Python基本語法

地表最狂最有趣的基本語法

Scan me

周凡剛(Elwing)著

1 環境設置

1.1 電腦是什麼

對於我來說，電腦就是有很多『開關』組合起來的結合體，也就是由無數個『開』(大家說的 0) 或者『關』(大家說的 1) 組合起來的機器

所以大家會說電腦是個只懂 0 和 1 的機器

- 一個0或者1我們也會稱為一個bit
- 我們通常會用8個bits為一組，稱作1個byte
- 更多單位(使用二進位): 1KB = 1024bytes, 1MB = 1024KB
- 但在買硬碟的時候，硬碟廠商用的是1K = 1000，所以作業系統會看到比較少

1.2 程式是什麼

程式語言根據他的演化歷史大概分成三個階段，由機器語言 -> 組合語言 -> 高階語言

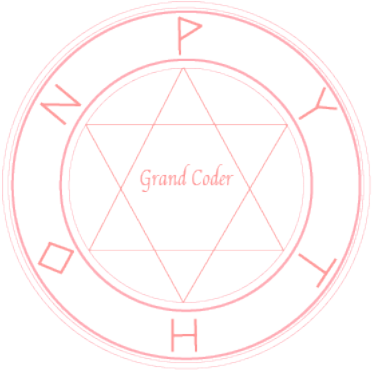
1.2.1 機器語言

機器語言是最初的的程式語言，就直接使用 0 和 1 來撰寫程式

1.2.2 組合語言

大家發現根本不可能直接用 0 和 1 來寫程式，所以 CPU 廠商制定了一套指令集，這些指令集由 CPU 負責『翻譯』成機器語言

- 所有的程式指令都是通過CPU來執行
- 組合語言例子: MOV R5 R6(把R6暫存器的值複製到R5)



1.2.3 高階語言

組合語言還是太基本，不適合人類直接撰寫，所以我們開發出了高階語言
你今天所有聽到的 C, Python, JAVA, Swift 都屬於高階語言的範疇
所以在執行程式的時候一定要做的就是要有個翻譯器把『高階語言』翻譯成『組合語言』

- 有各種的翻譯方式，直譯(執行時在翻譯)或者編譯(預先翻譯好)
- 不管什麼程式語言都要安裝對應的翻譯器(C++: GCC, Python: Python2/3)

1.3 需要安裝

不管我們在練習何種程式語言，我們都一定要安裝下面幾個軟體

1.4 翻譯器

不管寫什麼程式，我們一定需要把翻譯器安裝起來，Python 的翻譯器分成兩個系列

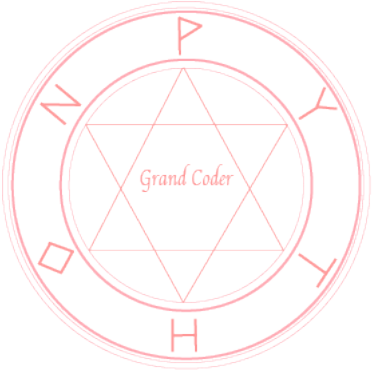
1. Python 2 系列: 已經停止版本更新，只會修正 bug，最後一個版本是 2.7

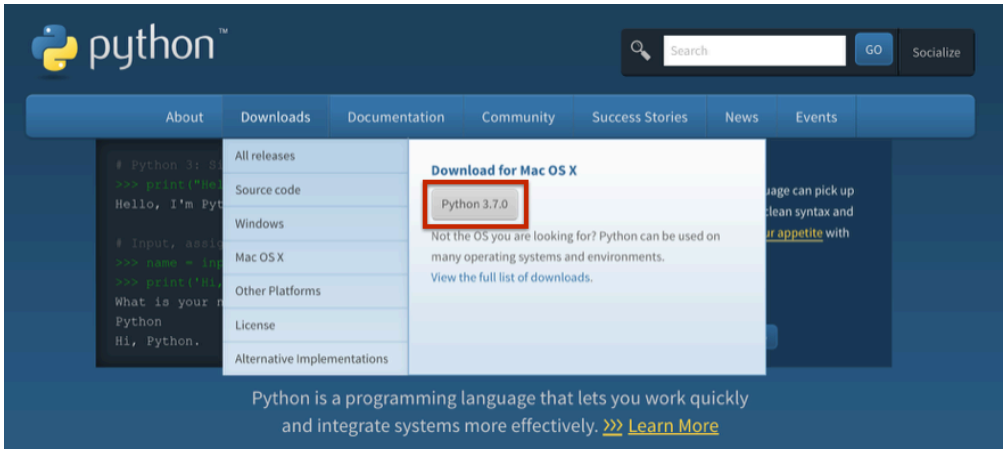
2. Python 3 系列: 跟 Python 語法有些許的不相容，目前還在更新中

那我們到底要安裝哪個版本呢?
這個市面上眾說紛紜的問題，我認為正確的答案是:『你想使用的函式庫 (工具) 最高支援哪一版本，你就只能用到那個版本，因為 Python 是個函式庫 (工具) 為主的語言』
所以當你遇到你覺得你的程式碼無論如何都沒錯，但卻一直執行錯誤的情況，我會建議你把你的 Python 翻譯器版本換成較為舊的版本
如果你不知道有什麼比較穩定，幾乎支援所有函式庫的版本，這裡推薦兩個版本

1. Python 2 穩定版本: 2.7. 最新版本
2. Python 3 穩定版本: 3.4.3

通常如果沒遇到上述的狀況，我們就可以安裝最新版本
請到 https://www.python.org/ 滑鼠移到 download 上，就有最新版本的下載點





下載完請一路安裝到完

1.4.1 翻譯器 32bit v.s. 64bit (Windows 必看)

事實上我們的作業系統有分成舊型 32bit(你也會看到 x86 這種稱呼) 和 64bit(你會看到 x64 這種稱呼)

那軟體就也有分 32bit 和 64bit 適用的

那你就會有個疑問了，我們的翻譯器是 32bit 還是 64bit 呢?

在回答這個問題前，我們先畫一個表來看 OS 和軟體的適配程度

軟體可以在作業系統上可以執行我們就打 O，否則打 X

能否執行	軟體 32bit	軟體 64bit
作業系統 32bit(通常 XP)	O	X
作業系統 64bit(通常 XP 以後)	O	O

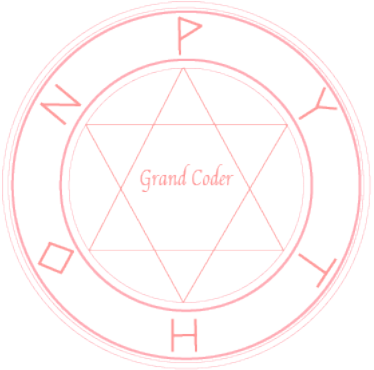
你會發現唯一不行的情況是新軟體配上舊的作業系統

那我們有時候做的功能會跟這有關係 (e.g. 打包成 exe)，你打包成的 exe 就跟你用的翻譯器是同一版本

所以如果你是 windows 的同學我會建議你先安裝 32-bit 版本 (事實上也是官網的預設值) 以防遇到 XP 的作業系統

那如果是其他作業系統那就不用想的選 64-bit 版本了

另外，如果在使用比較特別的函式庫 (e.g. TensorFlow, Keras) 也會被強迫使用 64-bit 的版本



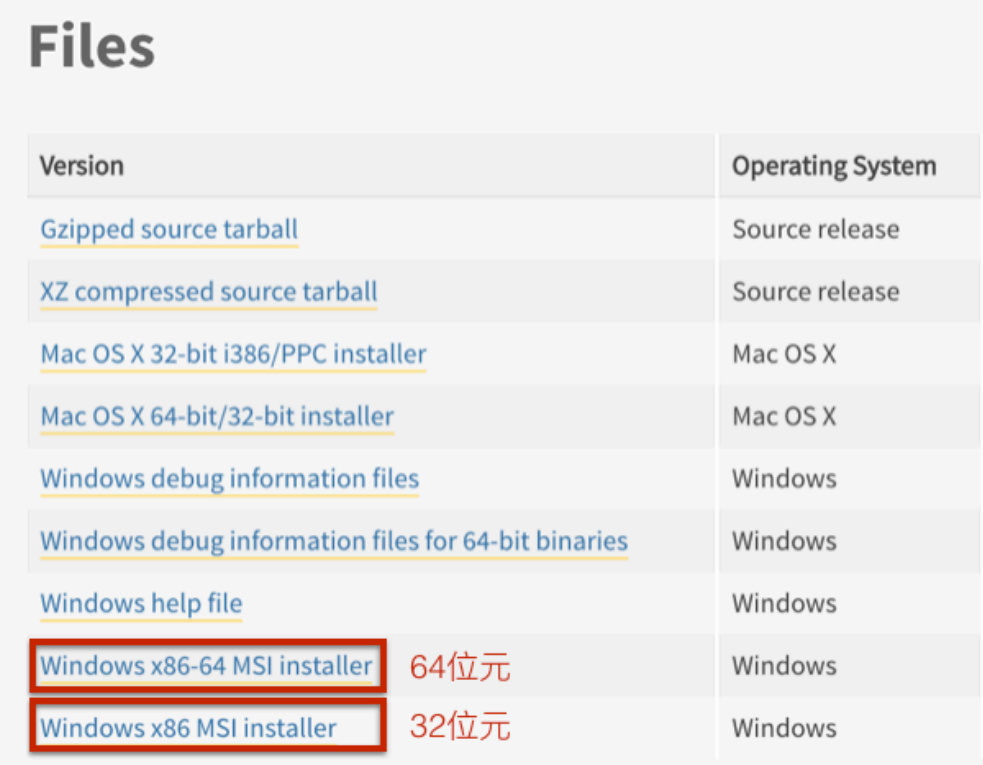
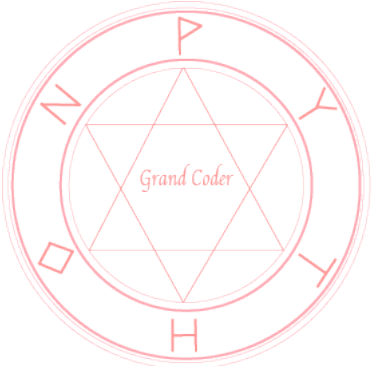


圖: 3.4.3 官網版本

1.5 輔助開發工具

有了翻譯器，其實我們已經可以開始寫程式了
不過我會建議安裝個輔助開發工具來幫你開發
這邊我選用 PyCharm 作為我們主要的輔助開發工具
因為 PyCharm 比較輕量級，而且可以好好地幫助你瞭解環境
首先先到 https://www.jetbrains.com/pycharm/ 官方網站下載 community 版本
接著安裝步驟如下



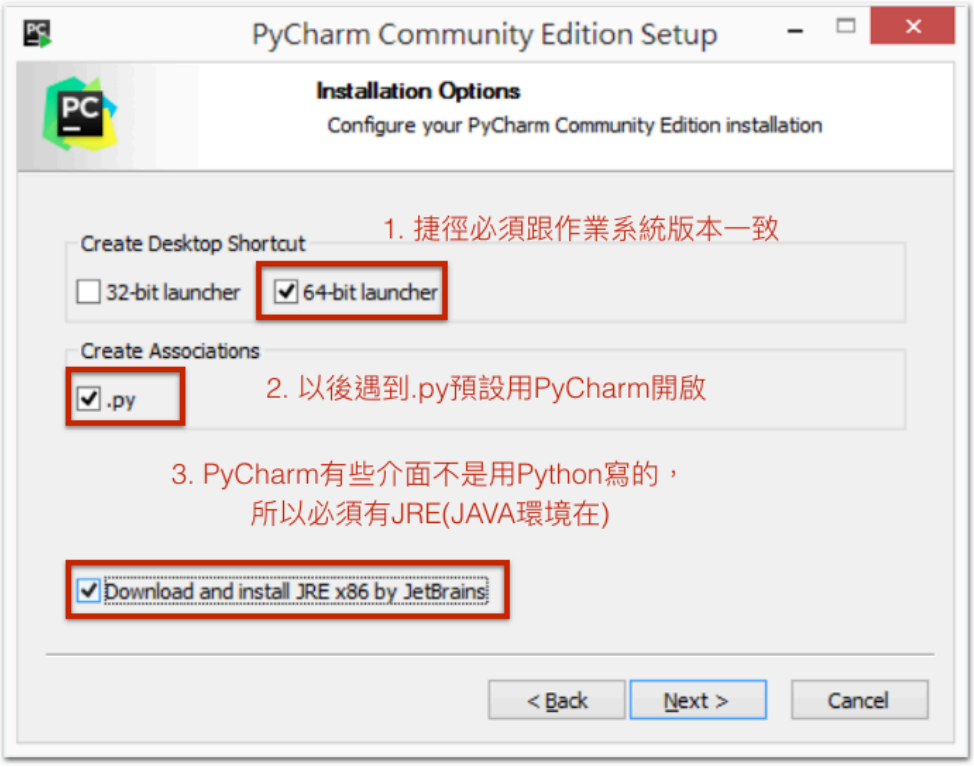
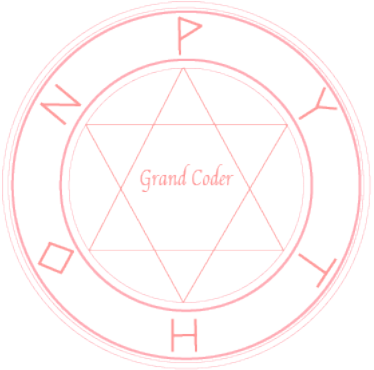


圖: 這裡要記得做出選擇

選好以後啟動程式



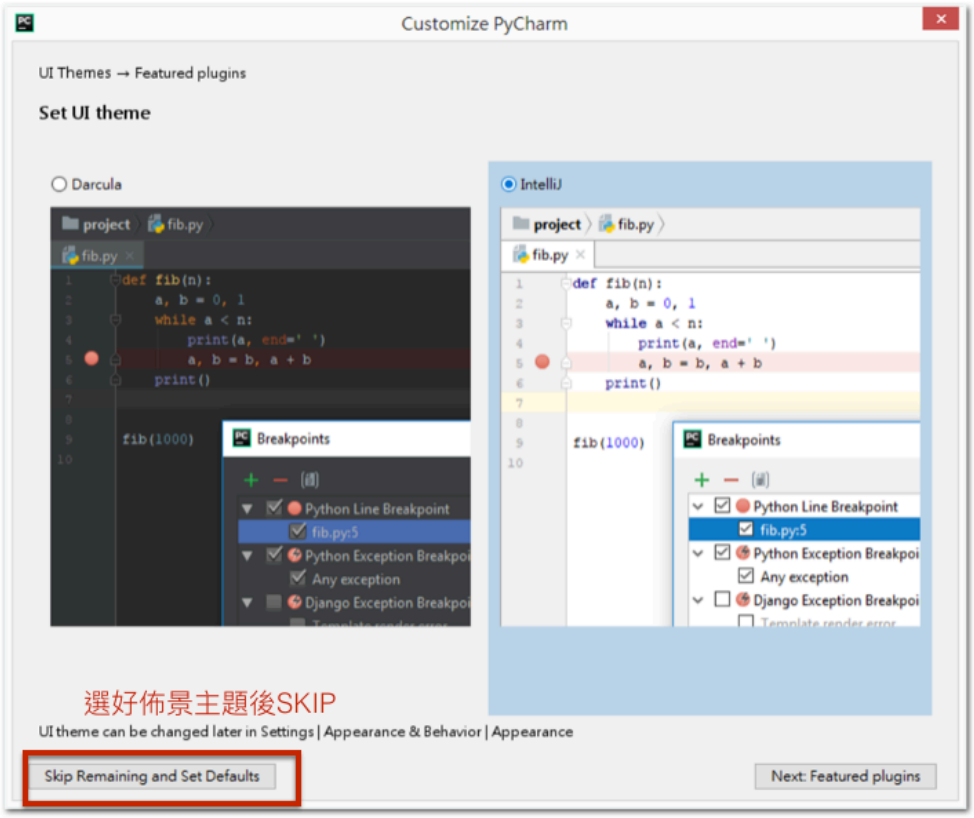


圖: 選完佈景主題 skip

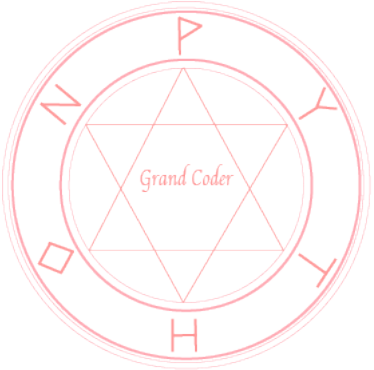
1.6 (同場加映) 線上環境

如果你是在有防火牆或者出外不方便的環境，我會建議你可以參考一下以下這個線上環境，但平常練習還是以線下環境為主

https://repl.it/repls

他有幾點好處

1. 所有東西寫完就存在雲端
2. 不用費神安裝環境
3. 你可以直接把網址列上的網址分享給別人



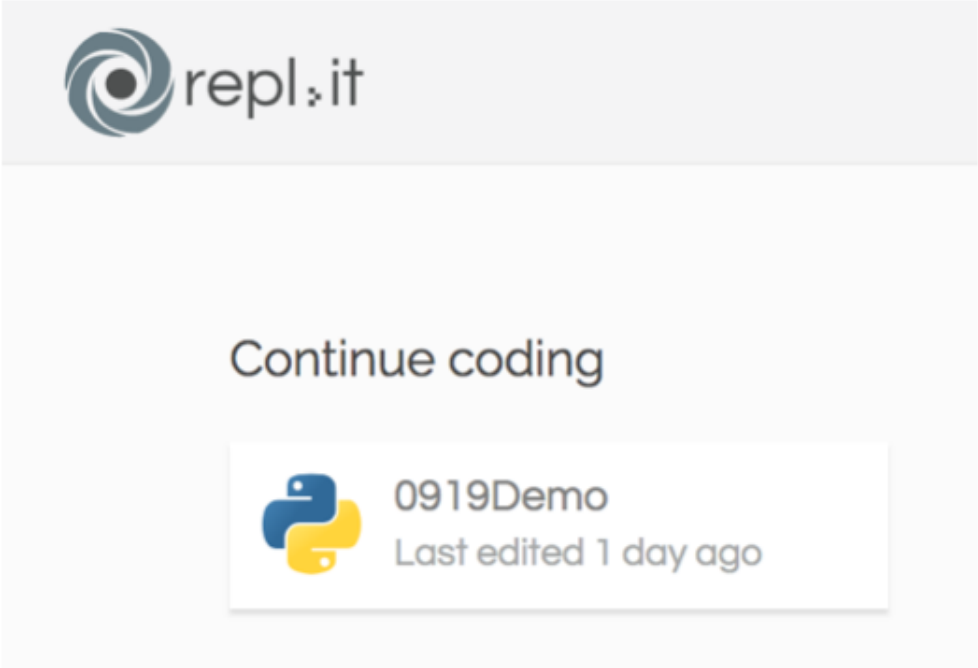


圖: repl 網站

1.7 第一個程式

1.7.1 開啟新 Project

首先在你開起來的 PyCharm 上選擇 Create New Project(第二次以後在 File)

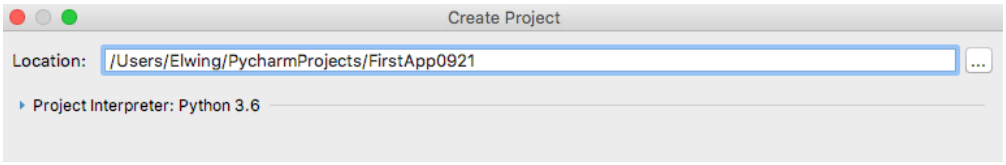


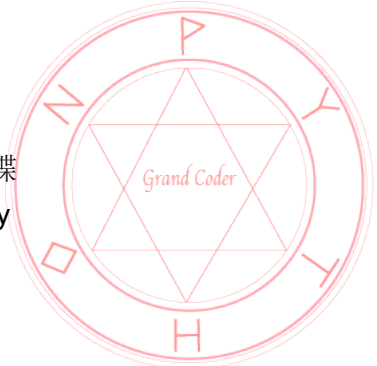
圖: 把你的 APP(Project) 開起來

在名稱那欄位改成你想要的名字，這裡有幾個『不要』

1. 不要中文
2. 不要空白鍵 (在命令列的時候容易被當成分隔符號)
3. 不要全形

1.7.2 開啟 Python 執行檔

Python 的構造很像我們一個遊戲光碟，APP(Project) 就是這光碟裡面可以有很多執行檔，不過這些執行檔不叫.exe，而是叫做.py



對 Project 點擊右鍵 -> New -> Python File

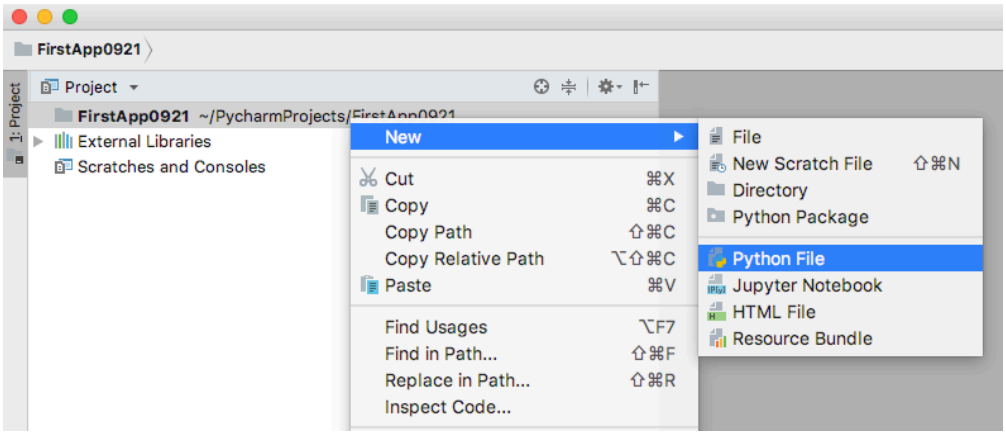


圖: 開啟方式

那我們通常第一支執行檔會命名為 main.py(主要的執行檔)

1.7.3 撰寫第一支 Python 程式

請在你的 main.py 區域幫我打上兩行

執行結果如下

```
[程式]: print(3 + 2)
        print("hello")

5
hello
```

1.7.4 基本概念

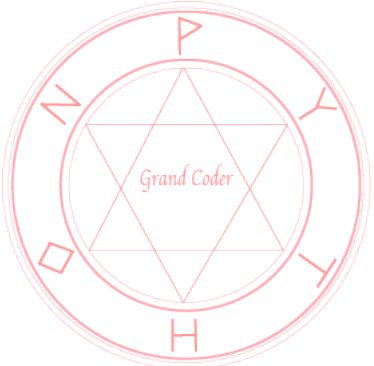
首先，我們要先知道程式的世界其實很簡單

他向我們的現實世界一樣，一樣有些基本的構成粒子!

這裡我們看到三種基本粒子中的兩種，數字 (直接寫就可以)，和文字 (要用雙引號或者單引號括號起來)

我們看到了兩種對於基本資料的操作，運算 (+)，和功能 (print 就是列印功能，我們把我們基本資料丟給他請他操作)

- 學程式的時候要盡量把程式的符號用中文去對照
- print後面的()在中文就是來承接丟進來資料的意思



1.7.5 註解

『程式碼是寫給別人，寫給未來的自己看的，而不是只是寫給電腦看的』
所以建議可以在一大段程式碼前面加個說明，讓自己回頭看的時候能看懂
只要在一行前面加個 **#**，這行就不會被當成程式處理，而是當成一個說明處理

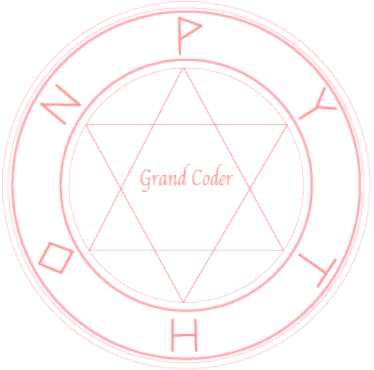
```
[程式]: # 這是註解: 練習一下基本資料
        print(3 + 2)
        print("hello")
```

```
5
hello
```

1.7.6 程式碼風格

最後在結束環境架設前，要跟各位討論一個很重要的事
『寫程式碼就像在寫文章，你的排版良好，大家一定會更喜歡看你的文章』
那可以建議初學者在一開始可以學習我的風格就可以了
有些基本風格會在後面的程式碼看到!

- 我在運算符號(e.g. +)左右兩邊會空出一格 -> 才不會太擁擠
- 我在逗號後也會空一格 -> 如同我們在寫英文一樣
- 兩大段的程式碼之間我會放下一個空行 -> 如同我們平常的分段



2 基本資料

這章節我們要就基本的資料來個全面的介紹

2.1 等號

在講數字之前，我們要先來看一個特別的符號 (=)

在現實的世界，= 這符號有兩個意思，1) 給名字: x = 3 2) 比較相等: 5 = 3 + 2

但在程式的世界，我們不能全部都要，於是我們在程式裡，= 只有『給名字的意思』

❤ 語法: 名字 = 資料

❤ 注意: 名字一定放在 = 左邊，資料一定放在 = 右邊

❤ 時機: 當你一個資料要用很多次的時候，給個名字就能重複使用

2.2 數字

數字比較需要注意的地方是其實還是分成小數和整數，Python 會依照你寫的型態自動幫你歸類

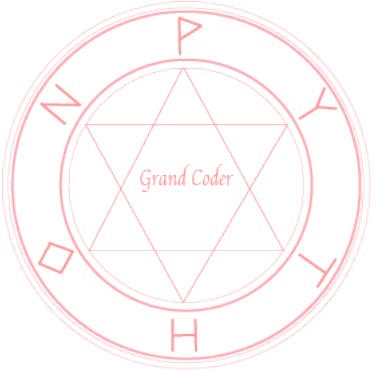
這裡比較要注意的是小數由於是使用科學記號法 (e.g. 十進位的科學記號法 $a \times 10^b$)，不過電腦是用二進位的科學記號法

由於科學記號法就會有無窮小數被『截斷』的問題，所以我們要知道小數是會有誤差的，這我們是覺得 OK 的 (e.g.50% 和 50.000001% 在普通的時候根本沒差)

初學者一定要在這裡習慣這件事! (如果需要精確運算，通常是計算機的時候，我們會使用 Decimal)

2.2.1 基本運算

這裡先介紹一點基本的操作



符號	用途
+	加法
-	減法
*	乘法
/	除法
%	做完除法取餘數
//	做完除法取整數
**	次方

[程式]:
```python
# = 應用: 算出數字, 給名字 a
a = 3 + 3.14
print(a)

# 重複利用 a, 並把結果叫做 b
b = a * 2
print(b)

# 進階! 非常重要!
# 把結果重新設定到 a 名字背後
# 那舊的自然就不見了, 而是換成新設定的東西
a = a * 3
print(a)
```

```
6.140000000000001
12.280000000000001
18.42
```

2.2.2 運算的禁忌

事實上在程式語言的基本，我們是不能把不同類型的東西拿來做運算的
譬如你寫下

```python
print("hello" + 3)
```

你就會看到

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-166b09aa1854> in <module>()
----> 1 print("hello" + 3)

TypeError: must be str, not int
```

那大家會有兩個問題

Q1. 為什麼其他程式語言可以呢?

A1. 因為那些程式語言偷偷先幫你轉換成同樣的類型，再開始做運算，Python 本質上是一個很嚴格的語言，所以他絕對不會幫你做偷偷轉換這件事!

Q2. 那如果我想印得漂亮一點，該怎麼做呢?

A2. print 這個技能可以帶入多個參數 (為何可以我們後面再說)，請看下面

```
[程式]: # 絕對不行, 不同類型
        # print(" 數字:" + 3.14)
        # 帶入多個參數
        print("數字:", 3.14)
```

```
數字: 3.14
```

2.2.3 基本技能

介紹一點基本的技能操作

abs(x)	絕對值
ceil(x)	比 x 大的最小整數
floor(x)	比 x 小的最大整數
pow(x, y)	x 的 y 次方
round(x [,n])	四捨五入，如果有帶入第二個參數，則會四捨五入到那個位數
sqrt(x)	x 的平方根

這裡要順便教各位一下什麼是技能

技能 (函數) 由三個部分構成:

1. 技能名字: 像 abs 這個技能的名字就是 abs

2. 參數 (技能需要的資料): 像絕對值這種技能是不可能單獨執行的，因為沒傳入東西不知道對誰做絕對值，用 () 來丟入需要資料

3. 回傳值 (回傳答案): 像 abs 做完就會有一個答案，但是像 print 這種技能是沒回傳答案的 (就像辦公室的印表機，一直印一直印，你不會停下來等答案)

```
[程式]: # abs 有回傳值, 我們可以直接用舊名字 a 來接他
        a = abs(-2 * a)
        print("絕對值:", a)

        # pow 需要兩個參數, 兩個參數之間我們用逗號隔開
        a = pow(a, 2)
        print("次方運算:", a)

        # bmi: (體重) / (身高公尺) ^ 2
        # 其實也可以使用 ** 來做次方
        bmi = 75 / pow(1.75, 2)
        print("BMI:", bmi)

        # 四捨五入到小數第二位
        bmi = 75 / (1.75 ** 2)
        print("BMI(四捨五入到小數第二位):", round(bmi, 2))
```

```
絕對值: 36.84
次方運算: 1357.1856000000002
BMI: 24.489795918367346
BMI(四捨五入到小數第二位): 24.49
```

2.2.4 (進階) 精確運算

這裡初學者不用練習，當要使用精確運算的話，我們會使用 Decimal
但事實上，使用時機非常非常少，大概只有計算機 APP 需要

```
[程式]: from decimal import *
        print(Decimal("3.14") + Decimal("3"))
```

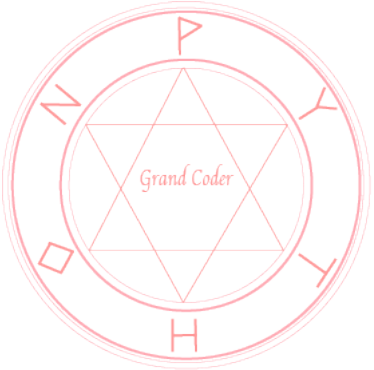
```
6.14
```

2.3 文字

文字就是一般人所說的『字串』，你需要使用『雙引號』或者『單引號』包起來
有時候你要特別注意：像 0912345678 這種電話不該是數字形態，而應該是文字型態 (1. 你不會去掉前面的 0 2. 你不會用幾萬幾千去唸他)

2.2.1 運算

文字支援 + 來做兩個文字的串連運算



2.3.2 技能

最常使用的技能就是 len 來算長度了

```
[程式]: a = "hello"
        print("原本:", a)
        a = a + "python"
        print("串連:", a)
        print("長度:", len(a))
```

```
原本: hello
串連: hellopython
長度: 11
```

2.3.3 特殊的操作

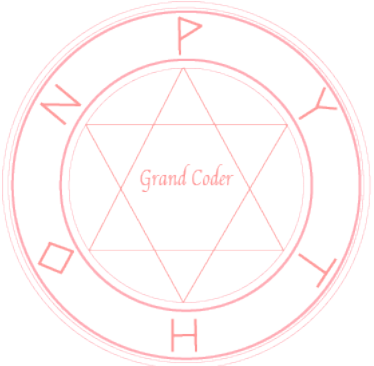
你可以透過做好來取得單一的字
假設我們創造了個 HELLO 這文字
你可以用正向座號或反向座號來取得對應的字
平常我們使用正向座號就可以了
這裡要注意一下：座號是從 0 開始，跟平常我們熟悉的 1 不太一樣

文字	H	E	L	L	O
正向	0	1	2	3	4
反向	-5	-4	-3	-2	-1

❤ 單一個字語法: 字串名 [座號]
❤ 多個字語法: 字串名 [想拿座號 1: 想拿座號 2+1] (第二個數字不包括，要多 +1)

```
[程式]: # 記得座號從 0 開始
        # [座號] -> 拿到背後的字
        a = "hello python"
        print("第三個字:", a[2])
        # 利用反向座號直接拿最後一個字, 就不用寫 a[len(a) - 1] 了
        print("最後一個字:", a[-1])
        # 也可以拿某一個字到某一個字
        # 要記得第二個數字要多 +1
        print("第三個字-第七個字:", a[2:7])
```

```
第三個字: l
最後一個字: n
第三個字-第七個字: llo p
```



2.3.4 (進階) 間隔拿取操作

我們可以在拿取子字串的操作中使用第二個: 來決定要間隔幾個字拿取一次

```
[程式]: a = "hello python"
        # 2->4->6->8
        print(a[2:9:2])
        # 如果前面什麼都不寫代表從頭, 後面什麼都不寫代表尾巴
        print(a[:])
        # 從頭到尾, 並且三個一拿
        print(a[::3])

lopt
hello python
hlph
```

2.3.5 逃脫字元

另外有一些單字是無法直接打出的，譬如: Enter 的換行 (被編輯器變成下一行程式)，TAB(被編輯器變成多個空白)

這時候我們會用 (逃脫字元) + 一個特殊的字來代表這個無法打出來的字

特殊符號	對應功能
\n	換行
\t	TAB
\b	BackSpace

```
[程式]: print("第一行\n第二行")

第一行
第二行
```

2.3.6 專屬技能

文字還有一個特別的使用方式，叫做專屬技能

白話文: 只有文字才能使用的技能

❤ 語法: 文字. 專屬技能 (參數)

❤ 注意: 這裡的. 就是我們中文『的』意思，也就是使用這個文字『的』一個專屬技能

❤ 參考: 請 Google “Python String method”，建議參考 https://www.tutorialspoint.com/python/python_strings.htm，最下面的 Built-in String Methods

我們先來一起練習其中的幾個

第一個: replace(“舊字串”, “新字串”, (選用) 要取代幾個)

```
[程式]: # 先準備一個字串
        a = "hellohellohello"
        # 使用 replace(取代的專屬技能)
        b = a.replace("hello", "goodbye")
        # 這裡要稍微注意, 你會發現 a 並沒有變動, 而是回傳一個新的答案
        print("原本的沒改變:", a)
        print("回傳新的答案:", b)
        # 所以如果你要 a 後面換成新的可以這樣寫 a = a.replace
        # replace 如果帶入第三個參數的話可以指定只取代幾個
        a = a.replace("hello", "goodbye", 2)
        print("設定回去:", a)

原本的沒改變: hellohellohello
回傳新的答案: goodbyegoodbyegoodbye
設定回去: goodbyegoodbyehello
```

再來練習兩個大小寫的轉換:

1. upper: 轉成大寫
2. lower: 轉成小寫

```
[程式]: # 這裡要注意, 由於他已經知道自己的所有東西, 所有不需參數
        print("轉成大寫", "hello".upper())
        print("轉成小寫", "HELLO".lower())

轉成大寫 HELLO
轉成小寫 hello
```

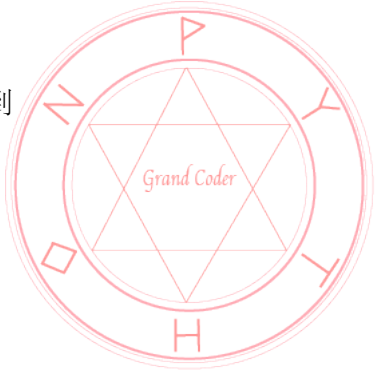
2.4 布林

有了文字，有了數字，我們已經可以表達大部分的資料了，但是如果今天我們要表示的『漂亮與否』的答案，文字和數字就不夠用了，因為這答案很明顯只有兩個!

一個叫『是』，一個叫『否』，在 Python 裡我們要寫『True』和『False』(沒錯！記得要第一個字大寫)

這就是我們的『是否資料』，也就是大家常說的『布林資料』

但是我們其實比較少直接寫 True 和 False，而是由別的運算得到



2.4.1 產生是否的運算

數字: **>**(大於), **<**(小於), **>=**(大於等於), **<=**(小於等於), **==**(相等, 因為一個 **=** 是給名字, 所以相等是 **==**)

文字: **==**(相等), in(檢查前面的字串是不是後面字串的一部份)

```
[程式]: # 直接使用 True False
        print("直接使用:", True)
        # 數字運算產生布林
        print("大於:", 5 > 3)
        print("相等:", 2 == 2)
        # 文字運算產生布林 (要記得大寫小寫是完全不同的)
        print("文字相等:", "hello" == "HELLO")
        print("文字包含:", "hel" in "hello")
```

```
直接使用: True
大於: True
相等: True
文字相等: False
文字包含: True
```

2.4.2 布林結合運算

有時候我們會把兩個布林運算結合在一起，其實就是我們中文說的『而且』和『或』，Python 寫成『and』和『or』

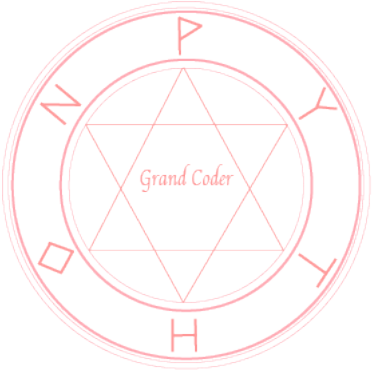
這裡其實非常好記，你只要想到一個例子：假設一個女生喜歡男生條件是『帥』『而且』『有才華』，看起來就是兩個都要有才行。但如果是『帥』『或者』『有才華』，看起來只要有一個可以就可以了!

我們詳細地列出表格

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

簡單的說，and 兩個都對才對，or 兩的都錯才錯



2.4.3 運算子順序

我們現在學過了很多種不同的運算，但是到底哪個運算先做呢? 這裡只告訴各位最重要的一句話!

千萬不要去背運算子順序!! 因為運算子順序其實是根據大部分人的直覺制定的

```
5 >= 3 + 2 and True
```

這裡我想你應該不會想先做 2 and True，你會覺得 and 兩邊應該要先做完，所以把 and 放後面點，而你也不會先做 5 >= 3，再把結果 + 2，所以實際上就是先算 3 + 2，再來 5 >= 5，再來 True and True

這裡就會很多人問，那不如就用 () 來確定誰先做，盡量也不要這樣做

```
((5 >= (3 + 2)) and True)
```

你覺得這樣寫和上面那樣寫那個比較漂亮呢?! 而且太多 () 會導致你不知道你少了幾個，也不知道下一個要加在哪裡

```
[程式]: print("and 結合:", True and False)
        print("or 結合:", True or False)
        # 請運用你對運算子的天然直覺
        print("複雜結合:", 5 >= 3 + 2 and True)
```

```
and 結合: False
or 結合: True
複雜結合: True
```

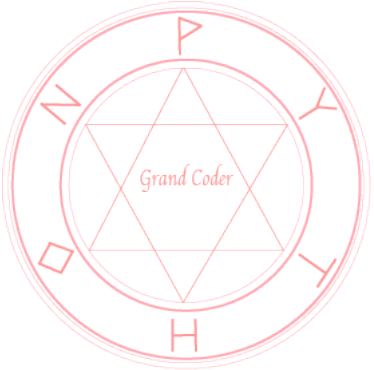
2.5 結論

現在你已經學會了程式語言基礎部分了!!!

我把他總結成一個讓你好記的口語

『三種基本資料』: 數字，文字，布林『二種基本操作』: 運算，技能『一個特殊符號』: =(給名字)

請好好記得這『三二一』!



3 if 語法

3.1 二選一

之前我們的程式都是從第一行執行到最後一行，但我們的程式理論上要有那種可以選擇的分支才對

其實就是我們平常中文說的:『如果』『否則』

中文是這樣說的

如果 XXX，我就做事情 1，否則我就做事情 2

換成 Python 語法

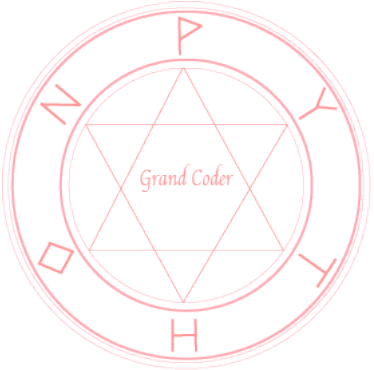
```
if 布林判斷:
    程式碼片段 1
else:
    程式碼片段 2
```

有幾個要注意的點

❤ if 和布林判斷間要加個空白鍵

❤ Python 當遇到這種有階層性架構，也就是『屬於』架構 (片段 1 屬於 if)，要加上縮排 (TAB) 和: 排版

```
[程式]: # 這裡我們學一個新的功能叫做 input, 可以讓使用者自行輸入
        # 但是這裡的輸入得到的答案一定是文字, 所以記得要轉換成你想要的
        # 我們用 float 轉換成數字的小數
        score = float(input("請輸入成績:"))
        if score > 60:
            print("PASS")
        else:
            print("FAIL")
```



```
請輸入成績:87.2
PASS
```

3.2 多選一

多選一的時候，你可以用『從上而下的單選題』來記

語法是這樣的

```
if 選項 A:
    A 對的話做完這段程式碼離開
elif 選項 B:
    B 對的話做完這段程式碼離開
elif 選項 C:
    C 對的話做完這段程式碼離開
else:
    上面都不對直接做完這段程式碼離開
```

3.2.1 注意的點

但是如果多選一是有包含關係的話，記得比較嚴苛的條件要放上面

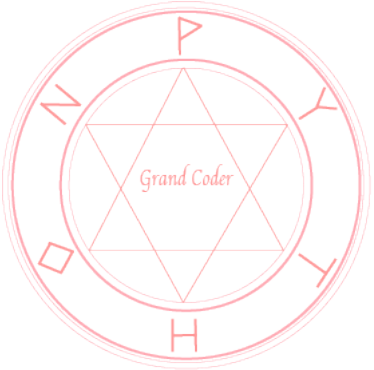
譬如下面三個選項

1. 成績 > 90
2. 成績 > 80
3. 成績 > 70

很明顯條件 1 是比較嚴苛的條件，就要放在最上面，做出第一次的過濾

```
[程式]: score = float(input("請輸入成績:"))
        if score > 90:
            print("RANK A")
        elif score > 80:
            print("RANK B")
        elif score > 70:
            print("RANK C")
        else:
            print("RANK D")
```

```
請輸入成績:78.2
RANK C
```



3.3 (實戰演練) 剪刀石頭布

剪刀石頭布是個看似簡單，但卻富有哲理的遊戲
如果你用最粗淺的想法來想，你會寫出 9 個 if-elif-else，因為你的 3 個狀況 x 對面的 3 個狀況
那我們就會想可不可以把輸，贏和平手真正的意義找出來

3.3.1 真正意義

其實剪刀石頭布是一個下一個拳一定贏前一個拳的遊戲
換句話說，如果換成選數字，就是下一個數字一定贏前一個數字的遊戲
❤ 重要概念: 當你發現其實東西的本質是有比大小，就像剪刀石頭布，請用數字來替代這些東西！因為會讓你比較大小很順利！

3.3.2 取餘數運算

這裡特別把取餘數運算拿出來特別說，因為取餘數運算其實是一個把值限制住的運算
a % b -> 小於 b 的數

3.3.3 (預先使用) 使用其他.py 的功能

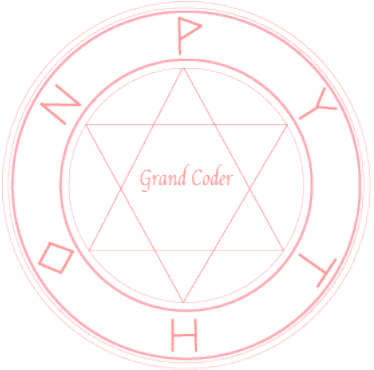
當我們要使用其他的.py(就算是內建的)，我們一定要先引用她
你可以想像成你在使用別人文章的句子，你要先說我要引用 (import) 這本書的哪一句 (功能)
於是我們使用 import random 來引用內建的 random.py
再使用裡面的 randint 功能 (使用的時候要用.，我們之前說過的『的』)
語法

```
import (某 py 的名字)
(名字).(裡面你想用的功能)
```

3.3.4 (預先使用) 清單

因為我們希望可以把 0, 1, 2 翻譯成剪刀石頭布
所以我們準備了一個翻譯清單 0 -> 剪刀 1 -> 石頭 2 -> 布
Python 準備清單是使用 []，用座號做出轉換一樣是清單名字 [座號]

```
[程式]: # 引用內建的 random.py
        import random

        me = int(input("請出拳 [0] 剪刀 [1] 石頭 [2] 布"))
        com = random.randint(0, 2)
```



```
trans = ["剪刀", "石頭", "布"]
# 清單名字 [號碼牌]
print("你出的拳:", trans[me])
print("電腦的拳:", trans[com])

if me == com:
    print("平手")
# 針對 me= 剪刀, com= 布 的 case 處理
# +1 相當於往下一個拳前進, 但是不該有 3, 布 +1 應該是剪刀才對
# 於是我們對 3 取餘數, 讓他回到 0
elif me == (com + 1) % 3:
    print("我贏了")
else:
    print("我輸了")
```

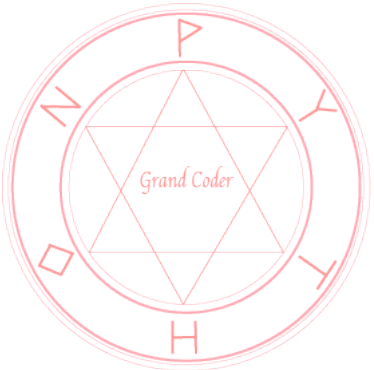
```
請出拳 [0] 剪刀 [1] 石頭 [2] 布 0
你出的拳: 剪刀
電腦的拳: 布
我贏了
```

3.4 (進階練習) 棒打老虎雞吃蟲

棒打老虎雞吃蟲很明顯也是跟剪刀石頭布類似的遊戲

先不要看下面的答案，練習一下你是否也可以寫出類似的小遊戲呢?

```
[程式]: import random

me = int(input("請出拳 [0] 蟲 [1] 雞 [2] 老虎 [3] 棒子"))
com = random.randint(0, 3)

trans = ["蟲", "雞", "老虎", "棒子"]

print("你出的拳:", trans[me])
print("電腦的拳:", trans[com])

# 跟剪刀石頭布不一樣的是平手條件反而最複雜
# 所以我們平手放在最後
if me == (com + 1) % 4:
    print("我贏了")
elif com == (me + 1) % 4:
    print("電腦贏了")
else:
    print("平手")
```



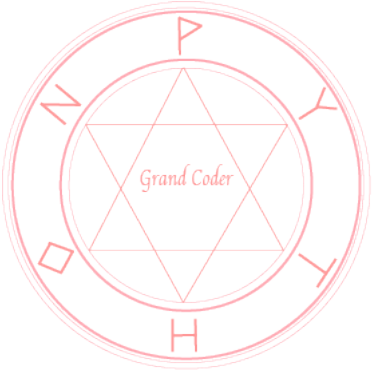
```
請出拳 [0] 蟲 [1] 雞 [2] 老虎 [3] 棒子 0
你出的拳: 蟲
電腦的拳: 雞
電腦贏了
```

3.5 結語

我們已經學會第一個重要的文法 if 了，還練習了一個剪刀石頭布遊戲

你應該慢慢發現寫程式其實只是把你用中文思考的邏輯用更清晰的方式轉換成為程式!

也發現，剪刀石頭布換個想法，竟然可以變得這麼簡單，這麼有擴展性，可以擴展到不管多少對多少的遊戲!



4 迴圈

首先先問各位一個問題，如果我們要印出十次 hello，應該怎麼做?
你可能會回答我:『打十行 print(“hello”)』
的確可以，但是如果是一千行，一萬行，這法子還有用嗎?
很顯然是不行的，我們需要一個方法可以讓我們只寫一行，但是 Python 知道要幫我做很多次!

4.1 迴圈基本構成

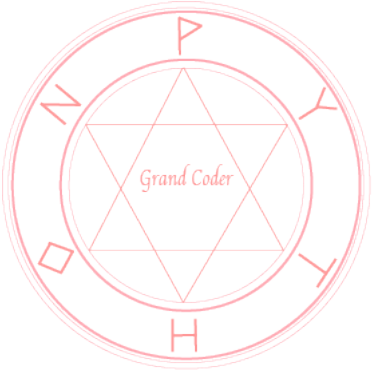
怎麼做呢? 以小時候罰寫當例子，老師如果叫我們寫十次，你會怎麼寫呢?
你會用三個構成來決定!

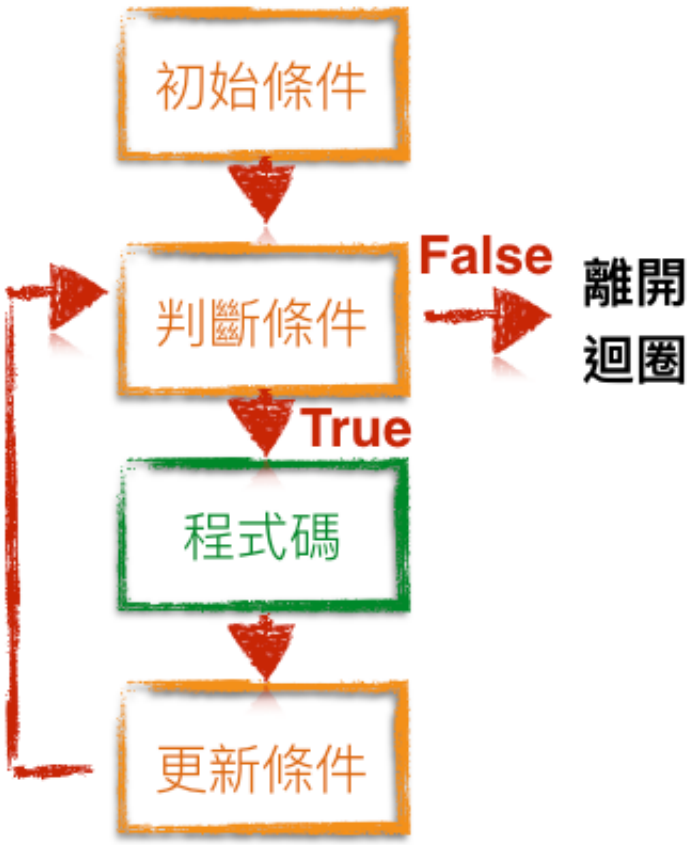
1. 初始條件: 還沒下筆前做一次初始化 (罰寫次數 = 0)
2. 判斷條件: 每次下筆前問自己要不要寫 (罰寫次數 < 10)
3. 更新條件: 每次寫完把次數增加 (罰寫次數 = 罰寫次數 + 1)

我們教各位的第一個迴圈語法是這樣的

```
初始條件
while 判斷條件:
    (程式碼)
    更新條件
```

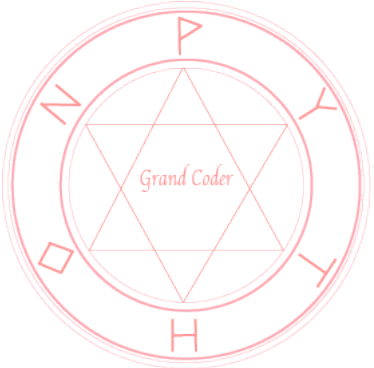
❤ 注意: 更新條件要放在縮排內






```
[程式]: # 初始條件
        times = 0
        # 判斷條件
        while times < 10:
            print("hello")
            # 更新條件
            times = times + 1
```

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```



4.2 迴圈重要概念

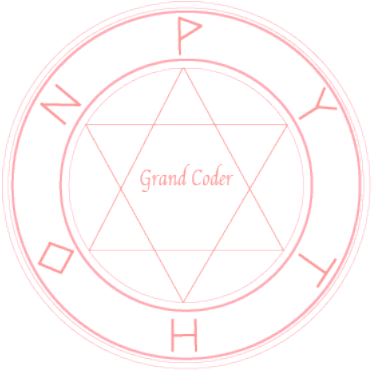
這裡要給各位一個重要的概念，很多初學者會常常在數迴圈到底做幾次
但其實根本不用數！你只要記得永遠這樣寫

```
x = 0
while x < 你要的次數:
    x = x + 1
```

不管你裡面要做什麼，三個條件永遠用 **=0, <, +1** 來做，因為這樣只要看 **<** 後面的數字就知道幾次了

那你會問我，如果我想在裡面做什麼難道這三個都不會變嗎?
是的!『做幾次』和『做什麼』你一定要分開，我做兩個例子給各位看!

```
[程式]: # 我現在想從 10 印到 1, times=0~9, 我就用 times 改出來
        times = 0
        while times < 10:
            print(10 - times, "hello")
            times = times + 1

10 hello
9 hello
8 hello
7 hello
6 hello
5 hello
4 hello
3 hello
2 hello
1 hello


[程式]: # 我現在想從 2 開始每次跳兩個, 2, 4, 6, ... 十次
        times = 0
        while times < 10:
            print(2 * (times + 1), "hello")
            times = times + 1

2 hello
4 hello
6 hello
8 hello
10 hello
12 hello
14 hello
```



```
16 hello
18 hello
20 hello
```

❤ 你可以看到我從來沒有去改那三個條件!!! 這是程式設計師一定要養成的好習慣，別在小地方浪費自己的時間!

4.3 記憶型迴圈

這裡困擾很多初學者，但我要用一個特別的方法教你! 就是『記憶』的概念!

❤ 問題: 從 1 + 2 + 3 + ... + 10 等於多少

❤ 思考: 其實我們人在做這個問題的時候會在腦裡有一個記憶位置，+1 完後這位置變成 1，再 +2 完這位置變成 3，最後把位置裡的數拿出來就可以看到答案

[程式]:
```
# 記憶
result = 0

# 做十次
times = 0
while times < 10:
    # 更新記憶
    result = result + (times + 1)
    times = times + 1

# 拿出記憶的答案
print("答案是:", result)
```

```
答案是: 55
```

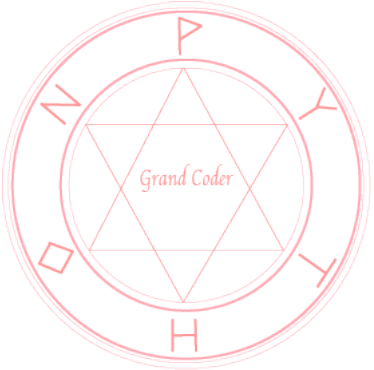
來一個難一點的，『費氏數列』是一個每個數等於前兩個數字相加的數列

費氏數列: 0, 1, 1, 2, 3, 5, 8, 13, ...

❤ 問題: 幫我印出十項費氏數列

❤ 思考: 要印出十項要記憶什麼東西? 我想應該要記得兩個數字就好 (前一個和前兩個)

[程式]:
```
# 第一個記憶區: 前兩個數
lasttwo = 0
# 第二個記憶區: 前一個數
lastone = 0
times = 0
while times < 10:
    # 前兩項直接更新就好
    if times == 0:
```



```
            lasttwo = 0
            ans = 0
        elif times == 1:
            lastone = 1
            ans = 1
        # 前兩項以後就是把記憶區的東西拿出來 +
        # 並且更新記憶區
        else:
            ans = lasttwo + lastone
            lasttwo = lastone
            lastone = ans
        print("第", times + 1, "項:", ans)
        times = times + 1
```

```
第 1 項: 0
第 2 項: 1
第 3 項: 1
第 4 項: 2
第 5 項: 3
第 6 項: 5
第 7 項: 8
第 8 項: 13
第 9 項: 21
第 10 項: 34
```

❤ 記憶區的概念非常重要，大家一定要在做迴圈前問自己，這個迴圈應該要記憶哪些東西!

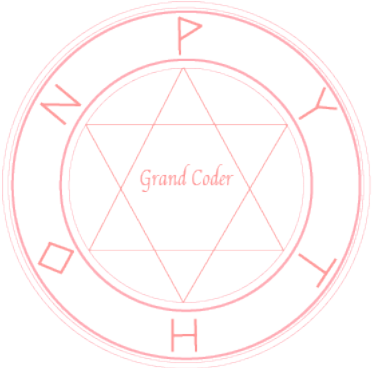
4.4 不固定次數的迴圈

還有另外一種的迴圈，不是叫你固定做幾次的，而是做到滿足某個條件為止就好
這時候你還是要數次數，不過要稍微修改判斷條件

❤ 問題：1 + 2 + 3 + … + x > 50，這個 x 是多少

❤ 思考：> 50 停止，代表 <= 50 就要做下去

```
[程式]: times = 0
        # 一個記憶區
        result = 0
        end = 50
        # 加到 > 50 才停, 你最後加的數字是多少
        while result <= end:
            result = result + (times + 1)
            times = times + 1
        print("最後加的是", times)
```



```
最後加的是 10
```

4.5 無窮迴圈

有的時候，我們會發現判斷條件很難寫，譬如下面的我們一直一直讓使用者輸入，直到使用者輸入 exit

這時候我們會發現 while 後面很難寫

這時候我們就可以先別寫判斷條件，讓他永遠做下去 (while True)，在另外尋找個好地方做停止的判斷 (break)

```
while True:
    if 判斷條件:
        break
```

```
[程式]: while True:
            s = input("輸入 exit 停止:")
            if s == "exit":
                print("輸入 exit, 停止")
                break
            else:
                print("繼續, 輸入的是:", s)
```

```
輸入 exit 停止:hello
繼續, 輸入的是: hello
輸入 exit 停止:a
繼續, 輸入的是: a
輸入 exit 停止:1234
繼續, 輸入的是: 1234
輸入 exit 停止:exit
輸入 exit, 停止
```

4.6 (實戰演練) 終極密碼

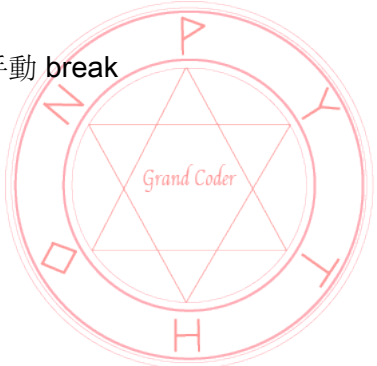
終極密碼是一個讓對方猜數字，然後我們會把範圍縮小讓他繼續猜的遊戲

我們用剛剛學到的東西來做一個終極密碼試試看吧!

❤ 終極密碼需要三個記憶區: 最低，最高，正確答案

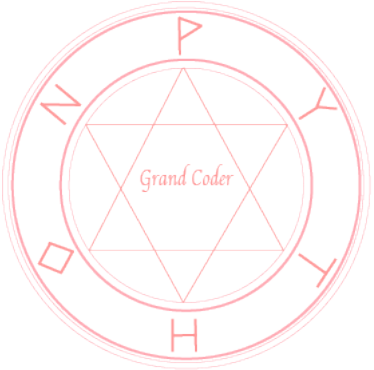
❤ 終極密碼的 while 後面條件很難寫，所以我們用無窮迴圈 + 手動 break

```
[程式]: import random
        low = 1
        high = 100
```



```
    ans = random.randint(low, high)
    # 額外記憶區: 記錄猜幾次
    times = 0
    while True:
        # 還記得字串不能跟數字做運算吧? 所以要先把 low, hight 轉成文字再串連
        msg = "請輸入 [" + str(low) + "-" + str(high) + "]"
        guess = int(input(msg))
        # 次數 + 1
        times = times + 1
        if guess == ans:
            print("恭喜! 猜對了!")
            break
        elif guess < ans:
            # 猜的比答案小, 更新 low
            low = guess
        else:
            # 猜的比答案大, 更新 high
            high = guess
    print("猜的次數", times)
```

```
請輸入 [1-100]50
請輸入 [1-50]25
請輸入 [25-50]38
恭喜! 猜對了!
猜的次數 3
```



5 群集: List

接下來我們要介紹一些特別的資料，這些特別的資料類型有個共同點，就是他們是『一群』資料

5.1 List

我們要介紹的第一種『一群』資料是所謂的『排隊』或者你可以說是『清單』，代表符號是 []

```
name_list = ["Elwing", "Bob", "Carol"]
```

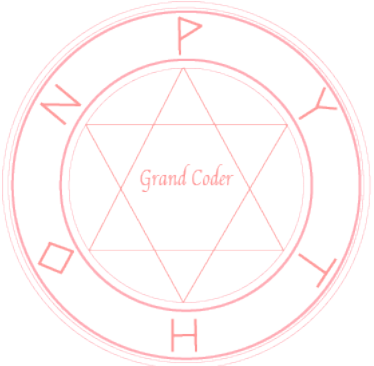
他有一個重要的特性

❤ 順序性：加進去的順序不會被輕易地改變，並且我們可以使用 [座號] 來取得對應的東西

那請記得使用的好習慣

❤ 好習慣：List 使用的時候請放入『同類型』的資料，因為我們不希望在走過 List 的時候還要判斷到底走到的東西是什麼類型

```
[程式]: name_list = ["Elwing", "Bob", "Carol"]
        print(name_list)

        # 不對: name_list + "Amy" -> 不同類型不能運算
        name_list = name_list + ["Amy", "Elwing"]
        print("List 串連:", name_list)
        print("長度:", len(name_list))

        # 拿取第幾個跟字串一樣, 使用 [座號], 拿出來不是 list
        print("第 2 個:", name_list[1])
        # 一樣支持拿一段, 不過拿一段拿到的是 list 喲
        print("第 2 個 - 第 5 個:", name_list[1:5])

        # 支援刪除操作
```



```
        del name_list[1]
        print("刪除本來的第 2 個:", name_list)
```

```
['Elwing', 'Bob', 'Carol']
List 串連: ['Elwing', 'Bob', 'Carol', 'Amy', 'Elwing']
長度: 5
第 2 個: Bob
第 2 個 - 第 5 個: ['Bob', 'Carol', 'Amy', 'Elwing']
刪除本來的第 2 個: ['Elwing', 'Carol', 'Amy', 'Elwing']
```

5.2 for…in 迴圈

這是我們的『群集』最重要的一個語法，我們 Python 的 for 也是專門為了群集而存在的一種迴圈

for…in 保證了兩件事: (1) 每次會拿群集裡面的一個東西給你 (2) 每個都會拿一次給你

語法是這樣的!

```
for 暫時名字 in 群集:
    操作暫時名字
```

這裡要注意的是那所謂的『暫時名字』，因為你如果不給這名字，裡面會不知道怎麼操作這個拿出來的東西

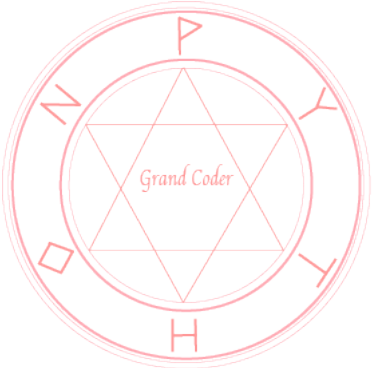
所以我們會暫時名字 = 這次拿出來的東西

```
[程式]: name_list = ["Elwing", "Bob", "Carol"]

        # 走過裡面的所有東西並且印出來
        # 這裡我們用 n 當暫時名字, 你可以用任何名字取代
        for n in name_list:
            # 每一次 n 背後都是不同東西
            print("-", n)
```

```
- Elwing
- Bob
- Carol
```

之前教你數固定次數，我們都用

```
i = 0
while i < 次數:
    i = i + 1
```



但事實上，如果我們能有一個 [0, 1, 2, …, 次數-1] 的 List，我們就可以用這簡單的 for…in 語法達成一樣的事

range 會幫我們產生這個 List，但是記得，range 的第二個參數是不包括的，要多 +1

```
for 暫時名字 in range(0, 次數):
    操作暫時名字
```

以後如果遇到數次數，就別再用 while 了，用 for…in + range 來達成

```
[程式]: # 記憶區還是要準備
        result = 0
        # range(0, 10) -> [0, 1, 2, 3,..., 9]
        for i in range(0, 10):
            result = result + (i + 1)
        print("1 + ... + 10 答案:", result)

1 + … + 10 答案: 55


[程式]: result = 0
        # 如果從 0 開始, 第一個參數可以省略
        for i in range(100):
            result = result + (i + 1)
        print("1 + ... + 100 答案:", result)

1 + … + 100 答案: 5050
```

5.3 List 專屬技能

List 一樣擁有一些好用的專屬技能，一樣可以參考https://www.tutorialspoint.com/python/python_lists.htm

介紹幾個我覺得非常好用的專屬技能

5.3.1 兩種專屬技能

記得我們以前在使用字串的專屬技能的時候，我們發現舊的沒有改，而是回傳一個新的複製品

但是 List 就不一樣了，你會發現他是直接改舊的 List，不回傳任何東西，因為產生一個新的隊伍是有點怪的

這是 Python 的一個設計哲學，通常 Python 的專屬技能只會屬於上面所說的兩種中的其中一種，而不會有混合的狀況

這裡一定要學會分辨

❤ 專屬技能 1: 不改舊的，而是回傳一個新的複製品, 字串的專屬技能屬於這類

❤ 專屬技能 2: 直接改舊的，不回傳任何東西, List 的專屬技能屬於這類

語法上有什麼差異呢? 我們看下去

```
[程式]: name_list = ["Elwing", "Bob", "Carol"]
        print("原 list:", name_list)
        # 我們假裝不知道, 一樣用個名字去接他
        # append 是把東西插入到最後的意思
        # 如果只要插入一個, 我就會用 append
        # 兩個 list 我會用 +
        b = name_list.append("Elwing")
        print(name_list)
        print(b)

原 list: ['Elwing', 'Bob', 'Carol']
['Elwing', 'Bob', 'Carol', 'Elwing']
None
```

你會發現，b 這名字接到的東西是一個叫做 None 的資料 (叫做沒有資料的資料)

這時候要問你一個問題了

```
name_list = name_list.append("Elwing")
```

你如果寫下這個跟我們以前在使用字串一樣的方式，會發生什麼事呢?

恭喜你，你親手的把你的 name_list 變成什麼都沒有 (None)，你現在已經沒有任何排隊了

所以第二種專屬技能千萬不能寫出上面的語法

那如何分辨兩種呢?

❤ 懶人分辨法：跟我們上面一樣，用個名字 (但不要是原本的) 去接他，如果接到 None，毫無疑問是第二種

❤ 勤勞分辨法：看一下文件，如果如下面所說，就是第二種

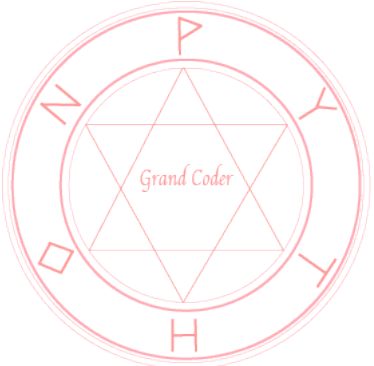
Return Value

This method does not return any value but updates existing list.

圖: 文件上告訴你 return nothing

我們再來試幾個

```
[程式]: name_list = ["Elwing", "Bob", "Carol"]
        print("原 list", name_list)
        # 既然已經知道直接改, 不回傳, 就不接了
        # insert 是插隊的意思
```



```
        # 參數 1: 插入位置, 參數 2: 插入東西
        name_list.insert(2, "Bob")
        print("插隊:", name_list)
        # remove 比較特別, 是 remove 掉第一個 match 的東西
        name_list.remove("Bob")
        print("移除第一個 match:", name_list)
        # reverse 倒過來的意思
        name_list.reverse()
        print("反轉:", name_list)
```

```
原 list ['Elwing', 'Bob', 'Carol']
插隊: ['Elwing', 'Bob', 'Bob', 'Carol']
移除第一個 match: ['Elwing', 'Bob', 'Carol']
反轉: ['Carol', 'Bob', 'Elwing']
```

5.4 (實戰演練) 三門問題

『三門問題』是一個很有趣的生活的問題，『決勝二十一點』這部電影也把它拿來當作課堂的一個問題

圖 (參考 wiki): 決勝二十一點電影海報

❤ 問題描述：今天你參加一個猜謎節目，你前面有三道門，你知道有一道門裡面有『跑車』，另外兩道門則是安慰獎『羊』，你在選了一道門以後，主持人這時候做了一個出人意表的動作，它打開剩餘兩道門的其中之一，並且跟你說了：『今天 special，我告訴妳這道門背後是一隻羊，你要不要換一道門?』這時候很多人覺得是一個心理學問題，其實不是，這是一個數學問題!

❤ 答案：一定要換，因為不換的時候，你贏的機率只有 33.3%($\frac{1}{3}$)，但是如果換的時候贏的機率是 66.6%($\frac{2}{3}$)

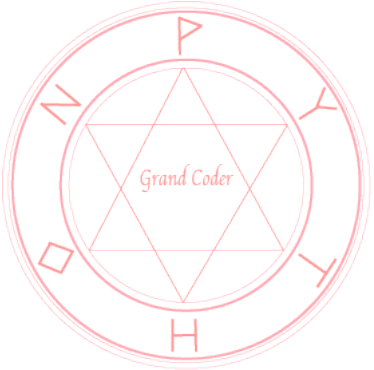
❤ 為什麼呢? 我們分成兩個 case 來討論

1. 不換: 那就是跟一開始一樣，贏機率 $\frac{1}{3}$，輸機率 $\frac{2}{3}$
2. 換: 我們是在討論換的 case！所以你一開始選的門一定不是最後的選擇，而是除了『一開始的門』,『主持人開的門』以外的『僅剩的門』，也就是說『謹慎的門是車』就贏了，那就意味著一開始你選的是『羊』，機率 $\frac{2}{3}$，『謹慎的門是羊』就贏了，那就意味著一開始你選的是『車』，機率 $\frac{1}{3}$，因此贏機率 $\frac{2}{3}$，輸機率 $\frac{1}{3}$

不相信嗎？我們用我們剛學到的 List 寫一個程式來試試看！看看機率是不是真的是這樣！

[程式]:
```
import random
# 準備記憶區記錄書贏次數
win = 0
lose = 0
for times in range(0, 100):
    # 準備三門, 隨機的讓車插隊進兩隻羊之間
    doors = ["羊", "羊"]
    c = random.randint(0, 2)
    doors.insert(c, "車")
    # 讓參賽者挑一個門
    # 因為參賽者選的一定不是最後, 所以我乾脆的刪除
    # 利用 del 刪除參賽者選的位置
    c = random.randint(0, 2)
    del doors[c]
    # 主持人開一隻羊出來
    # 主持人選的也不是最後, 乾脆的刪除
    # 利用 remove 刪除第一隻羊
    doors.remove("羊")
    # 確定一下到底是輸還贏
    # 要注意 doors 雖然只剩一個, 但還是一個隊伍, 所以要拿出來
    if doors[0] == "車":
        win = win + 1
    else:
        lose = lose + 1
print("總共:", win + lose, "次")
print("贏次數:", win, "次")
print("輸次數:", lose, "次")
total = win + lose
print("贏機率:", win / total * 100, "%")
print("輸機率:", lose / total * 100, "%")
```

```
總共: 100 次
贏次數: 67 次
輸次數: 33 次
贏機率: 67.0 %
輸機率: 33.0 %
```



好像不太準? 沒關係，程式最擅長的就是做重複的事，我們把次數提高，機率就會變得精確了

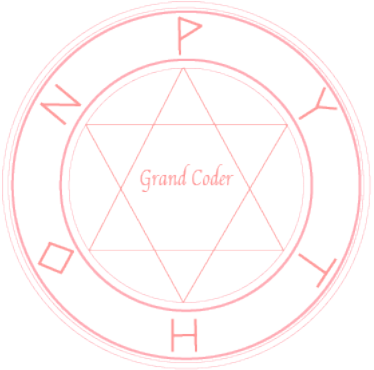
```
[程式]: import random
        # 準備記憶區記錄書贏次數
        win = 0
        lose = 0
        for times in range(0, 100000):
            # 準備三門, 隨機的讓車插隊進兩隻羊之間
            doors = ["羊", "羊"]
            c = random.randint(0, 2)
            doors.insert(c, "車")
            # 讓參賽者挑一個門
            # 因為參賽者選的一定不是最後, 所以我乾脆的刪除
            # 利用 del 刪除參賽者選的位置
            c = random.randint(0, 2)
            del doors[c]
            # 主持人開一隻羊出來
            # 主持人選的也不是最後, 乾脆的刪除
            # 利用 remove 刪除第一隻羊
            doors.remove("羊")
            # 確定一下到底是輸還贏
            # 要注意 doors 雖然只剩一個, 但還是一個隊伍, 所以要拿出來
            if doors[0] == "車":
                win = win + 1
            else:
                lose = lose + 1
        print("總共:", win + lose, "次")
        print("贏次數:", win, "次")
        print("輸次數:", lose, "次")
        total = win + lose
        print("贏機率:", win / total * 100, "%")
        print("輸機率:", lose / total * 100, "%")
```

```
總共: 100000 次
贏次數: 66521 次
輸次數: 33479 次
贏機率: 66.521 %
輸機率: 33.479 %
```

5.5 (進階演練) n 門問題

那你就會問老師，四道門還是要換嗎? 當然!
一樣分兩個 case

1. 不換: 贏機率 $\frac{1}{4} = 25\%$，輸機率 $\frac{3}{4} = 75\%$



2. 換: 這裡最後會剩下 (4 - 2) = 2 道門，有一道門會贏，還要做出選擇，所以贏機率 = 一開始輸 * 最後的選擇贏，$\frac{3}{4} \times \frac{1}{2} = \frac{3}{8} = 37.5\%$，輸的機率 = 一開始贏 + 一開始輸 * 最後的選擇輸 = $\frac{1}{4} + \frac{3}{4} \times \frac{1}{2} = \frac{5}{8} = 62.5\%$

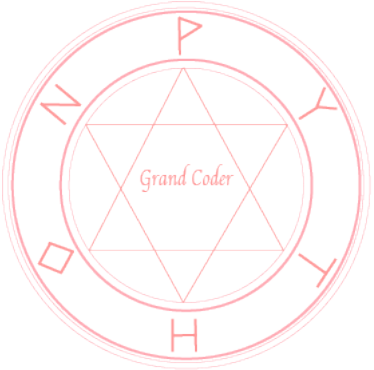
還是比不換多了 12.5% 的機率
雖然機率會不斷被稀釋 (因為最後還要做出選擇)，但是『換』還是會得到『增益』
事實上，如果嚴謹的寫，有 n 道門，換的情況下

1. 贏機率: 一開始輸 * 最後選擇贏 = $\frac{n-1}{n} \times \frac{1}{n-2}$
2. 輸機率: 一開始贏 + 一開始輸 * 最後選擇輸 = $\frac{1}{n} + \frac{n-1}{n} \times \frac{n-3}{n-2}$

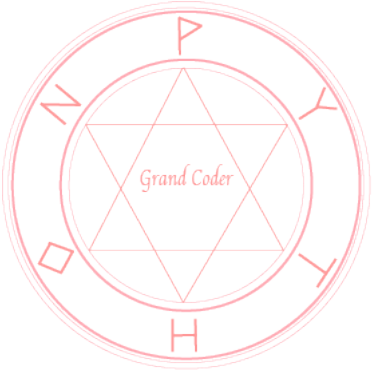
熟悉數學的同學會發現只有在無限多門的時候，才沒增益，因為無限多門贏機率 = 0，輸機率 = 1

但世界上沒有所謂無限多門，所以請參加節目的時候務必換一下!

[程式]:
```
import random
win = 0
lose = 0
door_number = int(input("你要幾門?"))
for times in range(0, 100000):
    # 比較進階的語法, 可以直接用乘法做出很多隻羊
    # 羊數目 = 門 - 1
    doors = ["羊"] * (door_number - 1)
    # 因為 randint 包含, 號碼牌最多只到長度 - 1
    c = random.randint(0, len(doors) - 1)
    doors.insert(c, "車")
    # 讓參賽者挑一個門
    c = random.randint(0, len(doors) - 1)
    del doors[c]
    # 主持人開一隻羊出來
    doors.remove("羊")
    # 要加一個動作, 讓他選剩下的門
    c = random.randint(0, len(doors) - 1)
    if doors[c] == "車":
        win = win + 1
    else:
        lose = lose + 1
print("總共:", win + lose, "次")
print("贏次數:", win)
print("輸次數:", lose)
total = win + lose
print("贏機率:", win / total * 100, "%")
print("輸機率:", lose / total * 100, "%")
```



```
你要幾門?4
總共: 100000 次
贏次數: 37510
輸次數: 62490
贏機率: 37.51 %
輸機率: 62.49 %
```



6 群集: Dictionary

6.1 Dictionary 介紹

Dictionary 跟前面講的 List 和 Set 有決定性的『使用習慣上的不同』

之前的 List 和 Set 我們在使用上都會放入同類型的東西，因為本來就該是『同類型的清單』和『同類型的集合』，而且這樣在『走過』操作的時候才不會需要判斷類型

但是字典 (Dictionary)，是拿來聚集『不同面向』，變成一個複雜資料的，語法用 {} 和: 構成

```
person1 = {"name":"Elwing", "height":175, "weight":75}
```

簡單來說，平常我們習慣這樣寫

```
name = "Elwing"
height = 175
weight = 75
```

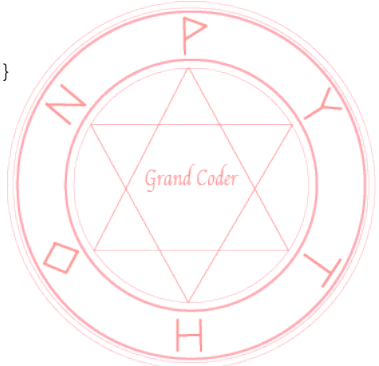
現在只是把這些名字和資料的對應聚集成一個『字典』而已，操作方式跟我們平常操作名字一模一樣 (見下面程式)

要注意一下，雖然和 set 都是用 {} 表示，但 set 不會出現:

6.2 Dictionary 基本操作

Dictionary 所有的操作都跟普通操作一樣，只是增加了字典名 [ ] 而已，我們一起從下面的例子看一下基本操作

```
[程式]: person1 = {"name":"Elwing", "height":175, "weight":75}
        print("原字典:", person1)

        # 拿取: 平常我們會直接 print(name)
        # 現在要用 person1(字典名)["name"]
```



```
        print("拿取:", person1["name"])

        # 增加: 平常就直接 gender = "M"
        person1["gender"] = "M"
        print("增加:", person1)

        # 改變: 平常做 height = 180
        person1["height"] = 180
        print("改變:", person1)

        # 拿取並更新: 平常 height = height + 5
        person1["height"] = person1["height"] + 5
        print("更新:", person1)

        # 這裡有一個之前不會用的操作: 刪除
        del person1["gender"]
        print("刪除:", person1)
```

```
原字典: {'name': 'Elwing', 'height': 175, 'weight': 75}
拿取: Elwing
增加: {'name': 'Elwing', 'height': 175, 'weight': 75, 'gender': 'M'}
改變: {'name': 'Elwing', 'height': 180, 'weight': 75, 'gender': 'M'}
更新: {'name': 'Elwing', 'height': 185, 'weight': 75, 'gender': 'M'}
刪除: {'name': 'Elwing', 'height': 185, 'weight': 75}
```

6.3 for…in 操作

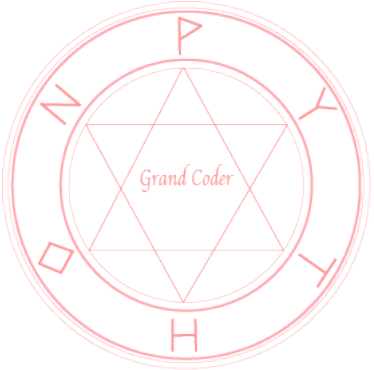
Dictionary 的 for…in 操作比較特別一點，由於每組東西都是兩個一起

但是我們在 for…in 間的暫時名字只能放一個，問題是名字: 資料到底要給你前面的還是後面的呢?

很明顯給你『名字』，你就可以利用 **[ ]** 的語法得到『資料』

```
[程式]: person1 = {"name":"Elwing", "height":175, "weight":75}
        print("原字典:", person1)
        # 我們常稱字典的一組叫做 key:value 的 pair, 所以我就取名叫 key
        for key in person1:
            print("名字:", key)
            print("對應資料:", person1[key])
```

```
原字典: {'name': 'Elwing', 'height': 175, 'weight': 75}
名字: name
對應資料: Elwing
名字: height
對應資料: 175
```



```
名字: weight
對應資料: 75
```

(進階) 但是如果你就是要兩個一起接也是可以，那就要使用我們後面教你的 tuple

items 這技能會回傳一個 list，裡面每個東西都是一個 tuple，你可以準備兩個名字直接去接 tuple 裡的兩個東西

```
[程式]: person1 = {"name":"Elwing", "height":175, "weight":75}
        print("使用 items:", person1.items())
        for key, value in person1.items():
            print("名字:", key)
            print("資料:", value)

使用 items: dict_items([('name', 'Elwing'), ('height', 175), ('weight', 75)])
名字: name
資料: Elwing
名字: height
資料: 175
名字: weight
資料: 75
```

6.4 (實戰演練) 統計文章字數

我們來練習一下統計文章字數，但在這之前我們要先瞭解一下編碼

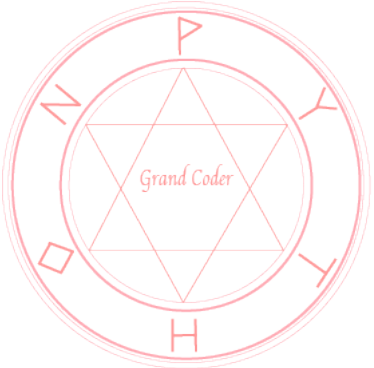
編碼：把你看到的文字轉換成電腦的 01

解碼：把電腦儲存的 01 換成你看到的文字

你可以看到『我』用不同的編碼轉換後變成不同的 01(4 個 01 組合成 16 進位，\x 指的是十六進位)

再用同樣的編碼轉換回來就可以看到原來的文字

```
[程式]: print("utf-8 編碼:", "我".encode(encoding='utf-8'))
        print("Big5 編碼:", "我".encode(encoding='big5'))
        # 在字串前方加入 b 指的是原始 01 的意思, \x 指的是四個位元一組, 以十六進位表示
        print("utf-8 解碼:", b'\xe6\x88\x91'.decode("utf-8"))
        print("Big5 解碼:", b'\xa7\xda'.decode("big5"))

utf-8 編碼: b'\xe6\x88\x91'
Big5 編碼: b'\xa7\xda'
utf-8 解碼: 我
Big5 解碼: 我
```



什麼時候會發生亂碼呢? 當你的編碼和你的解碼不一樣的時候就會發生亂碼，所以你一定要確實知道你使用什麼編碼!

那究竟我們的作業系統預設是用什麼編碼在處理我們的文件呢?

Windows 系統: ANSI，ANSI 是一套根據不同地區制定的編碼標準，對於繁體那就是 Big-5

MAC(UNIX) 系統: utf-8，UNIX 系統使用最常用的 utf-8 來作為編碼標準

但請記得下面三個重點

❤ 請盡量使用 utf-8 來作為主要編碼

❤ 不管讀檔案和寫檔案的時候都主動寫下編碼，才不會發生編碼不如預期的情況

❤ 請盡量把所有要處理的檔案放在專案底下，不然你寫了一個超厲害的程式讀取 C 槽的檔案，卻發現你的使用者沒有 C 槽，就很尷尬了

如果你使用 Window 預設的『記事本』來開啟純文字檔案，會使用 ANSI 來編碼，你必須使用『另存新檔』來改變編碼

我會建議如果使用 PyCharm 的話，可以使用 File -> New -> File 來開啟檔案 (PyCharm 一律使用 utf-8 編碼)

在我們 python 內，你開啟檔案的時候一定要指定開啟的模式:『r』: 閱讀 only，『w』: 寫入 (覆寫)only，『a』: 寫入 (接著結尾寫)

讀取純文字檔案的基本語法如下

```
f = open(檔案名, "r", encoding="utf-8")
# 讀取使用 read, article 是我習慣名字, 你可以任意取
article = f.read()
f.close()
```

寫入純文字檔案的基本語法如下

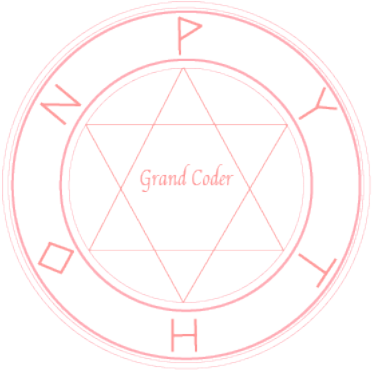
```
f = open(檔案名, "w", encoding="utf-8")
# 改成用參數傳入你想要寫入的東西
f.write(" 你想要寫入的字串")
f.close()
```

進階語法 (我在這裡先不使用)

with...as: 好處是無論發生什麼事都會自動幫你把檔案 close，你不用擔心沒有 close 的情況

```
# 以 read 為例, 你會發現 close 被藏起來, 無論如何都會執行
with open(檔案名, "r", encoding="utf-8") as f:
    article = f.read()
```

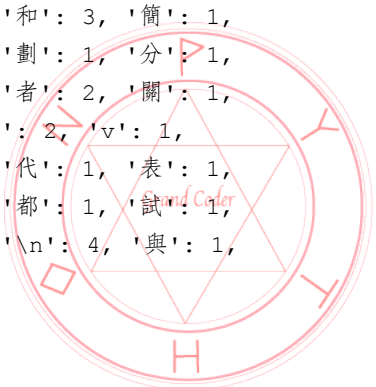
[程式]:
```
# 你可以使用 PyCharm 開啟這個 a.txt
# 並且去網路上尋找隨意一篇中文文章
# 我擷取的是 wiki 的 python 介紹
```



```
f = open("a.txt", "r", encoding="utf-8")
article = f.read()
f.close()
print(article)
# 接著我們準備一個空的字典來準備做出例如 {"A":3, "B":5} 的次數字典
# 空字典直接使用 {} 就可以創造
result = {}
# 字串也可以使用 for...in 走過
for c in article:
    # 這裡要注意一下, 會有兩個不同 case
    # 1. 第一次遇到的字: 如果想直接拿取 result[c] 會發生 KeyError
    # 2. 第二次遇到的字: 可以直接拿取
    # 因此我用 in 來判斷是不是有在裡面
    if c in result:
        # 第二次以上遇到, 把次數拿出來 +1 並且設定回去
        result[c] = result[c] + 1
    else:
        # 第一次遇到, 直接新增在字典裡並且把次數設定成 1
        result[c] = 1
# 注意縮排, 是整個做完才印出
print(result)
```

Python 是一種廣泛使用的高階程式語言，屬於通用型程式語言，由吉多·范羅蘇姆創造，第一版釋出於 1991 年。可以視之為一種改良（加入一些其他程式語言的優點，如物件導向）的 LISP。作為一種直譯語言，Python 的設計哲學強調程式碼的可讀性和簡潔的語法（尤其是使用空格縮排劃分程式碼塊，而非使用大括號或者關鍵詞）。相比於 C++ 或 Java，Python 讓開發者能夠用更少的代碼表達想法。不管是小型還是大型程式，該語言都試圖讓程式的結構清晰明了。 與 Scheme、Ruby、Perl、Tcl 等動態型別程式語言一樣，Python 擁有動態型別系統和垃圾回收功能，能夠自動管理記憶體使用，並且支援多種編程範式，包括物件導向、命令式、函數式和程序式編程。其本身擁有一個巨大而廣泛的標準庫。 Python 直譯器本身幾乎可以在所有的作業系統中執行。Python 的正式直譯器 CPython 是用 C 語言編寫的、是一個由社群驅動的自由軟體，目前由 Python 軟體基金會管理。

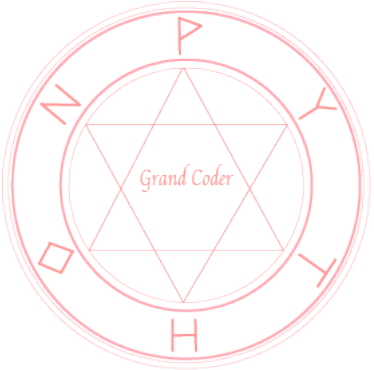
```
{'P': 10, 'y': 9, 't': 8, 'h': 9, 'o': 8, 'n': 8, '是': 6, '一': 8, '種': 4,
'廣': 2, '泛': 2, '使': 4, '用': 7, '的': 13, '高': 1, '階': 1, '程': 11, '式': 13,
'語': 8, '言': 7, ',': 13, '屬': 1, '於': 3, '通': 1, '型': 5, '由': 4, '吉': 1,
'多': 2, '·': 1, '范': 1, '羅': 1, '蘇': 1, '姆': 1, '創': 1, '造': 1, '第': 1,
'版': 1, '釋': 1, '出': 1, '1': 2, '9': 2, '年': 1, '。': 9, '可': 3, '以': 2,
'視': 1, '之': 1, '為': 2, '改': 1, '良': 1, ' (': 2, '加': 1, '入': 1, '些': 1,
'其': 3, '他': 1, '優': 1, '點': 1, '如': 1, '物': 2, '件': 2, '導': 2, '向': 2,
') ': 2, 'L': 1, 'I': 1, 'S': 2, '作': 2, '直': 3, '譯': 3, '設': 1, '計': 1,
'哲': 1, '學': 1, '強': 1, '調': 1, '碼': 3, '讀': 1, '性': 1, '和': 3, '簡': 1,
'潔': 1, '法': 2, '尤': 1, '空': 1, '格': 1, '縮': 1, '排': 1, '劃': 1, '分': 1,
'塊': 1, '而': 2, '非': 1, '大': 3, '括': 2, '號': 1, '或': 2, '者': 2, '關': 1,
'鍵': 1, '詞': 1, '相': 1, '比': 1, 'C': 3, '+': 2, 'J': 1, 'a': 2, 'v': 1,
'讓': 2, '開': 1, '發': 1, '能': 3, '夠': 2, '更': 1, '少': 1, '代': 1, '表': 1,
'達': 1, '想': 1, '不': 1, '管': 3, '小': 1, '還': 1, '該': 1, '都': 1, '試': 1,
'圖': 1, '結': 1, '構': 1, '清': 1, '晰': 1, '明': 1, '了': 1, '\n': 4, '與': 1,
```



```
'c': 2, 'e': 3, 'm': 1, '、': 6, 'R': 1, 'u': 1, 'b': 1, 'r': 1, 'l': 2,
'T': 1, '等': 1, '動': 4, '態': 2, '別': 2, '樣': 1, '擁': 2, '有': 3, '系': 2,
'統': 2, '垃': 1, '圾': 1, '回': 1, '收': 1, '功': 1, '自': 2, '理': 2, '記': 1,
'憶': 1, '體': 3, '並': 1, '且': 1, '支': 1, '援': 1, '編': 3, '範': 1, '包': 1,
'命': 1, '令': 1, '函': 1, '數': 1, '序': 1, '本': 2, '身': 2, '個': 2, '巨': 1,
'標': 1, '準': 1, '庫': 1, ' ': 1, '器': 2, '幾': 1, '乎': 1, '在': 1, '所': 1,
'業': 1, '中': 1, '執': 1, '行': 1, '正': 1, '寫': 1, '社': 1, '群': 1, '驅': 1,
'軟': 2, '目': 1, '前': 1, '基': 1, '金': 1, '會': 1}
```

6.5 結語

統計次數有什麼用呢? 如果單純統計『字』的次數是不太有用的，但是如果能統計『詞』出現的次數就有用多了，各位可以翻到後面的『自然語言處理 - 關鍵詞萃取』看看我們基本的 tf-idf 理論!



7 群集: Set

7.1 Set 介紹

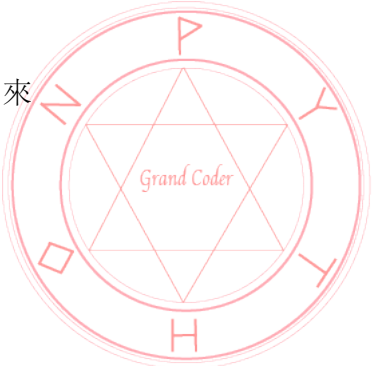
我們在日常中，常常遇到一種群，譬如：樂透的七個數字，登入過網站的所有 ip...等等
你會發現這兩個例子群裡面的東西都是『不重複的』
Set 跟清單一樣使用時機適用於一群『不重複』東西的集合，代表符號是大括號

```
name_set = {" 周", " 王", " 林"}
```

不過不一樣的是，Set 具有兩個很重要的特性
❤ 不重複性：裡面的東西不管你怎麼加，他一定只會幫你保持一個
❤ 無順序性：跟 List 不一樣的是，Set 裡面的東西是沒有順序性的 (如果有兩個重複，幫你保留一個的時候拿哪個順序?)，所以你會發現加進去的順序跟你印出來的順序不一定一樣，這是非常正常的!
如果用白話文來說，Set 的群集型態就像『裡面不會有重複東西的購物袋』，購物袋裡面東西可沒有排成一個隊伍
用比較嚴謹的方式來說，Set 就像數學上所說的『集合』

```
[程式]: name_set = {"周", "王", "林"}
        print(name_set)
        print("長度:", len(name_set))

{'林', '王', '周'}
長度: 3
```

Set 沒有所謂的『+』，而有的是 1. 『&』交集: 兩的都有的列出來
2. 『|』聯集: 一個有列出來
3. 『-』差集: 前面有但後面沒有的列出來



```
[程式]: name_set_1 = {"周", "王", "林"}
        name_set_2 = {"王", "何", "謝"}
        print("set1:", name_set_1)
        print("set2:", name_set_2)

        print("交集:", name_set_1 & name_set_2)
        print("聯集:", name_set_1 | name_set_2)
        print("差集:", name_set_1 - name_set_2)
```

```
set1: {'林', '王', '周'}
set2: {'何', '王', '謝'}
交集: {'王'}
聯集: {'周', '何', '林', '王', '謝'}
差集: {'林', '周'}
```

7.2 Set 專屬技能

Set 也有一系列的專屬技能，不過這裡要注意一下 Set 專屬技能剛好有兩種不同類型的專屬技能

❤ 第一種專屬技能: 回傳新的，不改舊的，union(聯集的專屬技能版)，intersection(交集的專屬技能版)，difference(差集的專屬技能版)

❤ 第二種專屬技能: 直接改舊的，不回傳新的

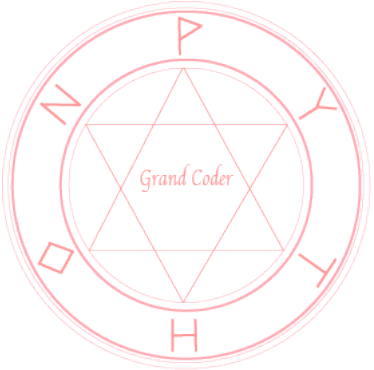
怎麼分辨請詳見我們的 List，一樣可以用個名字去接，接到 None 代表第二種，接到不是 None 代表第一種

❤ 回憶一下: 第二種專屬技能不能寫的 -> a = a. 專屬技能

先來看看屬於第二種的專屬技能

```
[程式]: # 屬於第二種的
        name_set = {"周", "王", "林"}
        print("原本:", name_set)
        name_set.add("謝")
        print("add 後:", name_set)
        # discard 行為: 有在裡面就刪除, 沒有就不理會
        name_set.discard("王")
        print("有就刪除:", name_set)
        name_set.discard("何")
        print("沒有就不做事:", name_set)
```

```
原本: {'林', '王', '周'}
add 後: {'林', '王', '周', '謝'}
有就刪除: {'林', '周', '謝'}
沒有就不做事: {'林', '周', '謝'}
```



事實上，還有另外一種刪除，這種刪除會在沒有的時候通知你

我戲稱他叫『勤勞的員工』，會用紅字通知你，但由於我還沒教你處理這種『警告』的語法，所以我先放在這給各位參考

後續你學會語法再回頭看

```
[程式]: name_set = {"周", "王", "林"}
        try:
            name_set.remove("何")
        except KeyError:
            print("[回報] 東西不在裡面, 是不是什麼東西出錯了?")
```

```
[回報] 東西不在裡面, 是不是什麼東西出錯了?
```

再來看看屬於第一種的專屬技能

```
[程式]: name_set = {"周", "王", "林"}
        print("原本:", name_set)
        # 由於是第一種, 原本的不會改, 你如果想要更新必須設定回去
        # union 跟 | 是一樣的效果
        name_set = name_set.union({"何", "林"})
        print("聯集:", name_set)
        # intersection 跟 & 一樣效果
        name_set = name_set.intersection({"王", "何"})
        print("交集:", name_set)
        # difference 跟-一樣效果
        name_set = name_set.difference({"何", "林"})
        print("差集:", name_set)
```

```
原本: {'林', '王', '周'}
聯集: {'周', '何', '林', '王'}
交集: {'何', '王'}
差集: {'王'}
```

7.3 (實戰演練) 樂透

我們從頭開始完成樂透的實驗: 同一期買幾張才能七個數字中 n 個數字

7.3.1 買一張樂透

樂透很明顯是一堆不重複的數字，這時候使用 Set 就可以輕鬆地製作出樂透


```
[程式]: import random
        # Set 記憶區: 用 set() 準備一個空的 set
```

```
        lottery = set()
        # 不確定幾次才能產生不重複七個數字, 用 while
        while len(lottery) < 7:
            lottery.add(random.randint(1, 49))
        print("產生的彩券:", lottery)
```

```
產生的彩券: {13, 19, 21, 24, 25, 27, 29}
```

7.3.2 對獎

對獎要換個想法: 樂透總公司為這期買了一張隨機彩券，只要跟這彩券長得一模一樣就是頭獎

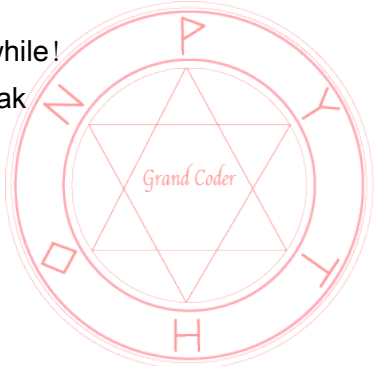
```
[程式]: import random
        # 這期的頭彩彩券
        prize = set()
        while len(prize) < 7:
            prize.add(random.randint(1, 49))
        print("頭獎的彩券:", prize)

        # 你買的彩券
        lottery = set()
        while len(lottery) < 7:
            lottery.add(random.randint(1, 49))
        print("產生的彩券:", lottery)

        # 利用學到的交集做快速的對獎
        # 記得交集是產生一個新的 Set(第一種專屬技能)
        same = prize.intersection(lottery)
        print("中獎數字:", same)
        print("中獎個數:", len(same))
```

```
頭獎的彩券: {32, 1, 36, 9, 47, 18, 23}
產生的彩券: {1, 40, 43, 17, 25, 27, 31}
中獎數字: {1}
中獎個數: 1
```

7.3.3 買到中

一直買下去，買到中為止，這裡由於是不固定次數，所以使用 while!
另外由於 while 後面的條件很難寫，所以使用 while True 和 break

```
[程式]: import random
        end = int(input("中幾個才停?"))
```



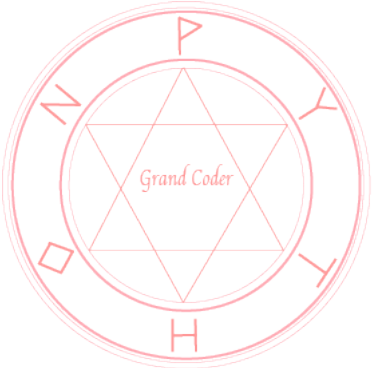
```
        # 這期的頭彩彩券
        prize = set()
        while len(prize) < 7:
            prize.add(random.randint(1, 49))
        print("頭獎的彩券:", prize)

        # 記錄買的彩券數
        times = 0
        # 買到中
        while True:
            # 你買的彩券
            lottery = set()
            while len(lottery) < 7:
                lottery.add(random.randint(1, 49))
            # 張數 + 1
            times = times + 1
            # 利用學到的交集做快速的對獎
            # 記得交集是產生一個新的 Set(第一種專屬技能)
            same = prize.intersection(lottery)
            # 中超過 n 個數字就 break
            if len(same) >= end:
                break
        print("中了的彩券:", lottery)
        print("中了的數字:", same)
        print("買了", times, "張才中了", len(same), "個數字")
```

```
中幾個才停?6
頭獎的彩券: {3, 5, 38, 37, 40, 45, 21}
中了的彩券: {3, 37, 5, 38, 45, 21, 23}
中了的數字: {3, 5, 37, 38, 45, 21}
買了 89365 張才中了 6 個數字
```

7.3.4 (進階) 函數定義

雖然後面我們才會講到函數的定義，不過我們可以先來體驗一下
你會發現我們有一個東西一直在寫，就是產生彩券的這個動作
那你就可以把『重複的程式碼』拿出來定義成一個流程，這就是『函式定義』

```
[程式]: import random
        end = int(input("中幾個才停?"))

        # 定義產生彩券函式
        # 無參數, 回傳一個 set
        def generate_ticket():
```

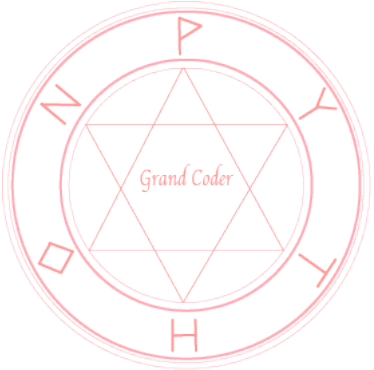


```
        ticket = set()
        while len(ticket) < 7:
            ticket.add(random.randint(1, 49))
        # 回傳產生的 set
        return ticket

    # 這期的頭彩彩券
    prize = generate_ticket()
    print("頭獎的彩券:", prize)

    # 記錄買的彩券數
    times = 0
    # 買到中
    while True:
        # 你買的彩券
        lottery = generate_ticket()
        times = times + 1
        # 利用學到的交集做快速的對獎
        # 記得交集是產生一個新的 Set(第一種專屬技能)
        same = prize.intersection(lottery)
        # 中超過 n 個數字就 break
        if len(same) >= end:
            break
    print("中了的彩券:", lottery)
    print("中了的數字:", same)
    print("買了", times, "張才中了", len(same), "個數字")
```

```
中幾個才停?6
頭獎的彩券: {2, 7, 14, 16, 49, 17, 28}
中了的彩券: {2, 35, 7, 14, 16, 17, 28}
中了的數字: {2, 7, 14, 16, 17, 28}
買了 13301 張才中了 6 個數字
```

7.4 (進階實戰演練) 大樂透

大樂透規則如下：您必須從 01~49 中任選 6 個號碼進行投注。開獎時，開獎單位將隨機開出六個號碼加一個特別號



中獎方式	中獎方式圖示	獎項
與當期六個獎號完全相同者		頭獎
對中當期獎號之任五碼＋特別號		貳獎
對中當期獎號之任五碼		參獎
對中當期獎號之任四碼＋特別號		肆獎
對中當期獎號之任四碼		伍獎 NT$2,000
對中當期獎號之任三碼＋特別號		陸獎 NT$1,000
對中當期獎號之任兩碼＋特別號		柒獎 NT$400
對中當期獎號之任三碼		普獎 NT$400

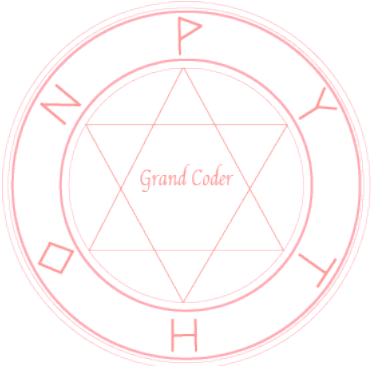
圖: 大樂透中獎規則

要實作這個大樂透，你必須清楚了解大樂透的真實用意

特別號其實就是一個『中獎一半的意思』。所以你可以把正號當成『2』點，特別號當成『1』點

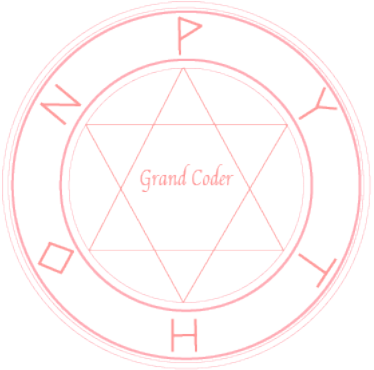
頭獎：2 * 6 點，貳獎：2 * 5 + 1...依序下去

```
[程式]: import random
        c = int(input("請選擇 [0] 頭獎 [1] 貳獎 [2] 參獎 [3] 肆獎:"))
        # 把特別號當作 +1, 正號當 +2, 算出每個獎的點數
        # 頭獎: 12 點 貳獎: 11 點 參獎: 10 點 肆獎: 9 點
        min_point = 12 - c

        trans = ["頭獎", "貳獎", "參獎", "肆獎"]
        print("中了", trans[c], "才停")

        def generate_ticket():
            ticket = set()
            while len(ticket) < 6:
                ticket.add(random.randint(1, 49))
            return ticket

        # 這裡我開始 random 特別號, 直到不是正號裡的數字才停
```



```
    prize = generate_ticket()
    print("頭獎的彩券:", prize)
    while True:
        special = random.randint(1, 49)
        if not special in prize:
            break
    print("特別號:", special)

    times = 0
    while True:
        lottery = generate_ticket()
        times = times + 1
        # 檢查正號: 利用交集檢查中了幾個
        same = lottery.intersection(prize)
        point = len(same) * 2
        # 檢查特別號: 沒中的號碼中可能有特別號
        remain = lottery.difference(prize)
        if special in remain:
            point = point + 1
        if point >= min_point:
            break

    print("中了的彩券:", lottery)
    print("中了的數字:", same)
    print("特別號有無中:", special in remain)
    # 換回 0 - 3
    print("買了", times, "張才中了", trans[12 - point])
```

```
請選擇 [0] 頭獎 [1] 貳獎 [2] 參獎 [3] 肆獎:1
中了 貳獎 才停
頭獎的彩券: {1, 4, 36, 46, 48, 28}
特別號: 20
中了的彩券: {1, 4, 36, 48, 20, 28}
中了的數字: {1, 4, 36, 48, 28}
特別號有無中: True
買了 820215 張才中了 貳獎
```



8 群集: Tuple

8.1 Tuple 介紹

我之所以把 Tuple 放在最後介紹，是因為 Tuple 其實是一個字典的簡化版，字典我們有名字:資料的組合

但是 Tuple 把名字的部分拿掉，只有資料的部分，用於你沒有需要那麼嚴謹的資料定義，只需要使用一兩次的狀況!

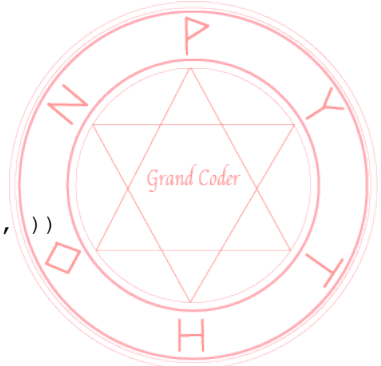
8.2 基本操作

基本語法：用小括號定義 tuple

```
person = ("Elwing", 175, 75)
```

這裡要注意一下，雖然 tuple 的操作跟 List 很像，不過其實它的『本質』跟字典一樣，是拿來組合『不同面向』的資料的

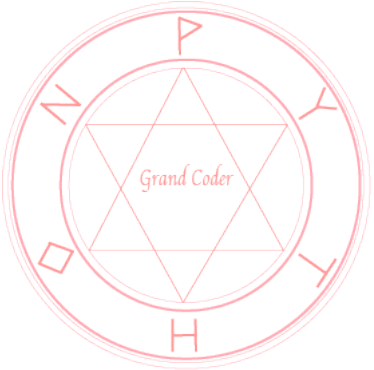
```
[程式]: person = ("Elwing", 175, 75)
        print("原 tuple:", person)

        # 取得 tuple 的某個資料: 使用座號
        print("第一個:", person[0])
        # 反向數: 從-1 開始
        print("最後一個:", person[-1])

        # 某個到某個, : 後面的座號不包括, 要多 +1
        print("第一個 (座號 0) 到第二個 (座號 1):", person[0:2])

        # 加法: 把兩個 tuple 串起來
        print("串連:", person + ("Taipei", "Male"))
        # 就算只串連一個, 也必須是個 tuple, 但不能只寫 (), 要寫 (,)
        print("串連裡面只有一個東西的 tuple:", person + ("Taipei",))
```



```
原 tuple: ('Elwing', 175, 75)
第一個: Elwing
最後一個: 75
第一個 (座號 0) 到第二個 (座號 1): ('Elwing', 175)
串連: ('Elwing', 175, 75, 'Taipei', 'Male')
串連裡面只有一個東西的 tuple: ('Elwing', 175, 75, 'Taipei')
```

tuple 沒有刪除的操作，因為 Python 在設計 tuple 的初衷就是簡單的使用，如果需要複雜的使用可以使用『字典』替代，但如果硬是要做，還是可以透過較為『狡詐』的手段完成

```
[程式]: person = ('Elwing', 175, 75, 'Taipei')
        # : 前面都不寫意味從最前開始, : 後面都不寫意味著到最後
        print("變相刪除 175:", person[:1] + person[2:])

變相刪除 175: ('Elwing', 75, 'Taipei')
```



9 自訂函式

我們用了這麼多次的函式 (ex. print, abs…等等)，卻從來沒有自己定義過函式！但在學會定義函式前，你一定要知道為什麼我們要定義函式

9.1 定義時機

當你發現有幾句程式你一直寫一直寫，那就是你該把它定義成一個『流程』(函式)，改天要用的時候就直接使用『流程』名字就好了！

沒錯，定義函式的時機就是當你『重複書寫』的時候

9.2 使用語法

這裡分兩個地方來看

❤ 傳入資料 (參數): 定義參數比較特別的是其實我們是定義一個『名字』來接傳入的『資料』

❤ 回傳答案 (回傳值): 使用 return 來將答案回傳，要注意一執行到 return，函式即會結束

```
def 函式名 (參數名 1, 參數 2):
    return 回傳資料
```

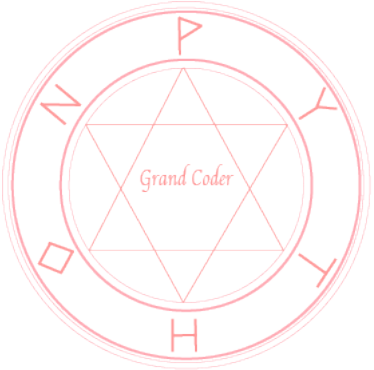
記得冒號和縮排要加上去！因為是有『屬於』的結構關係！

9.3 基本使用

```
[程式]: # 參數: 你不拿個名字 (n1, n2) 來接他, 你沒辦法使用它
        def add(n1, n2):
            return n1 + n2
```

正確的使用方式一定要帶入剛好 2 個參數，多或少都不行

```
[程式]: print(add(3, 5))
```



```
8
```

你會發現，下面這句是可以的，因為我們並沒有限制 n1 和 n2 到底是什麼類型的資料

```
add("hello", "abc")
```

這卻是不行的

```
add("hello", 3)
```

為什麼呢? 因為流程裡面使用的是『+』，運算只能對『同類型』的資料使用
這就是大家常說的『鴨子型別』
❤『非鴨子型別』: 例如 C++ 和 JAVA，你在定義 n1 和 n2 的時候就會確定傳進來的是什麼型態，如以下的例子，如果要我用白話文說，就是你蓋了座『鴨子游泳池』而且有個海關，只有『真正』的鴨子才能進來游泳! 好處是進到你的游泳池的『鴨子』絕對不會溺水，所以以前的程式語言，只要能執行，通常不會有太大的問題!

```
int add(int n1, int n2){
    return n1 + n2;
}
```

❤『鴨子型別』: 如 Python 之類的先進語言，並不會限制傳進來的東西是什麼型態，而是可以執行就執行，不能執行就直接報出錯誤，用白話文說，就是你一樣蓋了座『鴨子游泳池』，但並沒有做出任何限制，都進來，可以游泳的就自在游泳，不能游泳的就自然會『沉溺』(紅字報出錯誤)，好處是語法會很自由，但程式設計師自己一定要知道什麼操作可以，什麼操作不可以

```
[程式]: # 兩個字串是可以相加的!
        print(add("hello", "abc"))
```

```
helloabc
```

9.4 不定參數

9.4.1 單星號不定參數

你一定會好奇，為什麼我們的 print 好像可以帶入很多參數，是怎樣做到的呢? 因為我們上面的基本使用好像是定義幾個，就只能帶入幾個
我們先用一個比較『狡猾』的方法來做到，定義一個參數，來接受一個 list
要注意，下面的使用方式是不行的

```
print(add_multiple(["a", "b", "c"]))
```

因為 result 已經設定為 0，就只能接受數字的運算了

```
[程式]: # 這次我把參數的名字叫做 args
        def add_multiple(args):
            # 因為我預期是一個 list 了, 所以直接走過
            result = 0
            for i in args:
                result = result + i
            return result

        # 確實可以傳入多個參數了
        print("傳入一個 list:", add_multiple([3, 4, 5, 6, 7]))
```

```
傳入一個 list: 25
```

但是這樣感覺起來並不是真的可以帶入多個，那怎麼辦呢?

我們可以拜託 python 幫我們加這個 [] 嗎? 這樣看起來就會有『可以帶入多個的假象』了!

我們要用 * 拜託 Python 幫我們把帶入的東西加上 []

```
[程式]: # 大家會習慣使用 args 來跟著不定參數使用
        def test(*args):
            print(args)
        test(3, 4, 5, 6, 7)
```

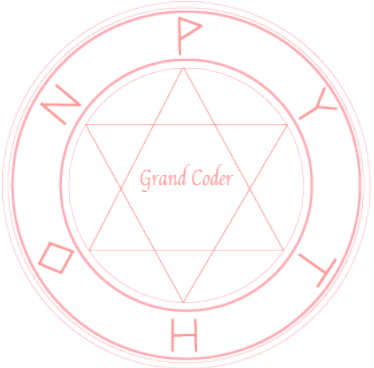
```
(3, 4, 5, 6, 7)
```

你會發現，其實 Python 是幫你把所有的東西用一個 tuple，其實是加上 () 才對，但如果要方便記憶，你記憶成幫忙加上 [ ](list) 也行

那我們把上面的修改成『不定參數』的版本

```
[程式]: # *: python 會把你所有帶入的東西組成一個群集
        def add_multiple(*args):
            # 一行都沒改
            result = 0
            for i in args:
                result = result + i
            return result

        # 你可以看到我這次沒有在自己加 []
        print("不定參數版本:", add_multiple(3, 4, 5, 6, 7))
```

```
不定參數版本: 25
```



9.4.2 (進階) 雙星號不定參數

還有一個加雙星號的，雙星號是指把你後面的名字 = 資料的組合加一個 { } 變成字典

```
[程式]: # 大家會習慣使用 kwargs 來跟著字典不定參數使用
        def test(**kwargs):
            print(kwargs)

        test(a = 3, b = 4, c = 5)
```

```
{'a': 3, 'b': 4, 'c': 5}
```

這有什麼好處呢? 好處是你隨時可以增加新的 option，而不用去改函式的定義

```
[程式]: def bmi(height, weight, **kwargs):
            bmi = weight / (height / 100) ** 2
            # 字典, 所以可以使用 in 檢查名字有沒有在裡面
            if "rounded" in kwargs:
                return round(bmi, kwargs["rounded"])
            else:
                return bmi

        print("不帶入 rounded:", bmi(175, 75))
        print("帶入 rounded:", bmi(175, 75, rounded=2))
```

```
不帶入 rounded: 24.489795918367346
帶入 rounded: 24.49
```

9.5 預設值

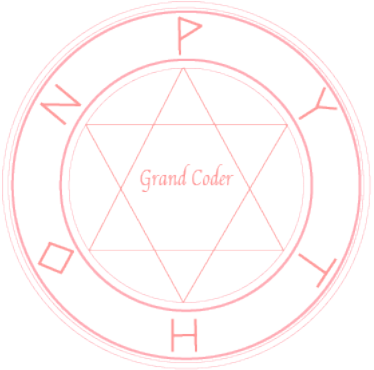
有時候你希望你的參數在不帶入的時候就使用預設值

❤ 當一個參數開始有預設值了以後，後面的所有參數都必須有預設值

❤ 有預設值的參數可以指定帶入，指定參數名 = 值 (e.g. open 的 encoding)

```
[程式]: # 我們讓 n3 是 *, n4 是/
        def add_default(n1, n2, n3=1, n4=1):
            return (n1 + n2) * n3 / n4

        # 有第三個, n3 被替換成 3
        print("替換 n3:", add_default(1, 2, 3))
        print("指定帶入 n4:", add_default(1, 2, n4=2))
```

```
替換 n3: 9.0
指定帶入 n4: 1.5
```



10 物件導向

10.1 物件導向簡介

『物件導向』一直是困擾許多程式初學者最大的問題，那到底什麼是『物件導向』呢?

我會這麼說：如果上一章節的『函式定義』是定義起一個『流程』讓你不用多次的撰寫。『物件導向』就是更進一步定義出一種『資料』讓你不用每次定義同樣的欄位!

這樣講可能會有點抽象，但我們用個例子來表示，我們上一章定義了 bmi 的計算函式

```
def bmi(height, weight):
    return weight / (height / 100) ** 2
```

但是如果我們又想要有一個也是使用 height 和 weight 來做事的『流程』，那我們就要在寫一次參數!

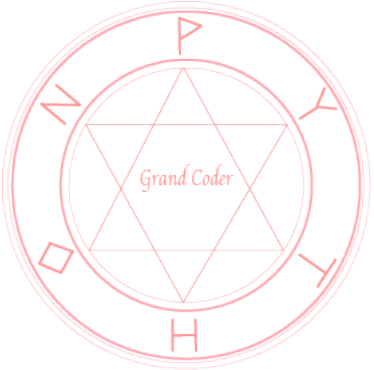
所以何不：定義起『人』的資料，把 height 和 weight 欄位正式的定義起來，接著把這些『流程』定義在『人』的資料底下呢?

10.2 物件導向的定義

如果要用比較正式的方式定義，我會這樣定義

『物件導向』:『大量』而且『快速』的創造『同類型』的資料

其實就是像我們現代社會的『大量生產』的概念



10.3 想像物件導向

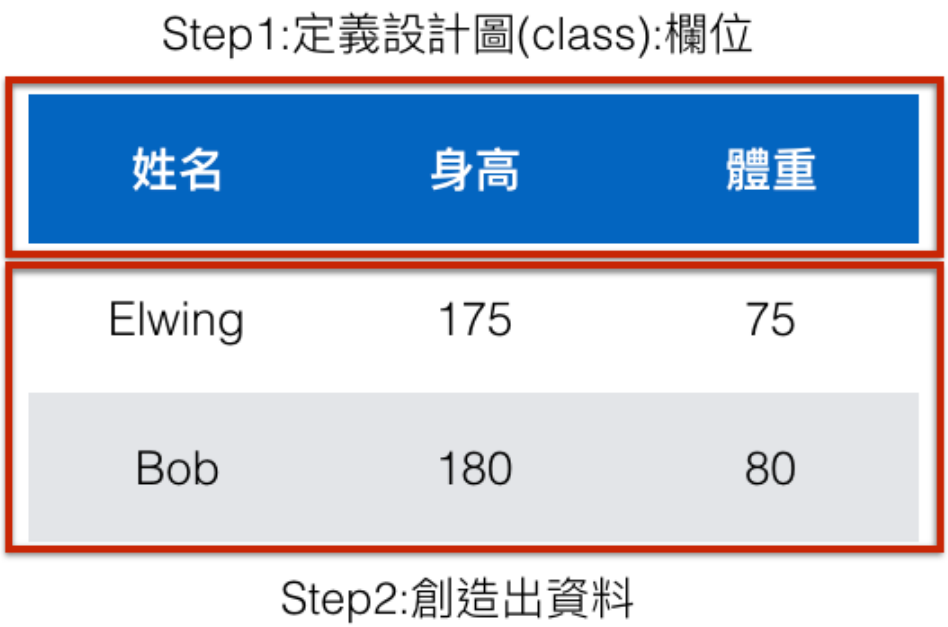


圖: 你該把物件導向想像成一個表格

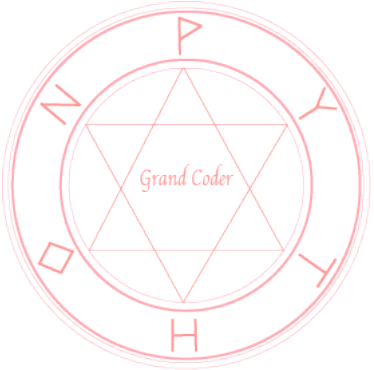
10.4 物件導向實作

10.4.1 Step1. 定義設計圖的欄位

[程式]:
```
# Step1. 用 class 關鍵字來定義設計圖
# 大家在給設計圖的名稱, 通常會第一個字大寫
class Person:
    # 其實定義欄位就是定義名字
    # 名字一定要在 = 左邊, 所以我們先設定 None 這個資料
    name = None
    height = None
    weight = None
```

10.4.2 Step2. 創造資料填入欄位的值

[程式]:
```
# Step2. 創造資料, () 你先把他視為創造的意思
# 給這個人叫做 p1 的名字
p1 = Person()
p1.name = "Elwing"
p1.height = 175
p1.weight = 75
bmi = p1.weight / (p1.height / 100) ** 2
print(p1.name, bmi)
```



```
Elwing 24.489795918367346
```

10.4.3 Step3. 定義專屬技能

你會發現如果我們要算 bmi，每次就要寫出算式，何不把它定義到『設計圖』底下，讓我們的人一長出來就會這個技能呢?

沒錯！聽起來很熟悉！就是我們的『專屬技能』

```
[程式]: class Person:
            name = None
            height = None
            weight = None
            def bmi(self):
                return self.weight / (self.height / 100) ** 2
```

你會發現我們多了一個叫做 self 的參數，這個 self 其實是因為我們把

```
p1.weight / (p1.height / 100) ** 2
```

放進去的時候，發現根本還沒有創造 p1，所以這裡 self 其實是前面是誰就會自動把 self = (前面的 p1 or p2)

```
[程式]: p1 = Person()
        p1.name = "Elwing"
        p1.height = 175
        p1.weight = 75
        print(p1.name, p1.bmi())

Elwing 24.489795918367346
```

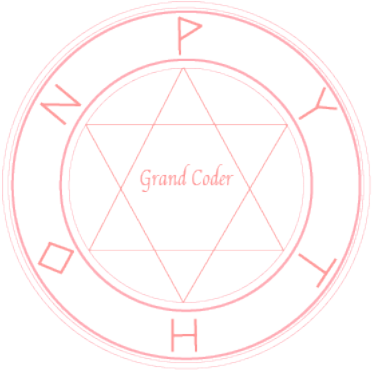
10.4.4 Step4. 定義初始流程

你會發現還有幾句程式一直在寫，而且很繁複，就是我們的設定資料

所以我們把他也拉成一個『流程』就好

不過這個『流程』必須給一個特別的名字，因為每次創造的時候要強迫做一次

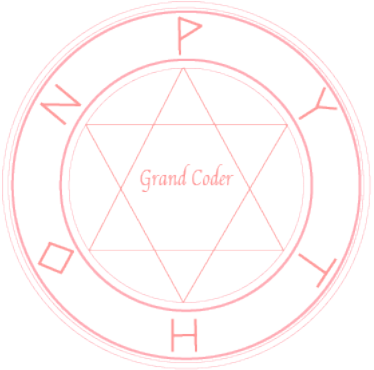
```
[程式]: class Person:
            # 這個特殊名字一定要叫做 __init__, 因為每次創造要自動執行
            # 不用再把欄位先寫在前面了
            def __init__(self, n, h, w):
                self.name = n
                self.height = h
```



```
            self.weight = w
        def bmi(self):
            return self.weight / (self.height / 100) ** 2

    # 沒錯這個 () 就是為了執行 init 流程的, 讓你帶入參數
    p1 = Person("Elwing", 175, 75)
    print(p1.name, p1.bmi())

    p2 = Person("Bob", 180, 80)
    print(p2.name, p2.bmi())
```

```
Elwing 24.489795918367346
Bob 24.691358024691358
```

我們最後終於完成整個『物件導向』了，讀者可以再回去看一次整個流程



11 錯誤處理

11.1 錯誤處理簡介

你會發現，就如同在現實生活中，有些東西是很難被預防的 (e.g. 地震，領錢餘額不足…等等)

譬如說：地震，如果要用預防的話，那會是『我預測今天中午十二點會發生地震，今天十二點時大家必須躲到屋外』，但事實上，我們沒辦法這樣來預防地震，所以我們採用的其實是『事後補救』，事先給大家『遇到地震的緊急措施』，等地震發生的時候趕快施行

譬如說：餘額不足，如果要用預防的話，那會是『一開始卡插入就連線銀行得到客戶的餘額，絕對不讓客戶輸入大於他餘額的錢』，事實上我們也不是這樣做的，而是，如果客戶輸入超出他餘額的錢，我們採取補救措施，取消剛剛不足的交易

這就是我所謂的『事後補救』，那程式的『事前預防』，其實就是我們以前學習到的『if-else』，而『事後補救』，就是我們今天要學習的『try-except』

```
try:
    可能會發生錯誤的程式碼
except XXXError:
    事後補救程式碼
```

另外要注意一下，當發生錯誤的程式碼那一行被執行的時候會立刻跳到 except 處理錯誤

11.2 實作

11.2.1 例子 1. 字典

之前我們在字典的時候做過數次數的例子，我們當初是使用『事前預防』的方法

```
test = " 這是測試字串, 測試測試測試"
result = {}
for c in test:
```

```
    if c in result:
        result[c] = result[c] + 1
    else:
        result = 1
```

那如果我們只寫

```
test = " 這是測試字串, 測試測試測試"
result = {}
for c in test:
    result[c] = result[c] + 1
```

會發生什麼事呢?

事實上我們會得到一個叫做 KeyError 的錯誤

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-1-d2496936efd9> in <module>()
      2 result = {}
      3 for c in test:
----> 4     result[c] = result[c] + 1

KeyError: '這'
```

圖:KeyError

我們試著捕抓這錯誤並且『補救』他

```
[程式]: test = "這是測試字串, 測試測試測試"
        result = {}
        for c in test:
            # 把有可能會發生錯誤的程式碼放在 try 裡
            try:
                result[c] = result[c] + 1
            # 在 except 針對錯誤解決
            except KeyError:
                result[c] = 1
        print("統計結果:", result)
```

```
統計結果: {'這': 1, '是': 1, '測': 4, '試': 4, '字': 1, '串': 1, ',': 1, ' ': 1}
```

11.2.2 例子 2. 檢查輸入

我們之前在讓使用者輸入整數的時候使用 int( ) 做出轉換，但大家應該也發現了，如果使用者亂輸入，會直接跳出紅字，我並沒有在之前的地方做出防範，因為 int 一下去的一剎那，錯誤就已經發生而來不及了，但是我們這裡可以透過『事後補救來解決』

```
i = int(input(" 請輸入整數:"))
```

你如果輸入一個 hello 得到的錯誤會是

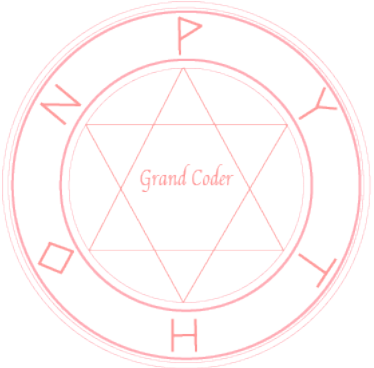
```
請輸入整數:hello

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-d9551b0d3372> in <module>()
----> 1 i = int(input("請輸入整數:"))

ValueError: invalid literal for int() with base 10: 'hello'
```

圖: 輸入 hello 得到的錯誤

```
[程式]: while True:
            try:
                i = int(input("請輸入整數:"))
                # 如果發生錯誤是絕對不會跑到 break 的
                break
            except ValueError:
                # 解決方式是直接忽略
                # 但配合上 while True + break 就可以一直到對為止
                print("請不要亂輸入")
        print("使用者的正確輸入:", i)

請輸入整數:abc
請不要亂輸入
請輸入整數:hello
請不要亂輸入
請輸入整數:345
使用者的正確輸入: 345
```

如果將『事後補救』(try-except) 加上『事前預防』(if-else)

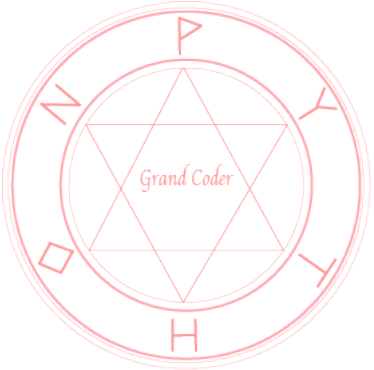
```
[程式]: while True:
            try:
                i = int(input("請輸入在 1~10 間的整數:"))
                # 加上 if-else, 正確範圍才 break
                if i >= 0 and i <= 10:
                    break
                else:
                    print("請輸入在正確範圍")
            except ValueError:
                print("請不要亂輸入")
        print("使用者的正確輸入:", i)

請輸入在 1~10 間的整數:abc
請不要亂輸入
```



```
請輸入在 1~10 間的整數:hello
請不要亂輸入
請輸入在 1~10 間的整數:345
請輸入在正確範圍
請輸入在 1~10 間的整數:8
使用者的正確輸入: 8
```

11.3 結語

『事後補救』和『事前預防』是我們處理錯誤的兩大機制，你應該把兩個都好好的善用，就可以處理幾乎所有的錯誤了!



12 Import

12.1 Import 基本使用

Python 有個很基本的準則，『只要用到別的.py 檔案，就一定要引述他』，記住，是只要不同檔案就要『引述』他為止唷，沒有所謂的『同一個資料夾下』或『同一層』之類任何的其他的規則，只要使用到那隻檔案就要 import 到他為止!

語法如下:

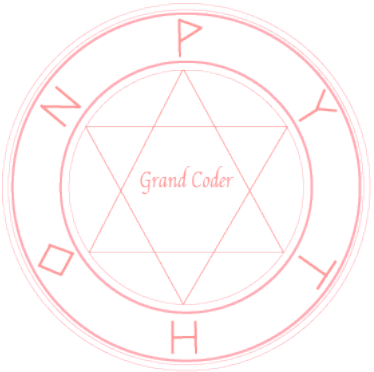
```
import (那隻 py 的名字)
```

12.2 練習基本 import

在 Python 內建裡，有個檔案叫做 this.py，他是 Python 創始者留下的一個彩蛋，裡面寫著一首叫做 The Zen Of Python 的文章，大致的主題是整個 Python 語言的良好寫作方式，我們用這隻 py 來教各位 import 的方式

你可以在作業系統直接搜尋 this.py，這裡我將它的內容大概截圖給你看

圖: 電腦裡的 this.py



```
s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
```

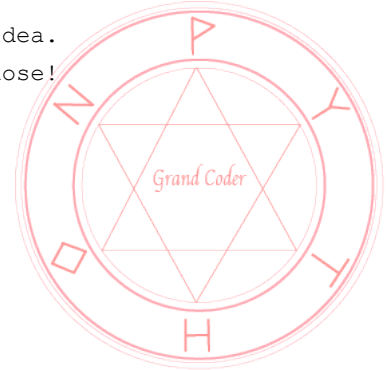
圖: this.py 的一些內容

你會很好奇為何是一些看起來是亂碼的字，事實上，這是一種叫做『凱薩密碼』的加密方式，對所有文字都做一個位置偏移，例如: 2 的偏移，a 就變成 c，f 就變 h

廢話不多說，先來 import 看看

```
[程式]: import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

為什麼會印出這樣一篇文章呢? 事實上是由



```
28    print("".join([d.get(c, c) for c in s]))
```

圖: 印出的程式碼

所以你現在也知道什麼是『引述』了，其實就像是把那 py 複製過來，執行一遍！所以才會執行到這行就 print 文章出來

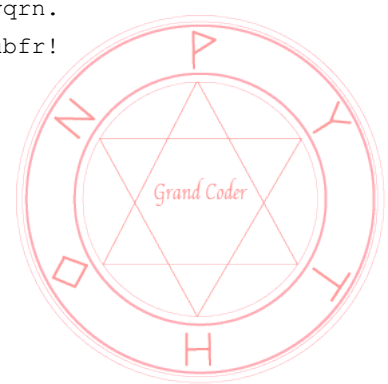
現在要解決你的最後一個疑惑了，那如果我想使用定義在這個 py 裡的一些東西該怎麼用呢?

答案是:『照抄一次你 import 後面寫的東西在. 下去』. 就是我們中文的『的』意思

語法:

```
import a
# 先抄一遍, 再繼續用你想用的東西
a.xxx
```

```
[程式]: import this
        # 注意一下最上面那張圖, 我們印出的是裡面的 s 這個字串
        print(this.s)

Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!
```



12.3 進階練習 - 關鍵詞萃取

還記得我們在字典的時候，做過數『字』出現的次數，但我們那時候說過，對於文字而言，真正有意義的其實是『詞』的次數

那數完詞的次數我們可以做什麼呢?

事實上，我們可以也最想做到的一件事就是，『重點的萃取』，什麼叫做重點的萃取呢?

人類在理解語言的時候，一個最重要的處理其實就是去蕪存菁，把真正有意思，想要知道的訊息從一堆詞裡萃取出來

那我們今天如果也想要教電腦『看懂文章』或『聽懂說話』，最重要的事就是也教電腦如何『抓出重點詞』

下面我們來介紹一下這個『抓出重點』的理論『TF-IDF 方法』

12.3.1 TF-IDF 方法

TF-IDF 是一個非常常使用來抓出『重點詞』的方法，概念非常簡單

❤ TF(Term Frequency): 在一篇文章中，如果一個詞出現越多次，我們就越覺得這個詞會是整篇文章的重點，例如:『牛排』在文章中出現 10 次，『蘋果』在文章中出現 1 次，我們給牛排較高的重要分數!

❤ IDF(Inverse Document Frequency): 雖然我們有了數次數，但有的詞雖然出現的多，但卻不會是整篇文章的重點，譬如:『我』,『在』這些連結詞，或者是『盈餘』這種太常出現的詞，最後我們有個結論，我們在看文章的時候，如果遇到我們平常越不常看到的『詞』，我們的腦袋越會把『關注』放在這個詞身上，所以我們怎麼計算 IDF 係數呢? 我們先蒐集足夠多的文章，然後把『詞出現在多少篇文章』算出來，出現在越多篇文章中過，代表這詞越不重要! 例如: 我事先蒐集好 10000 篇文章,『盈餘』在 9000 篇都有出現過,『愛情』只在 10 篇文章出現過，那我們會認為『愛情』其實是一個比較不常見的詞，給他較高的重要分數!

最後怎麼衡量一個詞的『完整重要程度』呢? 把 TF 分數和 IDF 分數乘起來就可以了

白話文來說，就是先『數次數』，再看這個詞的『慣用程度』來修正

完整公式 (參考用):

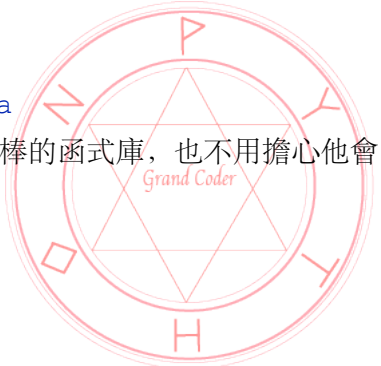
$$importance(\text{詞重要程度}) = \frac{\text{特定詞出現次數}}{\text{這篇文章所有的詞數}}(TF) \times \log_{10} \frac{\text{圖書館總文章數}}{\text{特定詞出現在多少篇文章}}(IDF)$$

圖: 完整公式

12.3.2 安裝 Jieba 第三方函式庫

Jieba 函式庫說明文件: https://github.com/fxsjy/jieba

首先我們先來安裝一下函式庫，Jieba 是一個中文語言處理非常棒的函式庫，也不用擔心他會對繁體支援差，事實上只有一些繁體常用詞會分辨不出來



如何安裝呢？有兩種方法

1. 使用 PyCharm 幫忙安裝 (建議)
2. 使用命令列安裝 (會遇到虛擬環境的問題)

PyCharm 請你先來到 File - Settings，MAC 電腦的同學請按上面的 PyCharm - Preference
接下來照著下面的圖選到 Project XXX - Project Interpreter

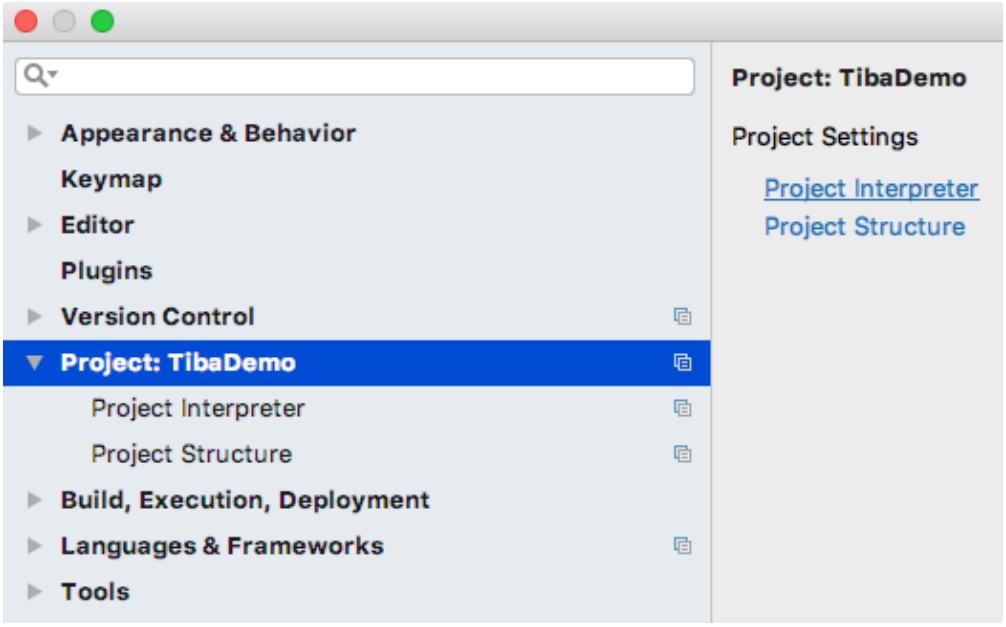
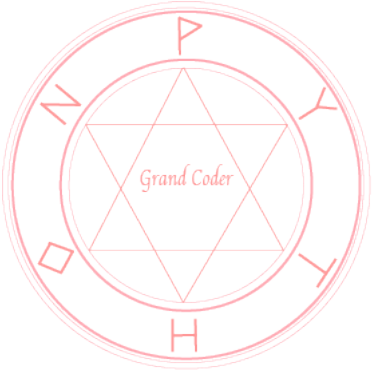


圖: Project Interpreter

接著點擊下面的 + 來搜尋函式庫



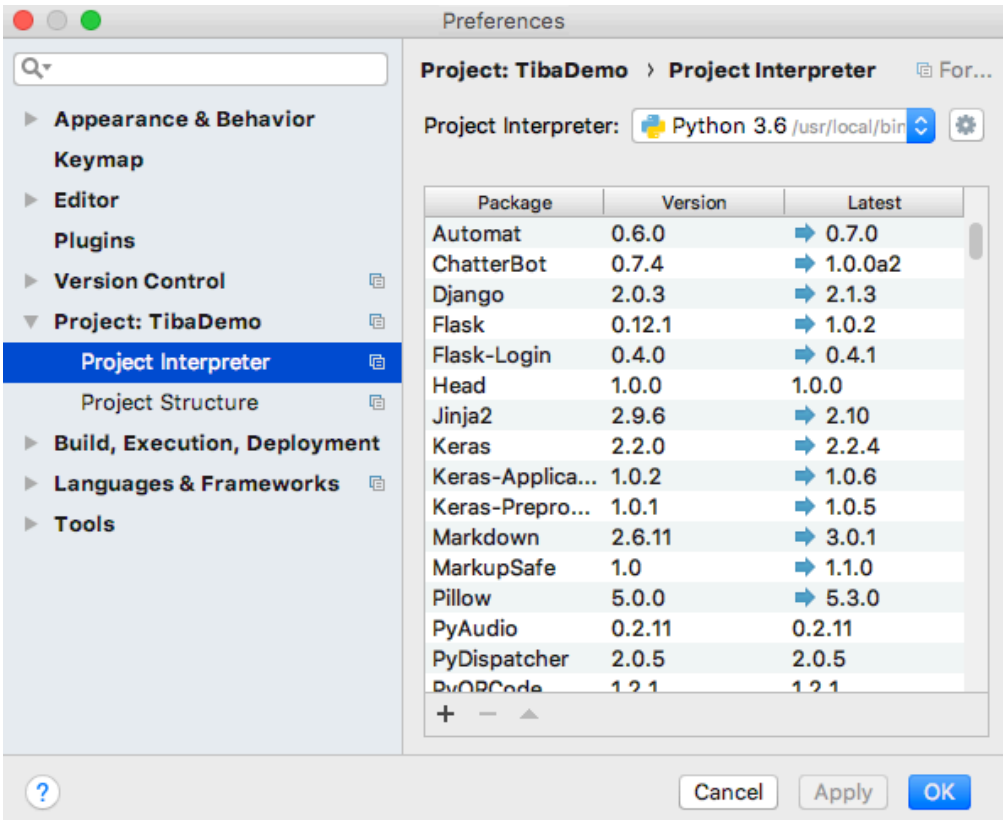
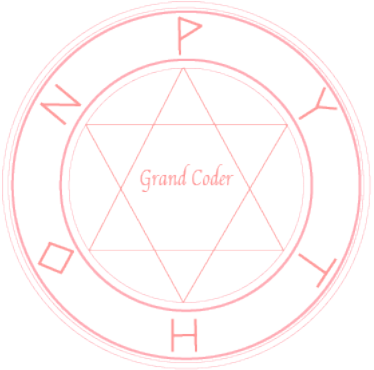


圖: 選下面的 +

搜尋 jieba 並且按下 install package



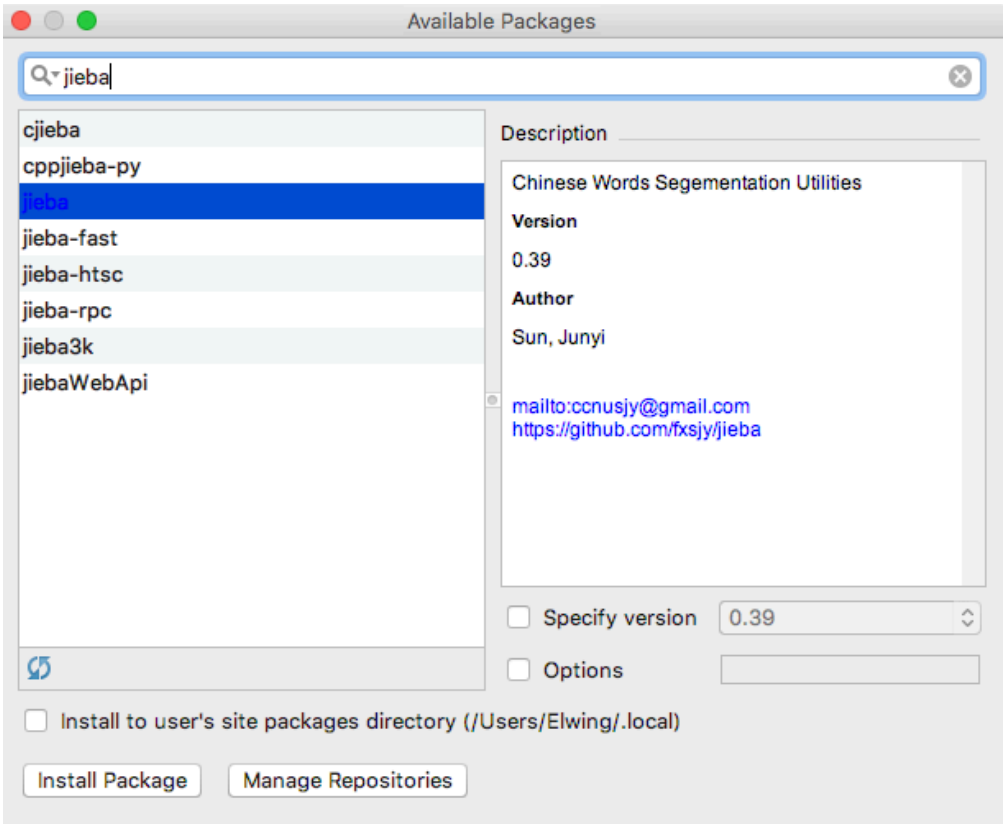
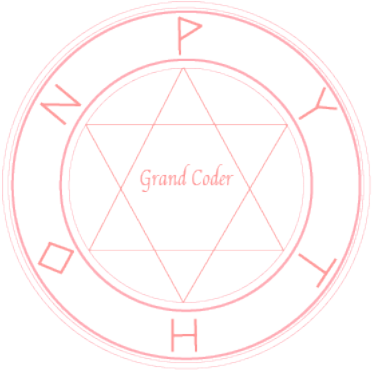


圖: 選擇 install package

最後會出現綠色的成功，就是安裝完畢了



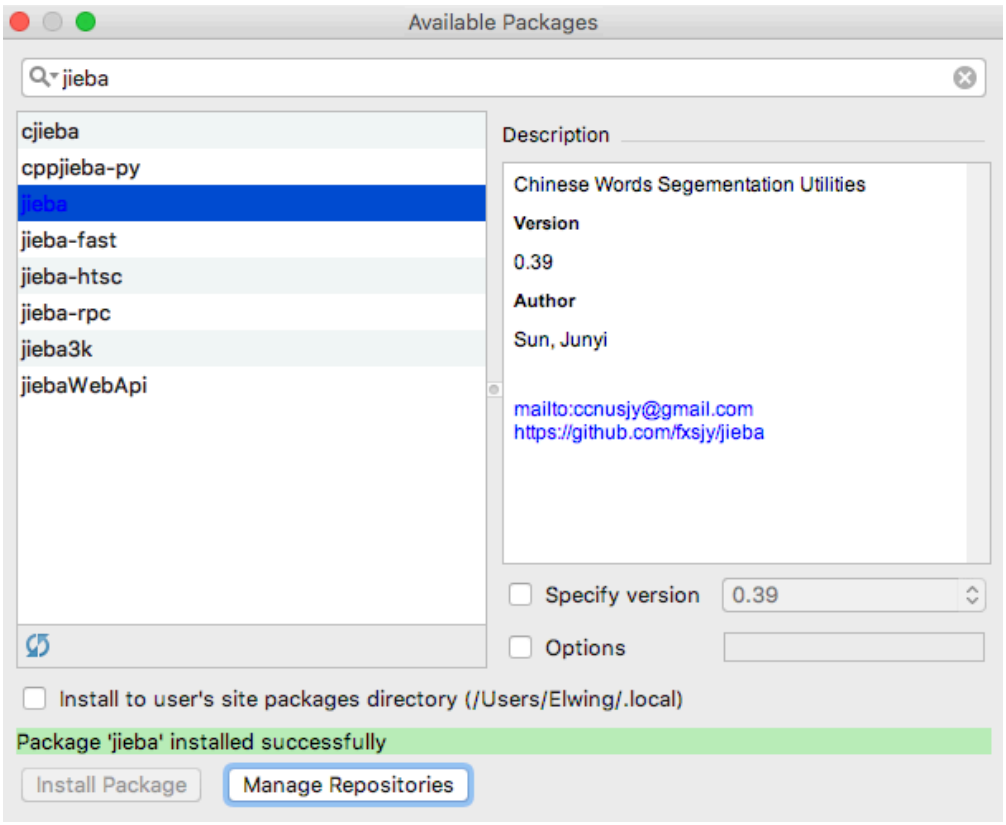


圖: 成功安裝

12.3.3 Jieba 函式庫結構

我們安裝的函式庫究竟都安裝在哪裡呢?

事實上，他是安裝在你的 Python 翻譯器身上，你仔細觀察一下你的 Project 下面連接翻譯器的部分

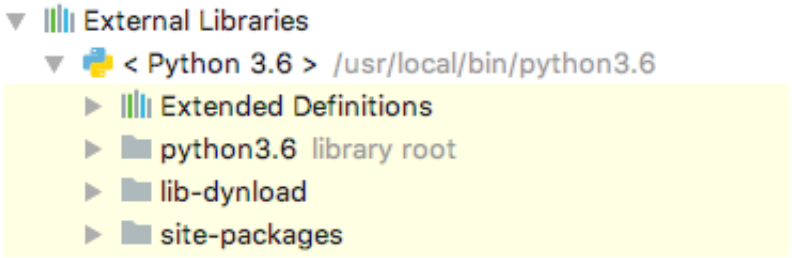
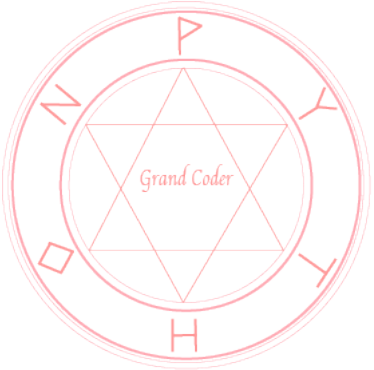


圖: 安裝所在

打開來看你就可以看到 jieba



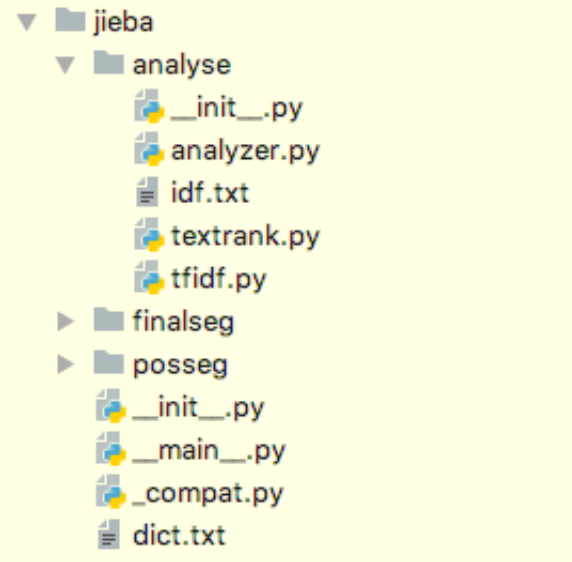


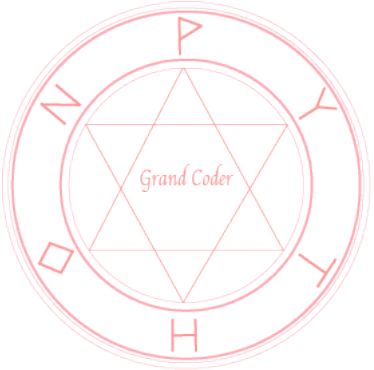
圖: jieba 函式庫結構

12.3.4 分詞

在 TF-IDF 方法前，我們要先做一件事，這件事情叫做『分詞』，就是把詞分開，Jieba 把詞分開的方式是

1. 先從詞典裡去找，找到對應的詞就將其分開
2. 有多種可能或者在詞典裡看不到的話，利用機率來決定哪個可能性大

你先觀察一下剛剛結構下面的 dict.txt



```
1   AT&T 3 nz
2   B超 3 n
3   c# 3 nz
4   C# 3 nz
5   c++ 3 nz
6   C++ 3 nz
7   T恤 4 n
8   A座 3 n
9   A股 3 n
10  A型 3 n
```

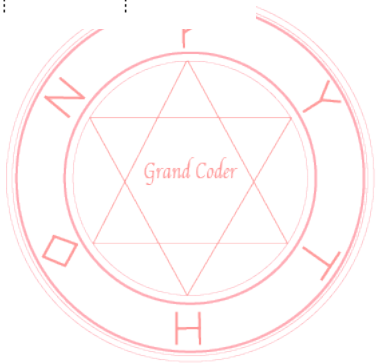
圖: 分割詞用的詞典

你會發現每個詞都以詞詞頻詞性來紀錄的，後面的兩個是為了機率算法而存的，所以你可以先忽略他

如果你對詳細詞性有興趣，我幫你把他列到下面

標註	詞性	標註	詞性	標註	詞性	標註	詞性
n	普通名詞	f	方位名詞	s	處所名詞	t	時間名詞
nr	人名	ns	地名	nt	機構團體名	nw	作品名
nz	其他專名	v	普通動詞	vd	動副詞	vn	名動詞
a	形容詞	ad	副動詞	an	名形詞	d	副詞
m	數量詞	q	量詞	r	代詞	p	介詞
c	連詞	u	助詞	xc	其它虛詞	w	標點符號

圖: 詞性



12.3.5 實際分詞

在看實際分詞前，我們要先看再看一次整個安裝來的結構
現在在你的世界裡，py 分成兩種，一種是名字叫做 __init__.py 另外一種是普通的.py
init.py 是整個資料夾 (部門) 的主管，而普通的.py 則是員工

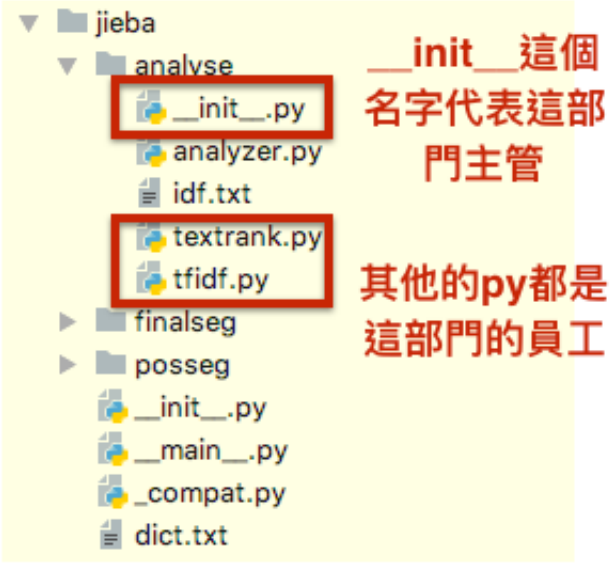


圖: 主管以及員工

在語法上

```
# 其實是引述 jieba.analyse.__init__
import jieba.analyse
# 普通的就要確實寫到那隻
import jieba.analyse.tfidf
```

分詞所使用的 cut 技能則在 jieba 資料夾正下方的 __init__.py 裡

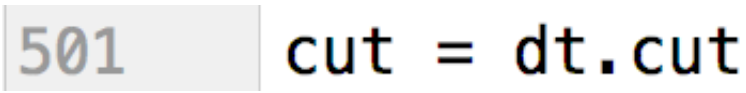


圖:cut 所在位置

```
[程式]: # 在這裡我們先學會一個字串的專屬技能 join
        # 你可以帶入一個群集給他, 他會將群集裡的東西利用前面的分隔符號組合回去變成一個字串
        print("/".join(["2018", "11", "10"]))
```

```
2018/11/10
```

你發現在下們我們的分詞有些分錯的地方，這是完全正常的，因為對於沒看過的詞，是用機率來判斷到底如何分類，所以難免會有點錯誤

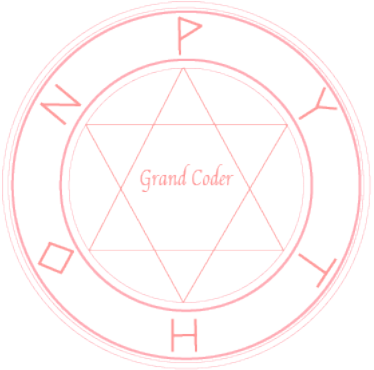
```
[程式]: import jieba
        # 這是我們之前學會的讀取檔案三部曲, "r" 代表唯讀模式
        f = open("a.txt", "r", encoding="utf-8")
        article = f.read()
        f.close()
        # 利用上面學到的 join, 我們用空白鍵把切割出來的字詞群集再用空白鍵組合回來
        print(" ".join(jieba.cut(article)))
```

```
Python 是 一種 廣泛 使用 的 高階 程式 語言 ，屬 於 通用型 程式 語言 ，由 吉多 ·范羅蘇姆
創造 ，第一版 釋出 於 1991 年 。可以 視之為 一種 改良 （加入 一些 其他 程式 語言 的 優點 ，
如 物件 導向 ）的 LISP 。作為 一種 直譯 語言 ， Python 的 設計 哲學 強調 程式 碼 的 可
讀性 和 簡潔 的 語法 （尤其 是 使用 空格 縮排 劃分 程式 碼塊 ，而 非 使用 大括號 或者 關鍵 詞 ）
。相比 於 C++ 或 Java ， Python 讓 開 發者 能夠 用 更 少 的 代碼 表達 想法 。不管 是
小型 還是 大型 程式 ，該 語言 都 試圖 讓 程式 的 結構 清晰 明 了 。 與 Scheme 、
Ruby 、 Perl 、 Tcl 等動態 型 別 程式 語言 一樣 ， Python 擁有 動態 型 別系統 和 垃圾
回收 功能 ，能夠 自動 管理 記憶體 使用 ，並且 支援 多種 編程 範式 ，包括 物件 導向 、命令式 、
函數式 和 程序 式 編程 。其 本身 擁有 一個 巨大 而廣泛 的 標準庫 。 Python 直譯器
本身 幾乎 可以 在 所有 的 作業 系統 中 執行 。 Python 的 正式 直譯器 CPython 是 用 C
語言 編寫 的 、是 一個 由 社群 驅動 的 自由 軟體 ，目前 由 Python 軟體 基金 會 管理 。
```

12.3.6 抓出關鍵詞

我們之前提過我們會使用 TF-IDF 方法來抓出關鍵詞，不過有個東西必須提早被計算，那就是 IDF 係數

請你把剛剛 jieba 資料夾下的 analyse 資料夾的 idf.txt 打開看看



```
1  劳动防护 13.900677652
2  生化学 13.900677652
3  奥萨贝尔 13.900677652
4  考察队员 13.900677652
5  岗上 11.5027823792
6  倒车档 12.2912397395
7  编译 9.21854642485
8  蝶泳 11.1926274509
9  外委 11.8212361103
10 故作高深 11.9547675029
```

圖: jieba/analyse/idf.txt

你可以看到在詞的後面有許多數字，那數字就是預先算好的 idf 係數，越大代表這詞越常用，重要係數越高，反之，越小則代表重要係數越小

接下來我們就 import jieba 下面的 analyse 下面的 __init__.py，記得沒有所謂 import 整個資料夾下面的檔案這種事，想要哪隻檔案就要 import 到他，我們要使用裡面的 extract_tags 這個功能

```
12  extract_tags = tfidf = default_tfidf.extract_tags
```

圖: jieba/analyse/init.py

```
[程式]: # 事實上是 import jieba.analyse.__init__
        import jieba.analyse
        # 照抄一遍再繼續寫，只抓出五個關鍵詞
        words = jieba.analyse.extract_tags(article, 5)
        print("關鍵詞:", words)

關鍵詞: ['Python', '語言', '程式', '一種', '導向']
```

12.4 from…import

想必這裡你也發現了，每次你要抄 jieba.analyse 真的累，所以我們其實有個更方便的語法，我喜歡叫他做部分 import

語法是這樣

```
from 基準點 import 想要的東西
```

因為我們的大原則是『import 後面是什麼就照抄一次再繼續寫』，所以你就可以少寫很多東西

但特別的是 from...import 甚至可以直接 import 到更細微的名字

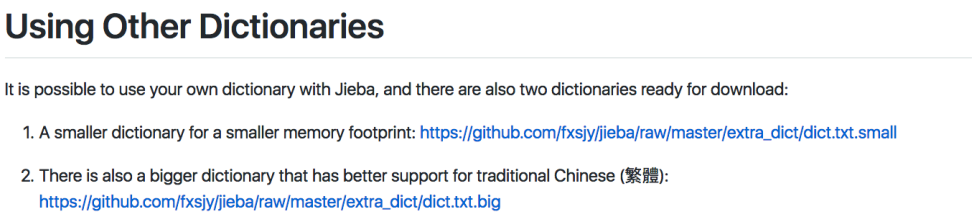
所以上面的同一段其實我們有三種不同的寫法

我個人是最喜歡第三種寫法，因為會讓你的 import 很清晰，大家看你的 import 就大概知道你的程式是做什麼用的!

```
[程式]: import jieba.analyse
        print("寫法 1:", jieba.analyse.extract_tags(article, 5))
        from jieba import analyse
        print("寫法 2:", analyse.extract_tags(article, 5))
        from jieba.analyse import extract_tags
        print("寫法 3:", extract_tags(article, 5))

寫法 1: ['Python', '語言', '程式', '一種', '導向']
寫法 2: ['Python', '語言', '程式', '一種', '導向']
寫法 3: ['Python', '語言', '程式', '一種', '導向']
```

12.4.1 (進階) 載入自行準備詞典

其實 Jieba 有提供一個更大的詞典，裡面包括更多繁體專用字詞，會讓分詞更精準，我們一起來試試看



Using Other Dictionaries

It is possible to use your own dictionary with Jieba, and there are also two dictionaries ready for download:

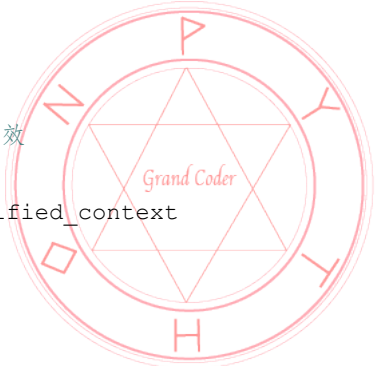
1. A smaller dictionary for a smaller memory footprint: https://github.com/fxsjy/jieba/raw/master/extra_dict/dict.txt.small
2. There is also a bigger dictionary that has better support for traditional Chinese (繁體): https://github.com/fxsjy/jieba/raw/master/extra_dict/dict.txt.big

圖: jieba 額外字典

當然我們可以把他下載下來放在同一個資料夾，不過我們這裡要酷一下，直接從網路下載，這樣就可以隨時保持最新了

你可以看到下面的分詞就精準多了，而且會抓出更正確的關鍵詞

```
[程式]: # 內建函式庫, 可以幫我們下載檔案
        from urllib.request import urlretrieve
        # MAC 電腦要特別加入這兩行
        # 因為 MAC 電腦 +python 會有個 bug, 把對面的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
```



```
        url = "https://raw.githubusercontent.com/fxsjy/jieba/master/extra_dict/dict.txt.big"
        # 下載到同一個資料夾的 bigdict.txt, 你可以在你的 Pycharm 看到檔案
        urlretrieve(url, "bigdict.txt")

        import jieba

        # 讀入額外的字典
        jieba.load_userdict("bigdict.txt")
        f = open("a.txt", "r", encoding="utf-8")
        article = f.read()
        f.close()
        # 利用上面學到的 join, 我們用空白鍵把切割出來的字詞群集再用空白鍵組合回來
        print(" ".join(jieba.cut(article)))

        from jieba.analyse import extract_tags
        print("關鍵詞:", extract_tags(article, 5))
```

```
Python 是 一種 廣泛 使用 的 高階 程式 語言 ，屬於 通用型 程式 語言 ，由 吉多 ·范 羅蘇 姆
創造 ，第一版 釋 出於 1991 年 。可以 視之為 一種 改良 （加入 一些 其他 程式 語言 的 優點 ，
如 物件 導向 ）的 LISP 。作為 一種 直譯 語言 ， Python 的 設計 哲學 強調 程式碼 的 可讀性
和 簡潔 的 語法 （尤其 是 使用 空格 縮排 劃分 程式碼 塊 ，而 非 使用 大括號 或者 關鍵詞 ）。
相比 於 C++ 或 Java ， Python 讓 開發者 能夠 用 更 少 的 代碼 表達 想法 。不管 是 小型
還是 大型 程式 ，該 語言 都 試圖 讓 程式 的 結構 清晰 明 了 。 與 Scheme 、 Ruby
、 Perl 、 Tcl 等 動態 型別 程式 語言 一樣 ， Python 擁有 動態 型別 系統 和 垃圾 回收
功能 ，能夠 自動 管理 記憶體 使用 ，並且 支援 多種 編程 範式 ，包括 物件 導向 、命令式 、函數
式 和 程序 式 編程 。其 本身 擁有 一個 巨大 而 廣泛 的 標準 庫 。 Python 直譯器
本身 幾乎 可以 在 所有 的 作業系統 中 執行 。 Python 的 正式 直譯器 CPython 是 用 C 語言
編寫 的 、是 一個 由 社群 驅動 的 自由 軟體 ，目前 由 Python 軟體 基金會 管理 。
關鍵詞: ['Python', '語言', '程式', '一種', '廣泛']
```

12.4.2 自行準備專業詞典

你也可以自己準備你自己的詞典，我們先用原本的來試試看，我先載入另外一篇文章，是周杰倫在 wiki 上的條目

```
[程式]: # 內建函式庫, 可以幫我們下載檔案
        from urllib.request import urlretrieve
        # MAC 電腦要特別加入這兩行
        # 因為 MAC 電腦 +python 會有個 bug, 把對面的 ssl 憑證視為無效
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
        url = "https://raw.githubusercontent.com/fxsjy/jieba/master/extra_dict/dict.txt.big"
        # 下載到同一個資料夾的 bigdict.txt, 你可以在你的 Pycharm 看到檔案
        urlretrieve(url, "bigdict.txt")
```

```
import jieba

# 讀入額外的字典
jieba.load_userdict("bigdict.txt")
f = open("b.txt", "r", encoding="utf-8")
article = f.read()
f.close()
# 利用上面學到的 join, 我們用空白鍵把切割出來的字詞群集再用空白鍵組合回來
print(" ".join(jieba.cut(article)))

from jieba.analyse import extract_tags
print("關鍵詞:", extract_tags(article, 5))
```

```
周杰倫 在 台灣 台北 林口 長大 。父親 周耀中 ，當時 任教於 台北 縣 蘆洲 國 中 ，教授 生物 ；母親 葉惠美 則是 林口 國 中 美術 教師 。 14 歲 時 父母 離異 ，由 父親 擔任 監護人 ，年滿 18 歲 後 選擇 與 母親 共同 生活 ，周杰倫 在 胡婉玲民視 的 《台灣 演義 》專訪 中 ，澄清 《爸 ，我 回來 了 》的 靈感 ，只是 對 社會 上 家暴 現象 的 感慨 ，並非 指涉 父母 間 的 狀況 ，父親 方面 的 親戚 也 曾 質疑 過他 ，他 還為 此 向 親戚 們 澄清 ，為 此 誤解 抱歉 過 。 周杰倫 他 自小 對 音樂 表現 出 濃厚 的 興趣 ，而且 喜歡 模仿 歌星 、演員 表演 和 變魔術 。 3 歲 開始 學習 鋼琴 。周杰倫 國 小時 住 在 臺北市 光華 商場 附近 ，就讀 忠孝國小 。國 中時 就讀 金華 國 中 ，此 時期 他 的 父母 因 長年 爭執 而 決議 離婚 ，使 周杰倫 的 性情 大受 影響 。除了 音樂 外 ，周杰倫 熱愛 籃球 ，他 在 國中 參加 籃球隊 ，結識 學長 陳建州 。在 同儕 中是 高手 ，球場上 的 表現 常受 眾人 圍觀 。 高中 就讀於 台北 縣 私立 淡江 中學 第一屆 音樂 科 (本來 是 想 報考 華岡 藝校 ，但 錯過 了 報名 期間 ，幸好 淡江 中學 恰巧 新設 了 音樂 科 )，主修 鋼琴 ，副 修 大提琴 ，為 將來 的 音樂 發展 打下 了 深厚 的 基礎 。這時 的 他 因 正值 青春期 ，常常 秀 琴技 想 吸引 女同學 的 注意 。但 學科 成績 不 甚 理想 ，故 高中畢業 時 ，大學 聯考 落榜 。又 因 患有 僵直 性 脊椎炎 ，依據 臺灣 兵役制度 得 以免 服 義務兵役 。 周杰倫 曾 說過 表示 少 年時 受到 香港 樂壇 「四大 天王 」之一 張學友 的 專輯 《吻別 》的 影響 ，從而 喜歡 並 開始 專注 於 流行音樂 。另外 他 也 透漏 過 ，除了 張學友 以外 ，李恕權 與 史帝夫 ·汪達 也 是 他 童 年時 影響 他 很 深 的 人 ，李恕權 每回 出現 在 電視 上 ，周杰倫 便會 在 電視機 面前 模仿 他 ，而史 帝夫 ·汪達 有 一首 《 I just called to say I love you 》是 他 的 嬸嬸 曾 在 他 叔父 的 葬禮 中 播放 的 歌曲 。由於 他 的 音樂 基礎 紮實 ，令其 在 流行音樂 創作 方面 如魚得水 。
關鍵詞: ['周杰倫', '音樂', '父親', '影響', '台北']
```

你可以看到像『國中』,『國小』,『高中』，這些字詞都沒有辨識出來，有時候這些沒正確分詞會影響你的關鍵詞抓取，於是我們自己準備一個 mydict.txt 在同一層資料夾，內容如下

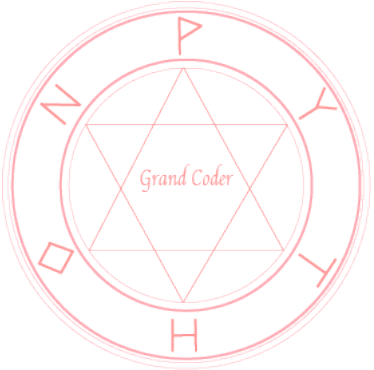


圖: 我自行準備的詞典

這裡你可以看到，詞頻和詞性你可以省略掉是完全沒關係的，建議各位在做專業的分詞的時候，可以載入一個自己的專業詞典

在我們載入這個自定義詞典後，整個分詞更精確了

```
[程式]: from urllib.request import urlretrieve
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
        url = "https://raw.githubusercontent.com/fxsjy/jieba/master/extra_dict/dict.txt.big"
        urlretrieve(url, "bigdict.txt")

        import jieba

        jieba.load_userdict("bigdict.txt")
        # 載入我自己的詞典
        jieba.load_userdict("mydict.txt")

        f = open("b.txt", "r", encoding="utf-8")
        article = f.read()
        f.close()
        print(" ".join(jieba.cut(article)))

        from jieba.analyse import extract_tags
        print("關鍵詞:", extract_tags(article, 5))
```

```
周杰倫 在 台灣 台北 林口 長大 。父親 周耀中 ，當時 任教於 台北 縣 蘆洲 國中 ，教授 生物 ；母親 葉惠美 則是 林口 國中 美術 教師 。 14 歲 時 父母 離異 ，由 父親 擔任 監護人 ，年滿 18 歲 後 選擇 與 母親 共同 生活 ，周杰倫 在 胡婉玲民視 的 《台灣 演義 》專訪 中 ，澄清 《爸 ，我 回來了 》的 靈感 ，只是 對 社會 上 家暴 現象 的 感慨 ，並非 指涉 父母 間 的 狀況 ，父親 方面 的 親戚 也 曾 質疑 過他 ，他 還為 此 向 親戚 們 澄清 ，為 此 誤解 抱歉 過 。 周杰倫 他 自小
```

對 音樂 表現 出 濃厚 的 興趣 ，而且 喜歡 模仿 歌星 、演員 表演 和 變魔術 。 3 歲 開始 學習 鋼琴 。周杰倫 國 小時 住 在 臺北市 光華 商場 附近 ，就讀 忠孝國小 。國 中時 就讀 金華 國中 ，此 時期 他 的 父母 因 長年 爭執 而 決議 離婚 ，使 周杰倫 的 性情 大受 影響 。除了 音樂 外 ，周杰倫 熱愛 籃球 ，他 在 國中 參加 籃球隊 ，結識 學長 陳建州 。在 同儕 中是 高手 ，球場上 的 表現 常受 眾人 圍觀 。 高中 就讀於 台北 縣 私立 淡江 中學 第一屆 音樂 科 （本來 是 想 報考 華岡 藝校 ，但 錯過 了 報名 期間 ，幸好 淡江 中學 恰巧 新設 了 音樂 科 ），主修 鋼琴 ，副 修 大提琴 ，為 將來 的 音樂 發展 打下 了 深厚 的 基礎 。這時 的 他 因 正值 青春期 ，常常 秀 琴技 想 吸引 女同學 的 注意 。但 學科 成績 不 甚 理想 ，故 高中畢業 時 ，大學 聯考 落榜 。又 因 患有 僵直 性 脊椎炎 ，依據 臺灣 兵役制度 得 以免 服 義務兵役 。 周杰倫 曾 說過 表示 少 年時 受到 香港 樂壇 「四大 天王 」之一 張學友 的 專輯 《吻別 》的 影響 ，從而 喜歡 並 開始 專注 於 流行音樂 。另外 他 也 透漏 過 ，除了 張學友 以外 ，李恕權 與 史帝夫 ·汪達 也 是 他 童 年時 影響 他 很 深 的 人 ，李恕權 每回 出現 在 電視 上 ，周杰倫 便會 在 電視機 面前 模仿 他 ，而史 帝夫 ·汪達 有 一首 《 I just called to say I love you 》是 他 的 嬸嬸 曾 在 他 叔父 的 葬禮 中 播放 的 歌曲 。由於 他 的 音樂 基礎 紮實 ，令其 在 流行音樂 創作 方面 如魚得水 。

關鍵詞: ['周杰倫', '音樂', '國中', '父親', '影響']

12.5 結語

我們在這章節，教了你 Python 最重要的一件事，就是載入工具，使用工具！還教了你一個非常有趣的函式庫，『Jieba』函式庫，以後配合大量的資料抓取，你就可以做到大量的資料分析！