

# Python Programming

2023 Spring; week 16

Instructor: Cheng-Chun Chang (張正春)  
Department of Electrical Engineering

Textbook: Python程式設計:從入門到進階應用(第三版) 2020

# 課程助教

協助**dubug**, 禁止同學看**code**照抄

---

第一排:

第二排:

第三排:

第四排:

第五排:

第六排:

打游擊:

# Ch7模組、套件 與獨立程式

## 7-1 模組

---

- 模組就是一個Python 檔案，每一個Python 檔案被視為一個模組，可以在程式中匯入其他Python 模組，模組就可以不斷地被其他程式再利用。
- 到此已經介紹程式的關鍵字(if、for、while…)、變數與運算子，可以想成單字

- 
- 多個關鍵字、變數與運算子組合成一行程式，可以想成句子
  - 多行程式可以組合成函式，可以想成段落
  - 多個函式可以組合成模組，可以想成是一篇文章，以下介紹模組的實作與匯入模組。


---


# Programming codes I



## 7-1-1 實作模組


- Python 的模組就是一個檔案，實作一個模組，可以隨機回傳「剪刀」、「石頭」、「布」三個其中一個。

行號	範例 (  : ch7\guess.py)
1	import random
2	status = ['剪刀', '石頭', '布']
3	def figure_guess():
4	return random.choice(status)

行號	範例 (  : ch7\ch7-1-2-1a-mod.py)	執行結果
1	import guess	石頭
2	computer = guess.figure_guess()	
3	print(computer)	

## 7-1-2 匯入模組


---

行號	範例 (  : ch7\ch7-1-2-1a-mod.py)	執行結果
1	import guess	石頭
2	computer = guess.figure_guess()	
3	print(computer)	




## 7-1-2 匯入模組

---

行號	範例 (  : ch7\ch7-1-2-1b-mod.py)	執行結果
1	from guess import figure_guess	剪刀
2	computer = figure_guess()	
3	print(computer)	

## 7-1-2 匯入模組

---


行號	範例 (  : ch7\ch7-1-2-1c-mod.py)	執行結果
1	<code>import guess as gs</code>	剪刀
2	<code>computer = gs.figure_guess()</code>	
3	<code>print(computer)</code>	

## 7-1-3 匯入模組的路徑

---

- 若想要知道Python 匯入模組的資料夾路徑與順序，需先匯入模組sys，讀取sys.path 的每一個元素就可以知道，可以發現第一個找尋模組是否存在的資料夾就在執行程式的資料夾下，若找到就不會到下一個資料夾去找尋。

## 7-1-3 匯入模組的路徑

行號	範例 (  : ch7\ch7-1-3-mod.py)	程式說明
1	<code>import sys</code>	第 1 行：匯入模組 <code>sys</code> 。
2	<code>for path in sys.path:</code>	第 2 到 3 行：使用 <code>for</code> 迴圈依序讀取模組 <code>sys</code> 的屬性 <code>path</code> 到變數 <code>path</code> ，印出變數 <code>path</code> 到螢幕上。
3	<code>print(path)</code>	

---

## 執行結果

```
K:\mybook\python\ch7  
K:\mybook\python  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python36.zip  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\DLLs  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib  
C:\Users\user\AppData\Local\Programs\Python\Python36-32  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib\site-packages
```

## 7-2 套件

---

- 多個模組( 檔案) 放在同一個資料夾下, 在該資料夾下新增一個檔案, 檔案名稱為「`__init__.py`」, 該資料夾就形成套件。

## 7-2-1 實作套件

---

- 實作一個套件game, 新增dice.py 可以產生擲骰子的點數, 與新增poker.py 可以產生撲克牌的花色與點數, 將這兩個檔案放在資料夾game 下, 在資料夾game 下新增一個檔案「\_\_init\_\_.py」
- 檔案「\_\_init\_\_.py」的內容可以是空的, 資料夾與檔案的關係如下。

## 7-2 套件

---

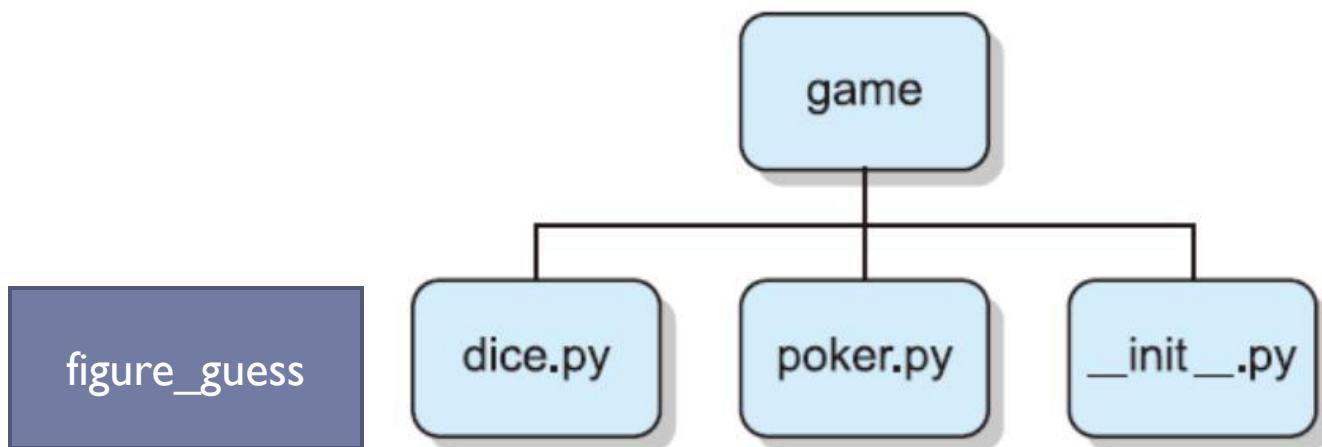


圖 7-1 套件 game 的資料夾關係



---


# Programming codes 2



## 7-2-1 實作套件


---

□ 新增dice.py, 程式碼如下

行號	範例 (  : ch7\game\dice.py)
1	from random import choice
2	def dice():
3	return choice(range(1,7,1))


## 7-2-1 實作套件

□ 新增poker.py, 程式碼如下:

行號	範例 (  : ch7\game\poker.py)
1	from random import choice
2	def poker():
3	a = ['C', 'H', 'D', 'S']
4	b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 'T', 'J', 'Q', 'K']
5	return choice(a)+str(choice(b))

## 7-2-2 套件的使用

---

行號	範例 (  : ch7\7-2-2-pkg.py)	執行結果
1	from game import dice, poker	2
2	for i in range(2):	SQ
3	print(dice.dice())	5
4	print(poker.poker())	H1

---

# Readings



## 7-3 腳本程式


---

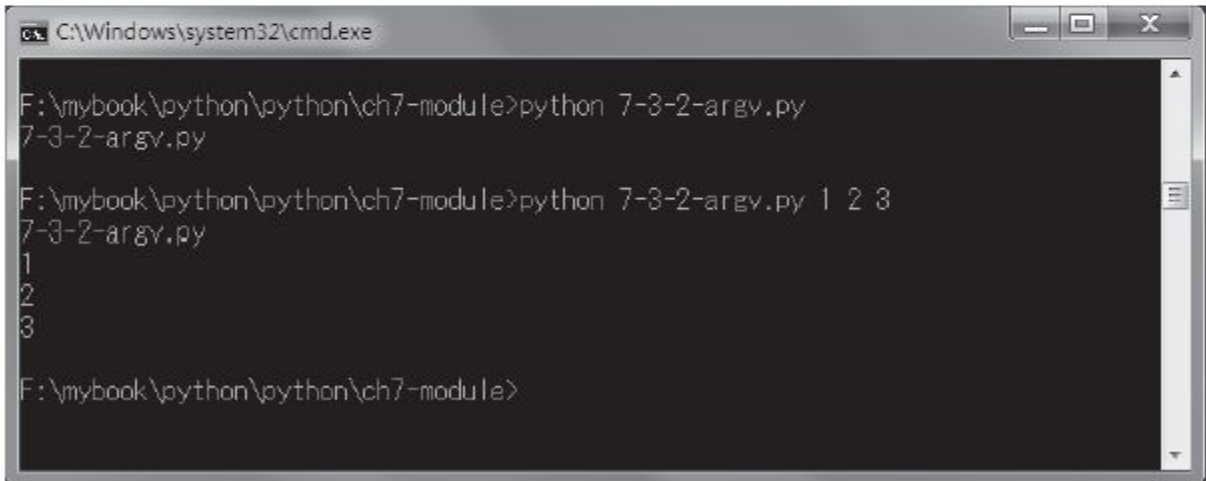
- ❑ 腳本程式可以當成模組被匯入，也可以成為獨立執行的腳本程式，使用「`if __name__ == '__main__':`」
- ❑ 串接在此判斷條件後的程式碼，在腳本程式被當成模組被匯入時，不會被執行，只有在腳本程式獨立執行時才會執行。

## 7-3-1 實作腳本程式

---

- 執行Python 腳本程式時，可以於執行腳本程式命令列後方加入引數，例如：「python 7-3-2-argv.py 1 2 3」

指令列引數範例 (  : ch7\7-3-2-argv.py) , 如下。

行號	範例	執行結果
1 2 3	<pre>import sys for i in sys.argv:     print(i)</pre>	<p>在命令列使用「python」獨立執行 Python 腳本程式，接著在 Python 腳本程式後方加入引數，第一次執行時不加入引數，第二次執行後方加入引數「1 2 3」，執行結果如下圖。</p>  <p>The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The prompt is at 'F:\mybook\python\python\ch7-module&gt;'. The first command entered is 'python 7-3-2-argv.py', which results in the output '7-3-2-argv.py'. The second command entered is 'python 7-3-2-argv.py 1 2 3', which results in the output '1', '2', and '3' on separate lines.</p>



## Ch8 類別與例外

---

# Programming codes I

(practice@home)



## 8-1 類別

---

- 類別像是一個蛋糕的模子，這個蛋糕的模子可以重複製作出相同的蛋糕，就像類別可以宣告出相同的物件，可以讓程式不斷地被重複利用。
- 在Python 使用class 宣告類別，就可以重複宣告此類別的物件，以下進行類別的實作。


## 8-1-1 實作類別

---

- Python 中類別就是使用class 定義類別內的資料與操作的方法
- 方法「`__init__`」表示宣告類別時會自動執行的方法，第一個參數為self, 表示自己，第二個參數為輸入類別的資料，可以在宣告屬於該類別的物件時，同時傳入資料到該物件，傳入的資料可以指定給「`self. 變數名稱`」，表示該物件有了儲存資料的變數。

## 8-1-1 實作類別

---

行號	範例 (  : ch8\8-1-1- 實作類別 .py)	執行結果
1	<code>class Animal():</code>	動物
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>a = Animal(' 動物 ')</code>	
5	<code>print(a.name)</code>	

---

# Programming codes 2

(practice@home)



## 8-1-2 繼承

---

- 可以繼承原有的類別，修改延伸出新的類別，原有的類別被稱為**基礎類別(baseclass)**或**雙親類別(parent class)**
- 新的類別被稱為**衍生類別(derived class)**或**子類別(childclass)**，這個衍生類別就自動擁有基礎類別的變數與方法。

## 8-1-2 繼承


---

- 使用「**class 衍生類別( 基礎類別)**」來定義類別間的繼承關係，衍生類別就繼承了基礎類別
- 在衍生類別中使用「**super(). 基礎類別的方法**」可以呼叫基礎類別的方法來幫忙，若衍生類別所需要的功能已經在基礎類別定義過了，就可以呼叫基礎類別幫忙，重複利用已經撰寫過的程式碼。



- 
- 當所有衍生類別都需要更改，而且所需功能都相同時，若該功能可以更改基礎類別達成，就直接修改基礎類別，就會影響所有的衍生類別。
  - 利用衍生類別與基礎類別的關係達成程式碼的重複利用，減少程式的錯誤，發揮物件導向程式設計的優點。

## 8-1-2 繼承

行號	範例 (  : ch8\8-1-2- 繼承 .py)	執行結果
1	<code>class Animal():</code>	動物 小狗小白
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>class Dog(Animal):</code>	
5	<code>    def __init__(self, name):</code>	
6	<code>        super().__init__(' 小狗 '+name)</code>	
7	<code>a = Animal(' 動物 ')</code>	
8	<code>d = Dog(' 小白 ')</code>	
9	<code>print(a.name)</code>	
10	<code>print(d.name)</code>	


## 8-1-3 覆寫方法

---

- ▣ 衍生類別可以繼承基礎類別的資料與方法，在衍生類別內重新改寫基礎類別的方法，讓衍生類別與基礎類別的相同方法有不同功能，這樣的改寫方法稱作「**覆寫**」。

- 
- 以下類別Dog 繼承了類別Animal, 兩個類別都有方法sound, 類別Animal 的方法sound 不執行任何動作, 而類別Dog 繼承類別Animal
  - 所以有了方法sound, 但在類別Dog 中重新覆寫方法sound, 讓類別Dog 的方法sound 會印出「汪汪叫」在螢幕上。

## 8-1-3 覆寫函式


行號	範例 (  : ch8\8-1-3- 覆寫函式 .py)	執行結果
1	<code>class Animal():</code>	小狗小黑 汪汪叫
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>    def sound(self):</code>	
5	<code>        pass</code>	
6	<code>class Dog(Animal):</code>	
7	<code>    def __init__(self, name):</code>	
8	<code>        super().__init__(' 小狗 '+name)</code>	
9	<code>    def sound(self):</code>	
10	<code>        return ' 汪汪叫 '</code>	
11	<code>d = Dog(' 小黑 ')</code>	
12	<code>print(d.name)</code>	
13	<code>print(d.sound())</code>	

## 8-1-4 新增參數的覆寫方法

---

- 在覆寫方法時，可以新增參數。
- 以下類別Dog 繼承了類別Animal，兩個類別都有方法「\_\_init\_\_」，類別Dog 繼承類別Animal，所以有了方法「\_\_init\_\_」，但在類別Dog 中重新覆寫方法「\_\_init\_\_」時，多出了參數leg，使用self.leg 儲存參數leg。

## 8-1-4 新增參數的覆寫函式

行號	範例 (  : ch8\8-1-4- 新增參數的覆寫函式 .py)	執行結果
1	<code>class Animal():</code>	小狗小黑 有 4 條腿 小狗小黑 汪汪叫
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>    def sound(self):</code>	
5	<code>        pass</code>	
6	<code>class Dog(Animal):</code>	
7	<code>    def __init__(self, name, leg):</code>	
8	<code>        super().__init__(' 小狗 '+name)</code>	
9	<code>        self.leg = leg</code>	
10	<code>    def sound(self):</code>	
11	<code>        return ' 汪汪叫 '</code>	
12	<code>d = Dog(' 小黑 ', 4)</code>	
13	<code>print(d.name, ' 有 ', d.leg, ' 條腿 ')</code>	
14	<code>print(d.name, d.sound())</code>	

## 8-1-5 新增方法

---

- ▣ 衍生類別可以繼承基礎類別的資料與方法，在衍生類別內新增基礎類別沒有的方法。
- ▣ 以下類別Dog 繼承了類別Animal，兩個類別都有方法「\_\_init\_\_」與方法sound，在衍生類別Dog 新增方法move，而基礎類別Animal 沒有方法move。



## 8-1-5 新增方法/函式

行號	範例 (  : ch8\8-1-5- 新增函式 .py)	執行結果
1	<code>class Animal():</code>	小狗小黑 汪汪叫 小狗小黑在馬路上行走
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>    def sound(self):</code>	
5	<code>        pass</code>	
6	<code>class Dog(Animal):</code>	
7	<code>    def __init__(self, name):</code>	
8	<code>        super().__init__(' 小狗 '+name)</code>	
9	<code>    def sound(self):</code>	
10	<code>        return ' 汪汪叫 '</code>	
11	<code>    def move(self):</code>	
12	<code>        print(self.name + ' 在馬路上行走 ')</code>	
13	<code>d = Dog(' 小黑 ')</code>	
14	<code>print(d.name, d.sound())</code>	
15	<code>d.move()</code>	

---

# Programming codes 3



## 8-1-6 多型(polymorphism)


---

- ▣ 多個類別可以定義相同的方法名稱，而相同方法名稱在不同類別可以定義各自特有的功能
- ▣ 經由呼叫物件的方法名稱，傳入不同的物件都定義此相同方法名稱而產生不同的功能，稱作「多型」，在 Python 裡這些類別不一定要有繼承關係。

- 
- 以下範例，類別Dog 繼承了類別Animal，兩個類別都有方法who 與方法sound，類別Bird 沒有繼承類別Animal，但有定義相同名稱的方法who 與方法sound

- 
- 所以類別Animal、類別Dog 與類別Bird 就可以實作多型的概念，定義一個函式talk 可以輸入物件，顯示輸入物件的方法who 與方法sound 的回傳結果，經由輸入不同的物件觀察顯示在螢幕上的字串，可以了解多型的概念。

## 8-1-6 多型(polymorphism)

行號	範例 (  : ch8\8-1-6.py)	執行結果
1	<code>class Animal():</code>	<p>動物 正在 None 小狗小黑 正在 汪汪叫 小鳥小黃 正在 啾啾叫</p>
2	<code>    def __init__(self, name):</code>	
3	<code>        self.name = name</code>	
4	<code>    def who(self):</code>	
5	<code>        return self.name</code>	
6	<code>    def sound(self):</code>	
7	<code>        pass</code>	
8	<code>class Dog(Animal):</code>	
9	<code>    def __init__(self, name):</code>	
10	<code>        super().__init__(' 小狗 '+name)</code>	
11	<code>    def sound(self):</code>	
12	<code>        return ' 汪汪叫 '</code>	
13	<code>class Bird():</code>	
14	<code>    def __init__(self, name):</code>	

## 8-1-6 多型(polymorphism)

```
15     self.name = '小鳥'+name
16     def who(self):
17         return self.name
18     def sound(self):
19         return '啾啾叫'
20 def talk(obj):
21     print(obj.who(), '正在', obj.sound())
22 a = Animal('動物')
23 talk(a)
24 d = Dog('小黑')
25 talk(d)
26 b = Bird('小黃')
27 talk(b)
```

---

# Readings





## 8-1-7 類別內無法直接存取的變數

---

- 在類別內的變數，若在變數名稱前加上「\_\_」，該變數無法直接使用「類別物件.\_\_變數名稱」進行存取，達到資料保護的目的，需要在類別內定義方法，回傳「self.\_\_變數名稱」才能存取到該變數。

## 8-1-7 類別內無法直接存取的變數

行號	範例 (  : ch8\8-1-7- 類別內無法直接存取的變數 .py)	執行結果
1	class Animal():	小狗小黑 有 4 條腿
2	def __init__(self, name):	
3	self.__name = name	
4	def sound(self):	
5	pass	
6	def show_name(self):	
7	return self.__name;	
8	class Dog(Animal):	
9	def __init__(self, name, leg):	
10	super().__init__('小狗'+name)	
11	self.leg = leg	
12	def sound(self):	
13	return '汪汪叫 '	
14	d = Dog('小黑 ', 4)	
15	#print(d.__name,'有 ', d.leg, ' 條腿 ')	
16	print(d.show_name(), '有 ', d.leg, ' 條腿 ')	

## 8-1-8 特殊方法(special method)


---

- 存在於類別內的特殊方法, Python 會讓運算子或內建函式可以與特殊方法自動對應

- 
- 例如判斷兩物件是否相等的運算子「==」會自動與類別內特殊方法「\_\_eq\_\_」
  - 所以在類別內重新定義特殊方法「\_\_eq\_\_」, 類別中使用運算子「==」的運算就會直接使用特殊方法「\_\_eq\_\_」進行是否相等的判斷

- 
- 以下程式範例就重新改寫特殊方法「`__eq__`」，判斷兩個物件內的變數`name` 是否相同，決定兩物件是否相等。

## 8-1-8 特殊函式(special method)

行號	範例 (  : ch8\8-1-8.py)	執行結果
1	class Animal():	
2	def __init__(self, name):	
3	self.__name = name	
4	def sound(self):	
5	pass	
6	def show_name(self):	True
7	return self.__name	True
8	def eq(self, other):	False
9	return self.__name == other.show_name()	False
10	def __eq__(self, other):	
11	return self.__name == other.show_name()	
12	class Dog(Animal):	
13	def __init__(self, name, leg):	
14	super().__init__('小狗'+name)	

## 8-1-8 特殊函式(special method)

---

15	self.leg = leg	
16	def sound(self):	
17	return '汪汪叫'	
18	d1 = Dog('小黑', 4)	
19	d2 = Dog('小黑', 4)	
20	print(d1.eq(d2))	
21	print(d1 == d2)	
22	d3 = Dog('小白', 4)	
23	print(d1.eq(d3))	
24	print(d1 == d3)	

## 8-1-8 特殊函式(special method)

特殊函式與運算子的對應，如下表。

表 8-1 比較運算函式對應的運算子

	特殊函式	對應的運算子
比較運算	<code>__eq__(self, other)</code>	<code>self == other</code>
	<code>__ne__(self, other)</code>	<code>self != other</code>
	<code>__gt__(self, other)</code>	<code>self &gt; other</code>
	<code>__ge__(self, other)</code>	<code>self &gt;= other</code>
	<code>__lt__(self, other)</code>	<code>self &lt; other</code>
	<code>__le__(self, other)</code>	<code>self &lt;= other</code>



## 8-1-8 特殊函式(special method)

表 8-2 算術與邏輯運算函式對應的運算子

	特殊函式	對應的運算子
算術與邏輯運算	<code>__add__(self, other)</code>	<code>self + other</code>
	<code>__sub__(self, other)</code>	<code>self - other</code>
	<code>__mul__(self, other)</code>	<code>self * other</code>
	<code>__truediv__(self, other)</code>	<code>self / other</code>
	<code>__floordiv__(self, other)</code>	<code>self // other</code>
	<code>__mod__(self, other)</code>	<code>self % other</code>
	<code>__pow__(self, other)</code>	<code>self ** other</code>
	<code>__lshift__(self, other)</code>	<code>self &lt;&lt; other</code>
	<code>__rshift__(self, other)</code>	<code>self &gt;&gt; other</code>
	<code>__and__(self, other)</code>	<code>self &amp; other</code>
	<code>__or__(self, other)</code>	<code>self   other</code>
	<code>__xor__(self, other)</code>	<code>self ^ other</code>

## 8-1-8 特殊函式(special method)

---

表 8-3 内建函式對應的運算子

	特殊函式	對應的函式
内建函式	<code>__len__(self)</code>	<code>len(self)</code>
	<code>__str__(self)</code>	<code>str(self)</code>
	<code>__repr__(self)</code>	<code>repr(self)</code>

## 8-1-9 組合(composition)

---

- 類別與類別之間不全然都是繼承關係，也有可能是類別A 是類別B 的一部分，腳是動物的一部分
- 但腳不是動物，腳無法繼承動物，這時就可以使用組合，在動物類別初始化時，將腳當成參數傳入，讓腳成為動物的一部分。

## 8-1-9 組合(composition)

行號	範例 (  : ch8\8-1-9- 組合 .py)	執行結果
1	class Leg():	小狗 有 4 隻 短短的 腿
2	def __init__(self, num, look):	
3	self.num = num	
4	self.look = look	
5	class Animal():	
6	def __init__(self, name, leg):	
7	self.__name = name	
8	self.leg = leg	
9	def show_name(self):	
10	return self.__name	
11	def show(self):	
12	print(self.show_name(), ' 有 ', self.leg.num, ' 隻 ', self.leg.look, ' 腿 ')	
13	leg = Leg(4, ' 短短的 ')	
14	a = Animal(' 小狗 ', leg)	
15	a.show()	

- 
- 類別方法(class method) 作用對象為類別，會影響整個類別，也會影響類別所產生的物件，類別方法的第一個參數通常取名為cls，需在類別中方法的前一行使用裝飾器「**@classmethod**」，這樣的方法稱作「**類別方法**」。

## 8-1-10 類別方法


行號	範例 (  : ch8\8-1-10- 類別方法 .py)	執行結果
1	<code>class Animal():</code>	<div>現在有 1 隻動物</div> <div>現在有 2 隻動物</div> <div>現在有 3 隻動物</div> <div>現在有 2 隻動物</div>
2	<code>    count = 0</code>	
3	<code>    def __init__(self):</code>	
4	<code>        Animal.count += 1</code>	
5	<code>    def kill(self):</code>	
6	<code>        Animal.count -= 1</code>	
7	<code>    @classmethod</code>	
8	<code>    def show_count(cls):</code>	
9	<code>        print(' 現在有 ',cls.count,' 隻動物 ')</code>	
10	<code>a = Animal()</code>	
11	<code>Animal.show_count()</code>	
12	<code>b = Animal()</code>	
13	<code>Animal.show_count()</code>	
14	<code>c = Animal()</code>	
15	<code>Animal.show_count()</code>	
16	<code>a.kill()</code>	
17	<code>Animal.show_count()</code>	

## 8-1-11 靜態方法

---

- ▣ 靜態方法(`static method`) 讓類別不需要建立物件, 就可以直接使用該類別的靜態方法, 需在類別中方法的前一行使用裝飾器「`@staticmethod`」。

---

行號	範例 (  : ch8\8-1-11- 靜態方法 .py)	執行結果
1	class Say():	Hello
2	@staticmethod	
3	def hello():	
4	print('Hello')	
5	Say.hello()	



## 8-2 例外(exception)

---


- 在執行程式的過程中產生錯誤，程式會中斷執行，發出例外訊息，以下介紹例外的程式區塊，與實作自訂的例外類別。

## 8-2-1 try-except

---

- 使用程式區塊「try...except...」可以攔截例外，在try 區塊中撰寫可能發生錯誤的程式，若發生錯誤，則會跳到except 區塊執行進行後續的處理。

## 8-2-1 try-except

行號	範例 (  : ch8\8-2-1-try-except.py)	執行結果
1	try:	輸入數字與字元則不會發生錯誤，密碼會儲存在變數 pwd，若輸入「ctrl+D」，則顯示「發生錯誤」。  (1) 輸入數字與字元 請輸入密碼 abc123  (2) 輸入「Ctrl+D」 請輸入密碼 ^D 發生錯誤
2	pwd = input(' 請輸入密碼 ')	
3	except:	
4	print(' 發生錯誤 ')	

## 8-2-2 try-except-else

- `except` 後可以接指定錯誤類型，常見錯誤類型，如下表。

表 8-4 常見的錯誤類型

錯誤類型	說明
KeyboardInterrupt	當使用者輸入中斷 (Ctrl+C) 時，發出此錯誤。
ZeroDivisionError	除以 0 時，發出此錯誤。
EOFError	接受到 EOF(end of file) 訊息時，發出此錯誤。
NameError	區域變數或全域變數找不到時，發出此錯誤。
OSError	與作業系統有關的錯誤
FileNotFoundError	檔案或資料夾找不到時，發出此錯誤。
ValueError	輸入資料與程式預期輸入資料型別不同時，發出此錯誤。


## 8-2-3 try-except-as-else

---

- 使用程式區塊「try...except...as...else...」可以攔截例外，在try 區塊中撰寫可能發生錯誤的程式，若發生錯誤，則會跳到except 區塊進行後續的處理

- 
- 在`except` 後面接上`as` 就會將錯誤類別轉換成對應的錯誤類別物件, `except` 區塊個數可以有許多個, 區分各種錯誤的類型, `except` 區塊內撰寫對應的錯誤處理程式; 若沒有發生錯誤, 則會跳到`else` 區塊執行。

## 8-2-3 try-except-as-else

行號	範例 (  : ch8\8-2-3-try-except-as-else.py)
1	try:
2	num = int(input(' 請輸入整數 '))
3	except EOFError:
4	print(' 輸入 EOF')
5	except ValueError as ve:
6	print(' 發生 ValueError 錯誤 ',ve)
7	except Exception as e:
8	print(' 發生其他錯誤 ',e)
9	else:
10	print(' 輸入整數為 ', num)

---

### (1) 輸入數字

請輸入整數 123

輸入整數為 123

### (2) 輸入「Ctrl+D」相當於輸入 EOF

請輸入整數 ^D

輸入 EOF

### (3) 輸入「abc」

請輸入整數 abc

發生 ValueError 錯誤 invalid literal for int() with base 10: 號 abc 號




## 8-2-4 try-except-as-else 與自訂例外類別

---

- 可以自訂例外類別，自訂例外類別需要繼承系統例外類別`Exception`，該類別就會成為例外類別，可以傳入參數到自訂例外類別，將錯誤資訊儲存在自訂例外類別，使用指令`raise` 發出例外，接著由`except` 進行例外處理。

## 8-2-4 try-except-as-else 與自訂例外類別

行號	範例 (  : ch8\8-2-4- 自訂例外類別 .py)
1	class PwdException(Exception):
2	def __init__(self,pwd,len):
3	super().__init__(self)
4	self.pwd=pwd
5	self.len=len
6	try:
7	pwd = input(' 請輸入密碼，長度至少 8 個字元 ')
8	if len(pwd) < 8:
9	raise PwdException(pwd,len(pwd))
10	except EOFError:
11	print(' 輸入 EOF')
12	except PwdException as pex:
13	print(' 密碼 ', pex.pwd, ' 長度為 ', pex.len, ' 密碼長度不足 ')
14	else:
15	print(' 輸入密碼為 ', pwd)

---

(1) 輸入密碼長度大於等於 8 個字元

請輸入密碼，長度至少 8 個字元 abcd1234

輸入密碼為 abcd1234

(2) 輸入「Ctrl+D」相當於輸入 EOF

請輸入密碼，長度至少 8 個字元 ^D

輸入 EOF

(3) 輸入「abc」

請輸入密碼，長度至少 8 個字元 abc

密碼 abc 長度為 3 密碼長度不足

## 8-2-5 try-except-as-else-finally 與自訂例外類別

---

- 使用程式區塊「try...except...as...else...finally...」可以攔截例外, 在try 區塊中撰寫可能發生錯誤的程式

- 
- 若發生錯誤，則會跳到except 區塊進行後續的處理，在except 後面接上as 就會將錯誤類別轉換成對應的錯誤類別物件，except 區塊個數可以有很多個，區分各種錯誤的類型，except 區塊內撰寫對應的錯誤處理程式

- 
- 若沒有發生錯誤，則會跳到`else` 區塊執行，不管有沒有發生錯誤，最後都要執行`finally` 區塊。

## 8-2-5 try-except-as-else-finally 與自訂例外類別

行號	範例 (  : ch8\8-2-5-try-except-as-else-finally.py)
1	class PwdException(Exception):
2	def __init__(self,pwd,len):
3	super().__init__(self)
4	self.pwd=pwd
5	self.len=len
6	try:
7	pwd = input(' 請輸入密碼，長度至少 8 個字元 ')
8	if len(pwd) < 8:
9	raise PwdException(pwd,len(pwd))
10	except EOFError:
11	print(' 輸入 EOF')
12	except PwdException as pex:
13	print(' 密碼 ', pex.pwd, ' 長度為 ', pex.len, ' 密碼長度不足 ')
14	else:
15	print(' 輸入密碼為 ', pwd)
16	finally:
17	print(' 請妥善保管密碼 ')

### (1) 輸入密碼長度大於等於 8 個字元

請輸入密碼，長度至少 8 個字元 abcd1234

輸入密碼為 abcd1234

請妥善保管密碼

### (2) 輸入「Ctrl+D」相當於輸入 EOF

請輸入密碼，長度至少 8 個字元 ^D

輸入 EOF

請妥善保管密碼

### (3) 輸入「abc」

請輸入密碼，長度至少 8 個字元 abc

密碼 abc 長度為 3 密碼長度不足

請妥善保管密碼



---

# □ Exercise




# Exercise 1

---

- ▣ 限定引用programming code1的guess.py模組，製作一個「剪刀、石頭、布」的小遊戲，電腦出拳為隨機出拳，使用者由介面輸入「剪刀」、「石頭」與「布」
- ▣ 比較電腦出拳與使用者出拳決定勝負的結果



行號	範例 (  : ch7\guess2.py)
1	from guess import figure_guess
2	def run():
3	
4	<div data-bbox="237 439 1402 1310"><h3>Exercise I</h3><p>(限定引用programming code1的 guess.py模組;1 pt)</p></div>
5	
6	
7	
8	
9	
10	
11	
12	



```
13 elif my ==  
14     if co  
15     p  
16     elif  
17     p  
18     els  
19     p  
20 else:  
21     if o  
22     p  
23     elif  
24     p  
25     els  
26     p  
27 if __nam  
28     for i in  
29     run  
30 else:  
31     print('我不')
```

## Exercise I

(限定引用programming code1的  
guess.py模組; 1 pt)

## 執行結果

### □ 執行兩次的結果範例

請輸入「剪刀」、「石頭」或「布」？剪刀  
電腦出 布  
玩家獲勝

請輸入「剪刀」、「石頭」或「布」？剪刀  
電腦出 剪刀  
平手



# Exercise

---

近日疫情嚴峻且快篩試劑缺乏，請設計一程式顯示消費者資料(姓名+學號)和藥局快篩剩餘數量(隨機產生0~1000)，並計算出你與藥局之間的距離。

消費者和藥局的位置(範圍介於(0,0)~(10, 10))由電腦隨機產生

**注意: 1.請先使用class宣告 “位置(location)”類別**

**2. “消費者”類別與 “藥局資料” 類別必須 繼承 於 “位置 “類別**

**3.必須將消費者及快篩數量資料寫於類別中，請勿直接print輸出**

```

1  import random
2  import math
3
4  class Location:
5      def __init__(self):
6
7
8
9      def move(self):
10
11
12         return (self.x, self.y)
13
14  class mydata(Location):
15      def __init__(self):
16
17      def nameandID(self):
18
19
20  class drugstore(Location):
21      def __init__(self):
22
23      def count(self):
24
25         return c

```

Exercise

Exercise

Exercise

Exercise

Exercise

Exercise



```

27 def Distance(s, p):
28     d = math.sqrt((s[0] - p[0]) ** 2 + (s[1] - p[1]) ** 2)
29     print('我與藥局之間的距離:', d)
30
31 me = mydata()
32 s1 = me.move()
33 p1 = me.nameandID(s1)
34
35 drug = drugstore()
36 p1 = drug.move()
37
38 drug.count()
39
40
41 Distance(s1, p1)

```

Exercise

Exercise





## 執行結果:

消費者的資料: Josh Zhang 110318xxx  
我的位置: (4, 8)

快篩剩餘數量: 550  
藥局的位置: (3, 4)

我與藥局之間的距離: 4.123105625617661



# To be continued.....

Instructor: Cheng-Chun Chang (張正春)  
Department of Electrical Engineering