

請輸入一個數字：4

4 不是質數

>>>



巢狀迴圈

- 迴圈內的敘述區段還有迴圈

- ➡ 構成「巢狀迴圈」

- 使用巢狀迴圈時，

- ➡ 每個 **for** 必須有對應的：

- ➡ 迴圈間不允許交錯。

- ➡ 每個 **For** 迴圈必須有自己的計數變數，不可重複。

- ➡ 製作規則性表格使用

如：九九乘法表，重複性圖案。

- 試印出一列 長度為8的星號

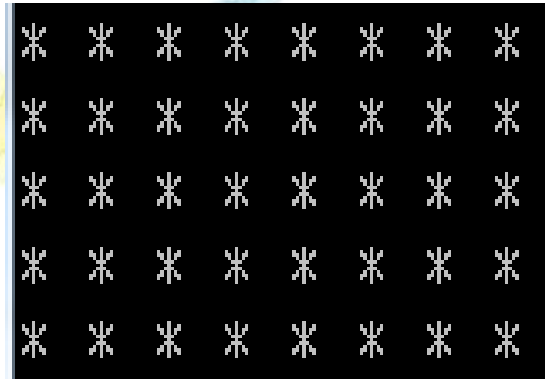
```
* * * * * * * *
```



巢狀迴圈

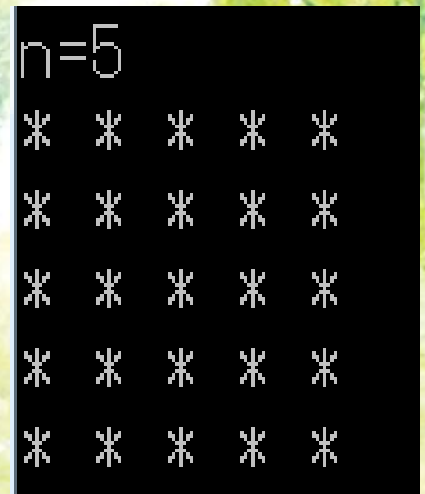
- 印出重覆性的圖案

```
for i in range(0,5):  
    for j in range(0,8):  
        print("*", end="")  
    print()
```



```
* * * * * * * *  
* * * * * * * *  
* * * * * * * *  
* * * * * * * *  
* * * * * * * *
```

- 請試著輸入一數字n並印出一 $n*n$ 之方塊



```
n=5  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```



巢狀迴圈

■ 巢狀迴圈操作練習

- 請寫一個程式，可輸入一個n利用 巢狀迴圈將這樣的圖形印在畫面中

```
*  
**  
***  
****  
*****
```

(a)

```
*****  
*****  
***  
**  
*
```

(b)

```
1  
2 3  
4 5 6  
7 8 9 10  
11 12 13 14 15
```

(c)



■ 九九乘法表

```
01 for i in range(1, 10, 1): # i 依照順序從1 到9，1 為每次迴圈i 的變動量
02     for j in range(1, 10, 1):      # j 依照順序從1 到9
03         print(str(j) + "*" + str(i) + "=" + str(i*j), "\t", end="")
                                # end 把換行字元換掉
04     print()                    # 換行
```

>>>

```
1*1= 1  1*2= 2  1*3= 3  1*4= 4  1*5= 5  1*6= 6  1*7= 7  1*8= 8  1*9= 9
2*1= 2  2*2= 4  2*3= 6  2*4= 8  2*5= 10 2*6= 12 2*7= 14 2*8= 16 2*9= 18
3*1= 3  3*2= 6  3*3= 9  3*4= 12 3*5= 15 3*6= 18 3*7= 21 3*8= 24 3*9= 27
4*1= 4  4*2= 8  4*3= 12 4*4= 16 4*5= 20 4*6= 24 4*7= 28 4*8= 32 4*9= 36
5*1= 5  5*2= 10 5*3= 15 5*4= 20 5*5= 25 5*6= 30 5*7= 35 5*8= 40 5*9= 45
6*1= 6  6*2= 12 6*3= 18 6*4= 24 6*5= 30 6*6= 36 6*7= 42 6*8= 48 6*9= 54
7*1= 7  7*2= 14 7*3= 21 7*4= 28 7*5= 35 7*6= 42 7*7= 49 7*8= 56 7*9= 63
8*1= 8  8*2= 16 8*3= 24 8*4= 32 8*5= 40 8*6= 48 8*7= 56 8*8= 64 8*9= 72
9*1= 9  9*2= 18 9*3= 27 9*4= 36 9*5= 45 9*6= 54 9*7= 63 9*8= 72 9*9= 81
```

>>>



回家作業

- 輸入一數字 n
- 印出 1 到 n 之間的質數
- Ex : n 為 100

>>>

2 is prime

3 is prime

5 is prime

7 is prime

11 is prime

13 is prime

17 is prime

19 is prime

23 is prime

NT 29 is prime

31 is prime

37 is prime

41 is prime

43 is prime

47 is prime

53 is prime

59 is prime

61 is prime

67 is prime

71 is prime

73 is prime

79 is prime

83 is prime

89 is prime

97 is prime

>>>



NESTED LOOPS

- 巢狀迴圈時常用string * 與 +來替代

....1
...2
..3
.4
5

Java/C/C++

```
1 for (int line = 1; line <= 5; line++) {  
2     for (int j = 1; j <= (5 - line); j++) {  
3         System.out.print(".");  
4     }  
5     System.out.println(line);  
6 }
```

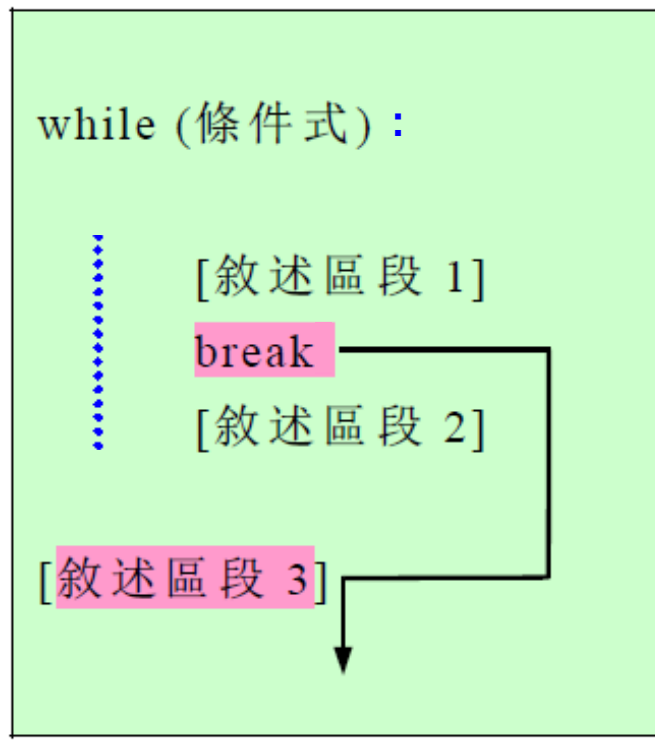
Python

```
1 for line in range(1, 6):  
2     print((5 - line) * "." + str(line))
```



強制結束：BREAK

- 如果在編寫有迴圈的程式碼時，有需要在終止條件達成前就強制終止迴圈使用 **break** 指令
- 用 **break** 跳出迴圈 **else:** 的程式碼就**不會**被執行



強制結束：BREAK

```
01 sum = 0                # 將sum 給定初始值為0
02 for i in range(101):   # 設定for 迴圈的執行條件，讓i 從0 迭代到100
03     sum += i           # 將i 的值加到sum
04     if(i == 99):       # 當迴圈執行最後一次時
05         print('break 示範')
06         break
07     else:
08         print('將1 到100 加總，所得總和為', sum)
                                # 在迴圈執行結束時將sum 值印出
```

```
>>>
```

```
break 示範
```

```
>>>
```



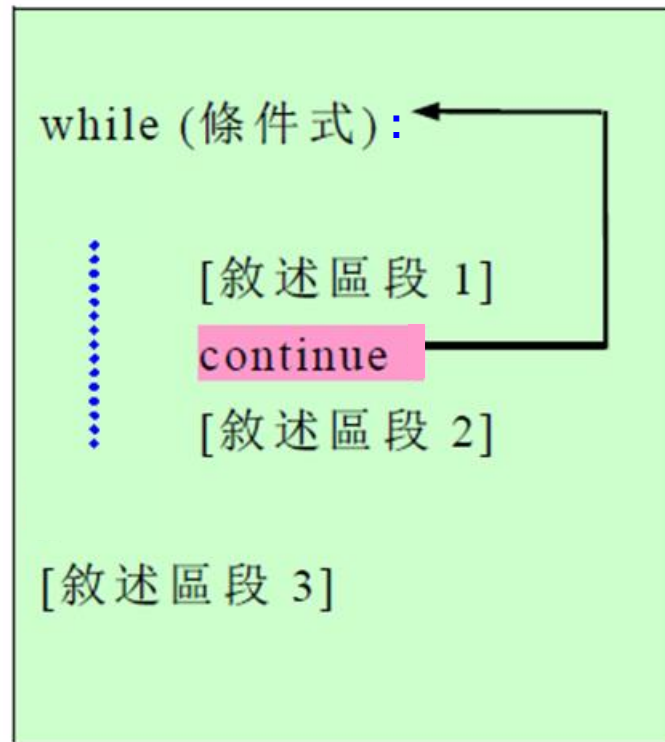
小練習

- 將上述程式改成
- 總和超過100分即跳離迴圈
- 並列出總分



強制回頭：CONTINUE

- continue：直接跳過後面的程式到迴圈開頭處繼續下一次執行



強制回頭：CONTINUE

- 團康遊戲「遇到三就跳過」，只要是三的倍數或者裡面含有三這個數字都跳過（如34、93等）
- 找尋在1到100之間符合條件的數有哪些？

```
01 i = 0                                # 設定初始條件
02 while i < 100:                        # 設定執行條件
03     i += 1                            # 設定i的調整條件
04     if (i % 3 == 0 or (i - 30 < 10 and i - 30 > 0) or ((i - 3) % 10 == 0)):
05         continue                    # 所有包含3的整數都跳過
06     print(i, end=" ")
```

>>>

1 2 4 5 7 8 10 11 14 16 17 19 20 22 25 26 28 29 40 41 44 46 47 49 50 52
55 56 58 59 61 62 64 65 67 68 70 71 74 76 77 79 80 82 85 86 88 89 91
92 94 95 97 98 100

>>>



強制回頭：CONTINUE

■ 列出一數字的因數

```
01 Number = input(" 請輸入一個數字：")
02
03 print("%s 的因數有：" % Number, end = "")
04
05 for i in range(1, int(Number) + 1):
06     if(int(Number) % i != 0): # 若Number 無法被i 整除，則不是因數，
07         continue # 直接跳過而不輸出
08     print(i, end = ",")
```

```
>>>
```

```
請輸入一個數字：12
```

```
12 的因數有：1,2,3,4,6,12,
```

```
>>>
```



回家作業

- 猜數字進階版-終極密碼

令程式產生1個1~100間的亂數

令使用者輸入一所猜之數字

若未猜中 程式須提示使用者縮小之範圍

直至猜中為止程式方可結束

- 寫好後可以大家一起玩懲罰遊戲喔！



