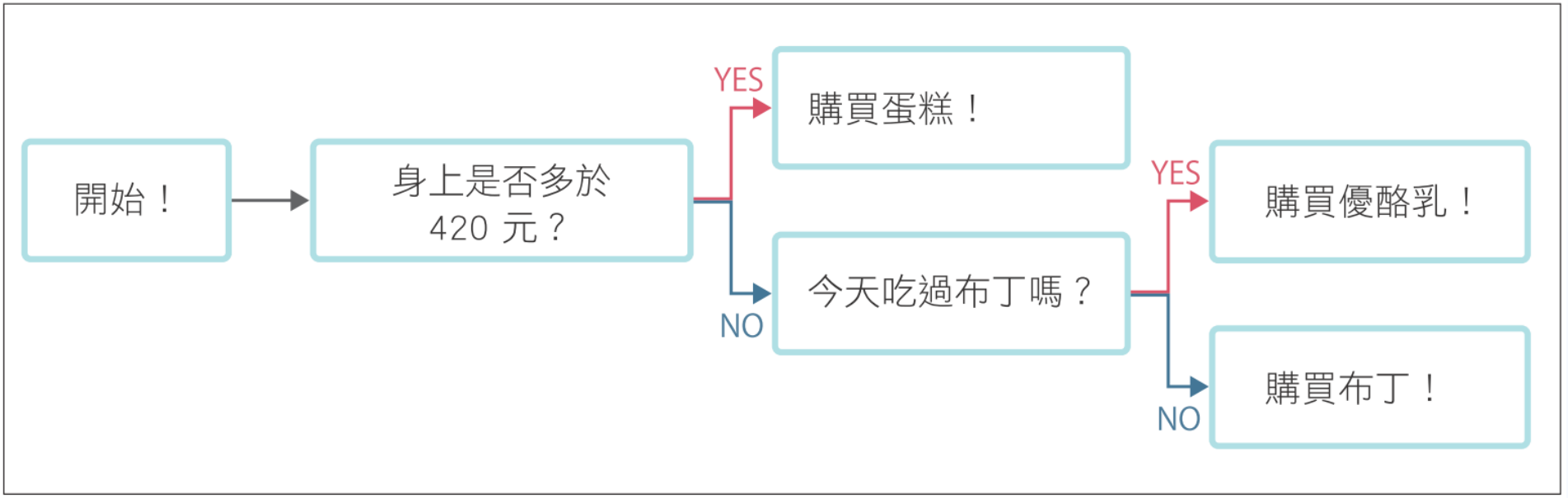
第 3 章

程式控制方式

- 什麼是條件分支？
  - 條件分支即是「程式按照當時狀況、分別執行不同處理動作的功能」。

日常生活中隨處可見的條件分支

條件分支的實例



- State above by design thinking ….
  - –If ….
  - –If …
  - –Because …

語法

```
if xx :
    tab ▽▽
```

Coding Thinking: 思考電影售票的流程 (一)

- 電影分級系統 1

語法

```
if (18歲以上):
    賣出電影票
```

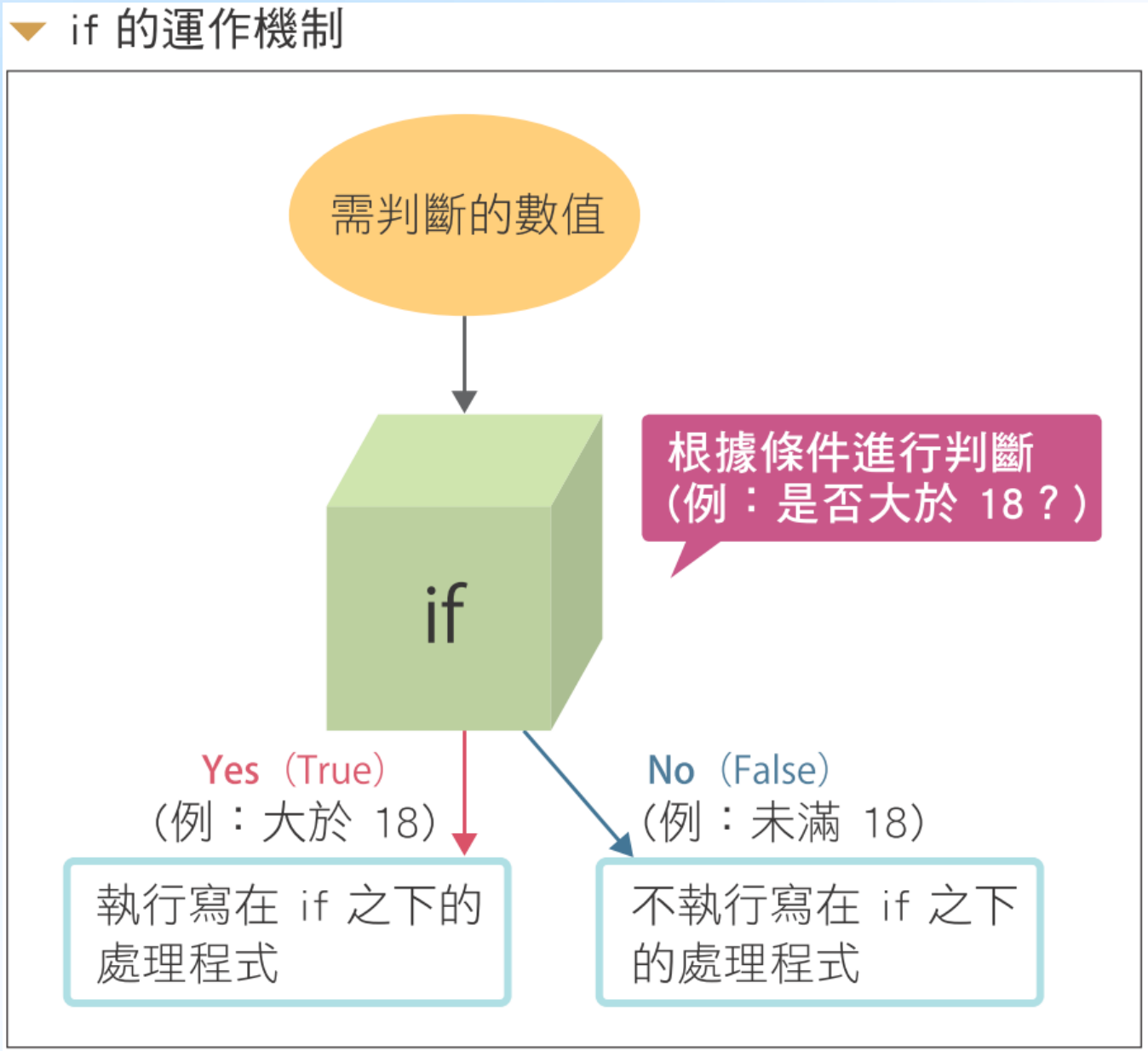
```
if (18 <= age):
    print('賣出電影票')
```

Try coding...


```
>>> age = 29 ↵          將購票客人的年齡 29 指派給變數 age
>>> if (18 <= age): ↵   此處為條件分支，age 大於 18 的時候
                        就執行下一行
... tab print('賣出電影票') ↵   顯示「賣出電影票」的訊息
... ↵                   再次按下 Enter 鍵，代表結束，並執行 if 部分的程式碼
賣出電影票               顯示結果
```


```
>>> age = 15 ↵          這次將客人的年齡設為 15
>>> if (18 <= age): ↵   條件和之前相同，年齡是否大於 18 歲
... tab print('賣出電影票') ↵   年齡大於 18 時，會執行此行的
                        print 函式
... ↵                   按下 Enter 鍵
>>>                      和之前不同，沒有任何訊息
```

If controlling...



Coding Thinking: 思考電影售票的流程(二)

- 電影分級系統 2

語法

```
if (條件):
    條件為 True 時執行的部分          [A]
else :
    條件為 False 時執行的部分         [B]
```

```
if (18 <= age):
    print('賣出電影票')
else:
    print('無法賣出電影票')
```

Coding practice...


```
>>> age = 15 ↵
>>> if (18 <= age): ↵
... tab print('賣出電影票') ↵
... else: ↵
... tab print('無法賣出電影票') ↵
... ↵
無法賣出電影票
```

Coding Thinking: 思考電影售票的流程(三)

- 敬老優待票 by if... elif... else...

語法

```
if (條件式 1):
    條件式 1 為 True 時執行的部分 ———— [A]
elif (條件式 2):
    條件式 2 為 True 時執行的部分 ———— [B]
else :
    條件式 1 與條件式 2 皆為 False 時執行的部分 ———— [C]
```

Coding practice...

```
01 if (60 <= age):
02 [tab] print ('票價為 1000 元')
03 elif (18 <= age):
04 [tab] print ('票價為 1800 元')
05 else:
06 [tab] print ('無法賣出電影票')
```

if age > 60 ....


```
>>> age=70 ↵
>>> if (60<=age): ↵
... tab print ('票價為 1000 元') ↵
... elif (18<=age): ↵
... tab print ('票價為 1800 元') ↵
... else: ↵
... tab print ('無法賣出電影票') ↵
... ↵
票價為 1000 元
```

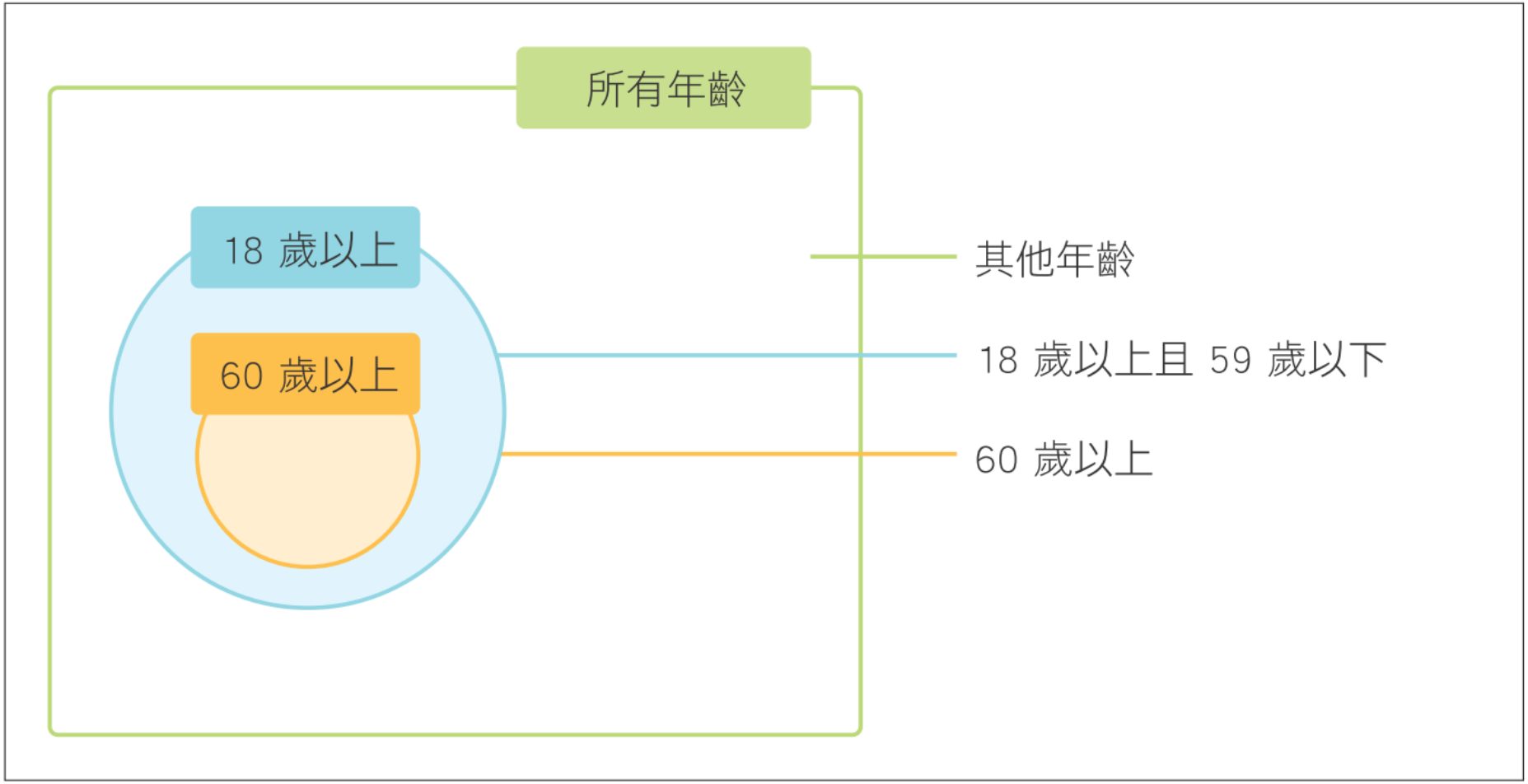
Any problem below coding....

- 各項條件的撰寫順序：由上而下...


```
>>> age=70 ↵                          這裡設定客人為 70 歲
... if (18<=age): ↵                   先判斷是否大於 18 歲並執行對應動作
... tab print ('票價為 1800 元') ↵
... elif (60<=age): ↵                 再判斷是否大於 60 歲並執行對應動作
... tab print ('票價為 1000 元') ↵
... else: ↵
... tab print ('無法賣出電影票') ↵
... ↵
票價為 1800 元
```

若改變了順序，則結果跟著改變...

以圖形表達條件



更嚴謹的思考邏輯像這樣寫... (記得：寫程式防錯的必要)


```
>>> age=70 ↵
>>> if (18 <= age <= 59): ↵
... tab print('票價為 1800 元') ↵
... elif (60 <= age): ↵
... tab print('票價為 1000 元') ↵
... else: ↵
... tab print('無法賣出電影票') ↵
... ↵
票價為 1000 元
```

Coding Thinking: 思考電影售票的流程 (四)

- 集滿 5 次有優惠價
  - –條件：
    - •客人有集點卡
    - •客人已看過5部電影


```
>>> pointcard = True ↵
>>> count = 5 ↵
>>> if (pointcard == True): ↵
... tab if (count == 5): ↵
... tab tab print ('感謝您的長久惠顧，此次為 1000 元優惠價') ↵
```

Coding thinking...

```
if (身上帶著集點卡、而且已經累積看過 5 部電影):
    print('感謝您的長久惠顧，此次為 1000 元優惠價')
```


```
>>> pointcard = True ↵
>>> count = 5 ↵
>>> (pointcard == True) ↵
True
>>> (count == 5) ↵
True
>>> ((pointcard == True) and (count == 5)) ↵
True
```

客人沒帶卡...客人沒帶卡...


```
>>> pointcard = False ↵
>>> count = 5 ↵
>>> (pointcard == True) ↵
False
>>> (count == 5) ↵
True
>>> ((pointcard == True) and (count == 5)) ↵
False
```

客人有帶卡

Console

```
>>> pointcard = True
>>> count = 5
>>> if ((pointcard == True) and (count == 5)):
...     print('感謝您的長久惠顧，此次為 1000 元優惠價')
...
感謝您的長久惠顧，此次為 1000 元優惠價
```

Review all conditions ...

- 18歲以下不賣此電影
- 18歲以、59以下1800
- 60以上1000
- 帶卡且累積5次則票價1000元

一開始為何檢查小於18歲？

- 寫法不只一種...

```
if(小於 18 歲):
    無法賣出電影票
elif(60 歲以上):
    票價為 1000 元
elif(集滿 5 次的優惠對象):
    票價為 1000 元
else:
    票價 1800 元
```

- 如何再減少程式碼？
- 如何再少一個elif判斷句？
- 做到彙整相同的處理動作即可...

結過一樣的合併處理（紅字的部分）

```
if(小於 18 歲):
    無法賣出電影票
elif(60 歲以上):
    票價為 1000 元
elif(集滿 5 次的優惠對象):
    票價為 1000 元
else:
    票價 1800 元
```

```
A條件 or B 條件
```

And / Or 用來彙整條件

- 條件都要成立用and
- 條件之一成立即可用or

(A 條件 and B 條件) 的時候

A 條件	B 條件	結果
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

(A 條件 or B 條件) 的時候

A 條件	B 條件	結果
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

用or來彙整, 如下....

```
if(小於 18 歲):
    無法賣出電影票
elif(60 歲以上 or 集滿 5 次的優惠對象):
    票價為 1000 元
else:
    票價為 1800 元
```

Try ... (少兩行程式)


```
>>> age = 35 ↵
>>> pointcard=True ↵
>>> count=5 ↵
>>> if (age < 18): ↵
... tab print('無法賣出電影票') ↵
... elif ((60 <= age) or ((pointcard == True) and (count == 5))): ↵
... tab print('票價為 1000 元') ↵
... else: ↵
... tab print('票價為 1800 元') ↵
```

說出這行的意思...

- 注意執行順序-括弧先做

```
elif ((60 <= age) or ((pointcard == True) and (count == 5))):
```

```
(A 條件 or (B 條件 and C 條件))
```

實際上的電影售票系統畫面

電影票販售系統的畫面範例

請選擇票券的種類

0/6 張　選擇完畢

全部清除

一般 General	1,800 元	張	
孩童 Child (3&up)	1,800 元	張	
大學生 Student (Collage)	1,500 元	張	
年長者 Senior	1,500 元	張	
高中生 Student (High)	1,800 元	張	
夫妻 50 優惠 Marriage 50 Discount	1,800 元	張	
中 / 小學生 Student (Jr.High, Elementary)	1,800 元	張	

返回

① 選擇上映場次 ② 選擇觀賞人數 ③ 選擇座位 ④ 選擇票券種類 ⑤ 支付款項

假如xx則執行YY 稱為條件分支

- 條件分支

語法

```
if (條件 A):
    處理動作          ——  當條件 A 為 True 時執行的部分
elif (條件 B):
    處理動作          ——  當條件 A 為 False、條件 B 為 True 時執行的部分
else :
    處理動作          ——  當條件 A 和條件 B 皆為 False 時執行的部分
```

(tab)

Try pp. 3-21 ...

```
>>> x = 100
>>> if (x <= 10):
... tab print('x 小於 10')
... elif (x < 30):
... tab print('x 是比 30 小的數字')
... else:
... tab print('x 大於 30')
...
x 大於 30
```

and 與 or

- 「A 條件 and B 條件」
  - 只有「A 條件為 True 且 B 條件亦為 True」的時候為 True, 其他狀況皆為 False
- 「A 條件 or B 條件」
  - 「A 條件為 True 或 B 條件為 True」的時候都為 True
  - 「A 條件和 B 條件皆為 False」的時候為 False。

迴圈

- 什麼是迴圈？
  - 電腦程式，「能處理大量的資料」。
  - 不過單純輸入資料，電腦並不會自動按照人們的想法，將這些資料處理完畢，必須把處理資料的動作或過程撰寫成程式。
- 要以程式碼告訴電腦 "重複執行相同的動作"

Your AI partner

- 熱狗麵包製作流成
  - 1. 加熱麵包
  - 2. 將熱狗煎好
  - 3. 將熱狗夾入麵包
  - 4. 加上番茄醬和黃芥末
- You: 接受點餐 收取現金
- AI:
  - 1
  - 2
  - 3
  - 4
- You: 將麵包交給客人

- for, while 用來做迴圈

for的使用方式

語法

```
for 變數名稱 in range(重複次數):
    想重複執行的處理動作
    想重複執行的處理動作
    .
    .
    .
```

Try ....


```
01 >>> for count in range(3): ↵
02 ... tab print('重複執行') ↵
03 ... tab print(count) ↵
04 ... ↵
重複執行
0
重複執行
1
重複執行
2
```

顯示'重複執行'字串

顯示 count 的內容值

解説

```
for count in range(3):
```

```
print('重複執行')
print(count)
```

for的應用, 用在哪裡...


```
>>> word = 'ninja' ↵
>>> for chara in word: ↵
... tab print(chara) ↵
... ↵
n
i
n
j
a
```

- 「資料」可以是String, List, Dict., Tuple, Set中的元素

語法

```
for 變數名稱 in 資料:
    使用變數執行的處理動作
```

這裡指的資料是包含多個字母的字串資料，或是包含多筆資料的容器型別資料

List type 的 for 迴圈

Console

```
>>> music_list = ['DEATH METAL', 'ROCK', 'ANIME', 'POP']
>>> for music in music_list:
...     print('now playing... ' + music)
...
now playing... DEATH METAL
now playing... ROCK
now playing... ANIME
now playing... POP
```

Dictionary type的 for 迴圈


```
01 >>> menu = {'拉麵':500, '炒飯':430, '煎餃':210} ↵
02 >>> for order in menu: ↵
03 ...  tab print(order) ↵
04 ...  tab print(menu[order] * 1.08) ↵
05 ... ↵
拉麵
540.0
炒飯
464.4
煎餃
226.8
```

```
menu = {'拉麵':500, '炒飯':430, '煎餃':210}
for order in menu:
```

While 迴圈

語法

```
while（條件式）:
    tab  重複執行的處理動作
```


```
>>> counter = 0 ↵
>>> while (counter < 5): ↵
... tab print(counter) ↵
... tab counter = counter + 1 ↵
... ↵
0
1
2
3
4
```

```
while (counter < 5):
```

無窮迴圈,,,,,, 死當.....


```
>>> counter = 0 ↵
>>> while (counter < 5): ↵
... tab print(counter) ↵
0
0
…省略
0
0
^C0 ———————— 迴圈在此處終止
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  KeyboardInterrupt
```

按「Control」+「C」跳出死當（break）


```
>>> while(True): ↵
... tab print('戰鬥') ↵
... ↵
戰鬥
戰鬥
…省略
^C ———— 迴圈在此處終止
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  KeyboardInterrupt
```

Try...


```
>>> while(True): ↵
... tab print('揮拳') ↵
... tab print('腳踢') ↵
... tab break ↵
... tab print('必殺奥義') ↵
... ↵
揮拳
腳踢
```

利用變數離開迴圈...


```
>>> power = 2
>>> while(True):
...     print('揮拳')
...     print('腳踢')
...     print('必殺奧義')
...     power = power -1
...     if (power == 0):
...         break
...
揮拳
腳踢
必殺奧義
揮拳
腳踢
必殺奧義
```

注意這裡要縮排兩次

解説

```
power = power -1
```

List & for 的運用; continue的運途


```
01 >>> family = ['ryu-ko', 'mako', 'satsuki'] ↵
02 >>> for kid in family: ↵
03 ...     print('早安！'+ kid) ↵
04 ...     print('起床') ↵
05 ...     print('吃早餐') ↵
06 ...     continue ↵
07 ...     print('出門上學') ↵
08 ... ↵
早安！ryo-ko
起床
吃早餐
早安！mako
起床
吃早餐
早安！satsuki
起床
吃早餐
```

插入 continue

此行程式碼被跳過沒有執行

小結

- 想利用 for 執行固定次數的迴圈時

語法

```
for 變數名稱 in range(重複次數):
    重複執行的處理動作（變數儲存著從 0 開始的數值）
```

- 想對串列型別的各項資料、重覆相同的處理動作時

語法

```
for 變數名稱 in 串列資料:
    重複執行的處理動作（變數儲存著串列的各項資料）
```

小結...

- 想對字典型別的各項資料、重覆相同的處理動作時

語法

```
for 變數名稱 in 字典資料:
    重複執行的處理動作（變數儲存著字典資料的 Key）
```

- 想利用 while 在特定條件下重覆執行迴圈時

語法

```
while (條件式):
    重複執行的處理動作
    用判斷句改變條件
```

小結...

- 想中途結束迴圈時

語法

在迴圈的程式碼中：

tab break

- 想跳過該輪迴圈的後續動作時

語法

在迴圈的程式碼中：

tab continue

函式

- 想想洗衣機做了哪些事...
  - 注水
  - 讓衣服沾濕
  - 重複旋轉和逆轉
  - 再注水和排水同時旋轉滾筒
  - 脫水
  - 烘乾
- 把每一件事作成函數

函式

- 函式的功用
  - –可以將多項處理動作彙集在一起，供後續呼叫使用
- 函式的製作方式

語法

```
def 函式名稱():
    處理動作 1
    處理動作 2
    .
    .
    .
```

- 函式的使用方式


```
>>> def washingMachine():
...     print('注水')
...     print('清洗')
...     print('洗淨洗劑')
...     print('脫水')
...     print('烘乾')
...
>>> washingMachine()
注水
清洗
洗淨洗劑
脫水
烘乾
```

呼叫函式

更聰明的洗衣函式

- 能處理各種條件的函式


```
>>> def softWash ():
...     print('注水')
...     print('輕柔清洗')
...     print('洗淨洗劑')
...     print('脫水')
...     print('烘乾')
```

Try...

- 利用條件分支撰寫不同的處理方式


```
>>> mode = 'soft' ↵
>>> if (mode == 'soft'): ↵
... tab print('輕柔清洗') ↵
... elif (mode == 'hard'): ↵
... tab print('強力清洗') ↵
... else: ↵
... tab print('一般清洗') ↵
... ↵
輕柔清洗
```

設定為輕柔清洗的模式

Try...

- 在函式中撰寫條件分支

參數的定義

```
>>> def washingMachine(mode): ↵
...     print('注水') ↵
...     if (mode == 'soft'): ↵
...         print('輕柔清洗') ↵
...     elif (mode == 'hard'): ↵
...         print('強力清洗') ↵
...     else: ↵
...         print('一般清洗') ↵
... ↵
```

```
def washingMachine(mode):
```

```
washingMachine('soft')
```

Try...

呼叫函式的同時傳遞參數

```
>>> washingMachine('soft') ↵
注水
輕柔清洗
>>> washingMachine('hard') ↵
注水
強力清洗
```

Try...


```
>>> washingMachine('normal')
注水
一般清洗
```

- 計算圓形面積的函式

撰寫函式

```
>>> def area(radius): ↵
... tab result = radius * radius * 3.14 ↵
... tab return result ↵
... ↵
>>>
```

呼叫函式

```
>>> area(5)
78.5
```

```
... tab return result ↵
```


```
>>> small = area(5) ↵
>>> big = area(10) ↵
>>> print(small) ↵
78.5
>>> print(big) ↵
314.0
```

Python 內建的函式

- len()

計算字串字數

```
>>> len('thunderbolt') ↵
11 ↵
```

計算元素數量

```
>>> animal = ['cat','dog','duck'] ↵
>>> len(animal) ↵
3
```

max()、min()


```
>>> max(100,10,50)
100
>>> min(300,30,3000)
30
```


```
>>> max('thunderbolt')
'u'
```

Try...


```
>>> min('1Aa') ↵
'1'
>>> max('1Aa') ↵
'a'
```

sorted()=> 排序並拆解，用在哪裡...


```
>>> sorted('thunderbolt') ↵
['b', 'd', 'e', 'h', 'l', 'n', 'o', 'r', 't', 't', 'u']
>>> sorted('1Aa') ↵
['1', 'A', 'a']
>>> sorted([100, 95, 55, 78, 80, 78]) ↵
[55, 78, 78, 80, 95, 100]
```

print()


```
>>> print(988+12)
1000
>>> print('Hey! World')
Hey! World
```

type()


```
>>> hatena_1 = 9800 ↵
>>> type(hatena_1) ↵
<class 'int'> ●———————— 数値型別
>>> hatena_2 = 'marshmallow' ↵
>>> type(hatena_2) ↵
<class 'str'> ●———————— 字串型別
>>> hatena_3 = ['osomatsu', 'karamatsu'] ↵
>>> type(hatena_3) ↵
<class 'list'> ●———————— 串列型別
```

dir() : 查詢某種資料型別有哪些方法可用...


```
>>> string = 'nikuman' ↵
>>> dir(string) ↵
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__
ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__
rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__
subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', lower', 'lstrip', 'maketrans', 'partition', 'replace',
'rfind', 'rindex', rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

dir() 不放值--> 我用過了哪些變數


```
>>> dir() ↵
['__builtins__', '__doc__', '__loader__', '__name__', '__package__',
 '__spec__', 'hatena_1', 'hatena_2', 'hatena_3', 'string']
```

Error , Exception

- 錯誤（Error）


```
>>> print('hello) ↵
  File "<stdin>", line 1
    print('hello)
                 ^
SyntaxError: EOL while scanning string literal
```

錯誤訊息

錯誤的類型

- 1. Python 語法（撰寫格式）有誤的時候
- 2. Python 執行過程中無法順利處理資料的時候

語法（撰寫格式）有誤的時候

- 當違反了 Python 語言所規定的語法，此時的錯誤訊息會顯示 SyntaxError（語法錯誤）的字眼，而 Syntax 這個單字即是「語法」的意思。

Python 無法順利處理資料的時候

- 「Python 試圖執行程式碼中撰寫的 prin 函式, 不過由於此名稱的函式尚未定義（宣告）, 所以程式無法順利執行」。

```
>>> prin('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'prin' is not defined
```

進一步認識例外（Exception）

- 例外就是「程式執行中無法順利處理資料的錯誤」。

▶ Python 3.5 英文線上文件

URL https://docs.python.org/3.5/library/exceptions.html

▶ Python 2.7 英文線上文件

URL https://docs.python.org/2.7/library/exceptions.html

例外處理的使用方式

語法

```
try ：
    處理程式 A（有可能會發生例外狀況的程式段落）
except ：
    處理程式 B（發生例外時要如何處理）
```

例外處理的範例


```
>>> try: ↵
... tab prin('a') ↵
... except Exception as e: ↵
... tab print(e.args) ↵
... ↵
("name 'prin' is not defined",)
```