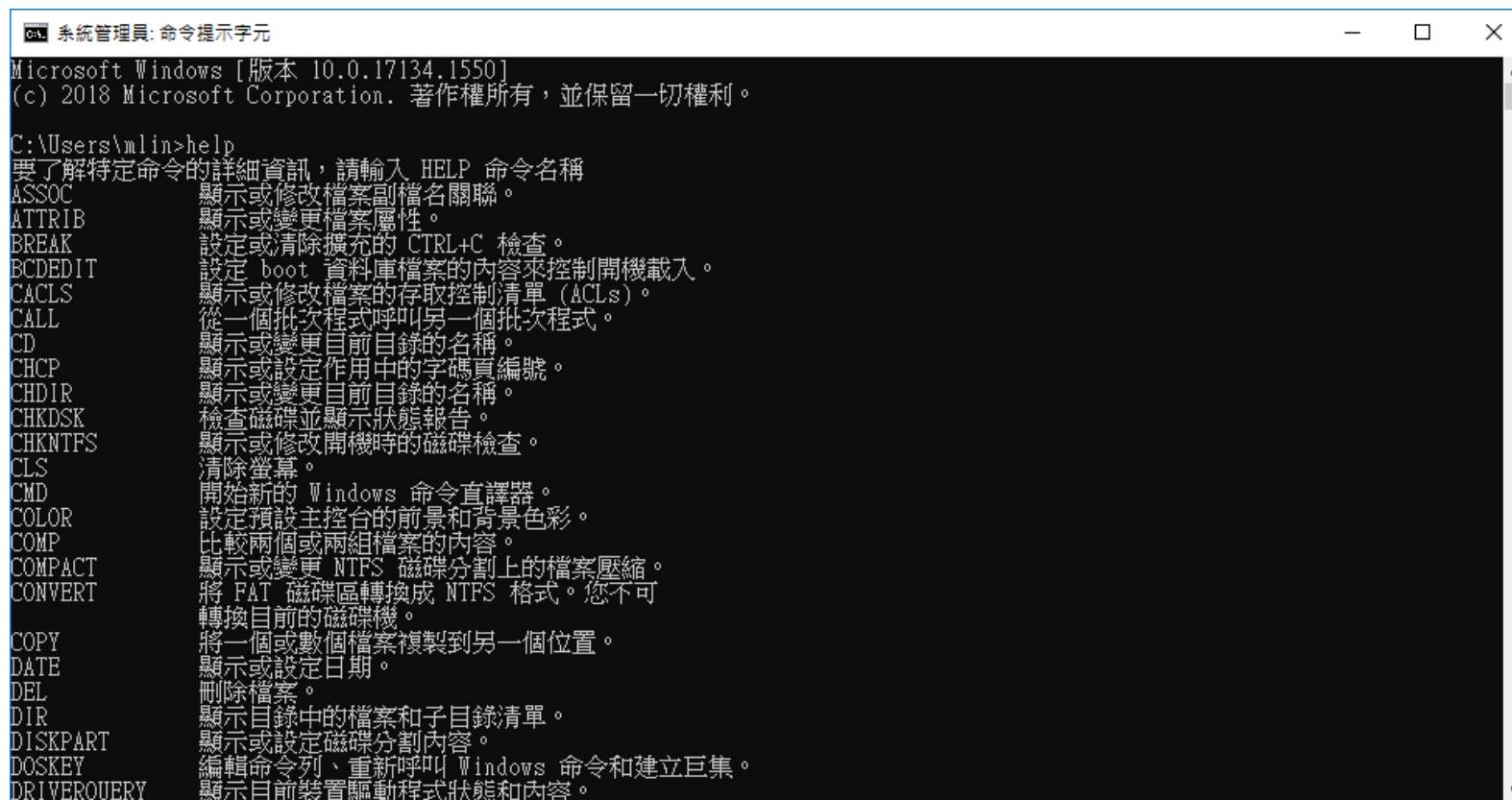
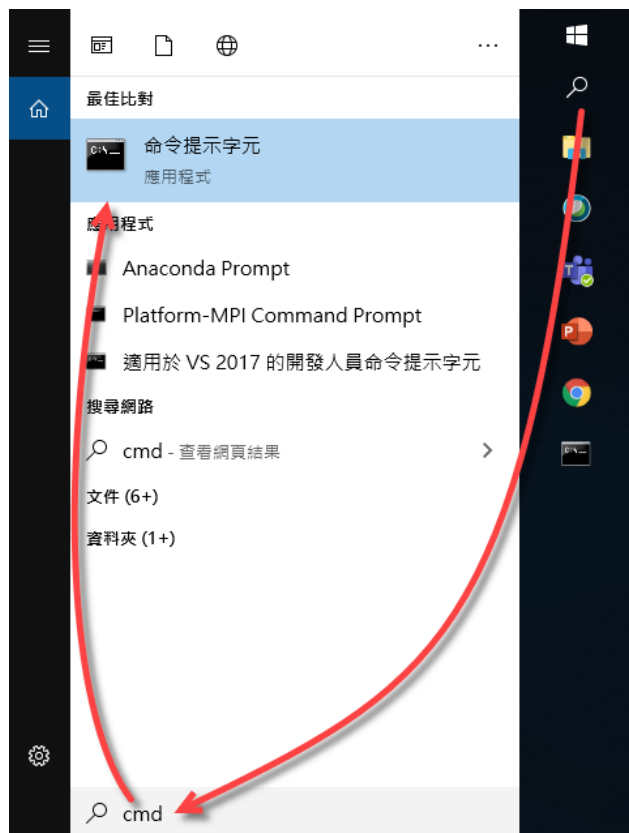


Week 4-透過批次指令達到模擬 自動化

命令提示字元



/ path

- **Path**環境變數對開發自動化程式很重要。當輸入要執行的程式時，如果沒有指定絕對路徑，則作業系統會到**Path**所列的目錄依序尋找。
- 在命令提示字元視窗對**Path**的更改只會作用在該視窗當中
- 拓展路徑：path=%path%;C:\Program Files\AnsysEM\AnsysEM20.2\Win64

```
C:\Users\mlin>path=%path%;C:\Program Files\AnsysEM\AnsysEM20.2\Win64

C:\Users\mlin>path
PATH=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\iCLS\;C:\Program Files\Intel\Intel(R) Management Engine Components\iCLS\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files\PuTTY\;C:\Program Files\Microsoft VS Code\bin;C:\Cadence\SPB_17.2\tools\bin;C:\Program Files\Google\Google Apps Sync\;%CDSROOT%\tools\bin;C:\Cadence\SPB_17.2\openaccess\bin\win32\opt;C:\Cadence\SPB_17.2\tools\specctra\bin;C:\Cadence\SPB_17.2\tools\libutil\bin;C:\Cadence\SPB_17.2\tools\bin;C:\Cadence\SPB_17.2\openaccess\bin\x64\opt;C:\Cadence\SPB_17.2\tools\capture;C:\Cadence\SPB_17.2\tools\pspice;C:\Cadence\SPB_17.2\tools\spectre\bin;C:\Cadence\SPB_17.2\tools\fet\bin;C:\Cadence\SPB_17.2\libutil\capture;C:\Cadence\SPB_17.2\tools\pcb\bin;C:\Users\mlin\AppData\Local\Microsoft\WindowsApps;C:\Cadence\SPB_16.6\tools\bin;C:\Program Files\AnsysEM\AnsysEM19.3\Win64\common\IronPython;C:\Program Files\AnsysEM\AnsysEM20.2\Win64\common\IronPython;C:\Users\mlin\AppData\Local\GitHubDesktop\bin;C:\Program Files\AnsysEM\AnsysEM20.2\Win64;C:\Program Files\AnsysEM\AnsysEM20.2\Win64

C:\Users\mlin>
```

AEDT環境有以下幾支程式可以支援命令列的執行方式

- nexxim.exe：電路模擬程式，輸出結果儲存在.sdf當中
- sdf2csv.exe：將.sdf格式轉換成.csv格式
- PinToPinSetup.exe：根據設定檔產生.aedb資料庫，主要用在PCB或封裝的設定
- ansysedt.exe：aedt主程式，搭配Python可以建立模型，執行模擬以及輸出模擬資料。其中包含5種模式：
 - BatchSave
 - BatchSolve
 - BatchExtract
 - RunScript
 - RunScriptandExit

Python的命令列操作指令

除了在作業系統的控制窗輸入指令，或者執行批次檔來啟動應用程式，我們也可以用Python來模擬作業系統命令列的操作。用Python的好處是其提供的功能比作業系統支援的指令集更彈性，更強大，比方說是字串的操控。Python模擬作業系統的指令包括：

- `os.environ['path'] += ';C:\Program Files\AnsysEM\AnsysEM20.2\Win64'`
- `os.system()`
- `subprocess.run()`
- `subprocess.check_call()`

範例解說：使用nexxim.exe執行netlist

定義需求

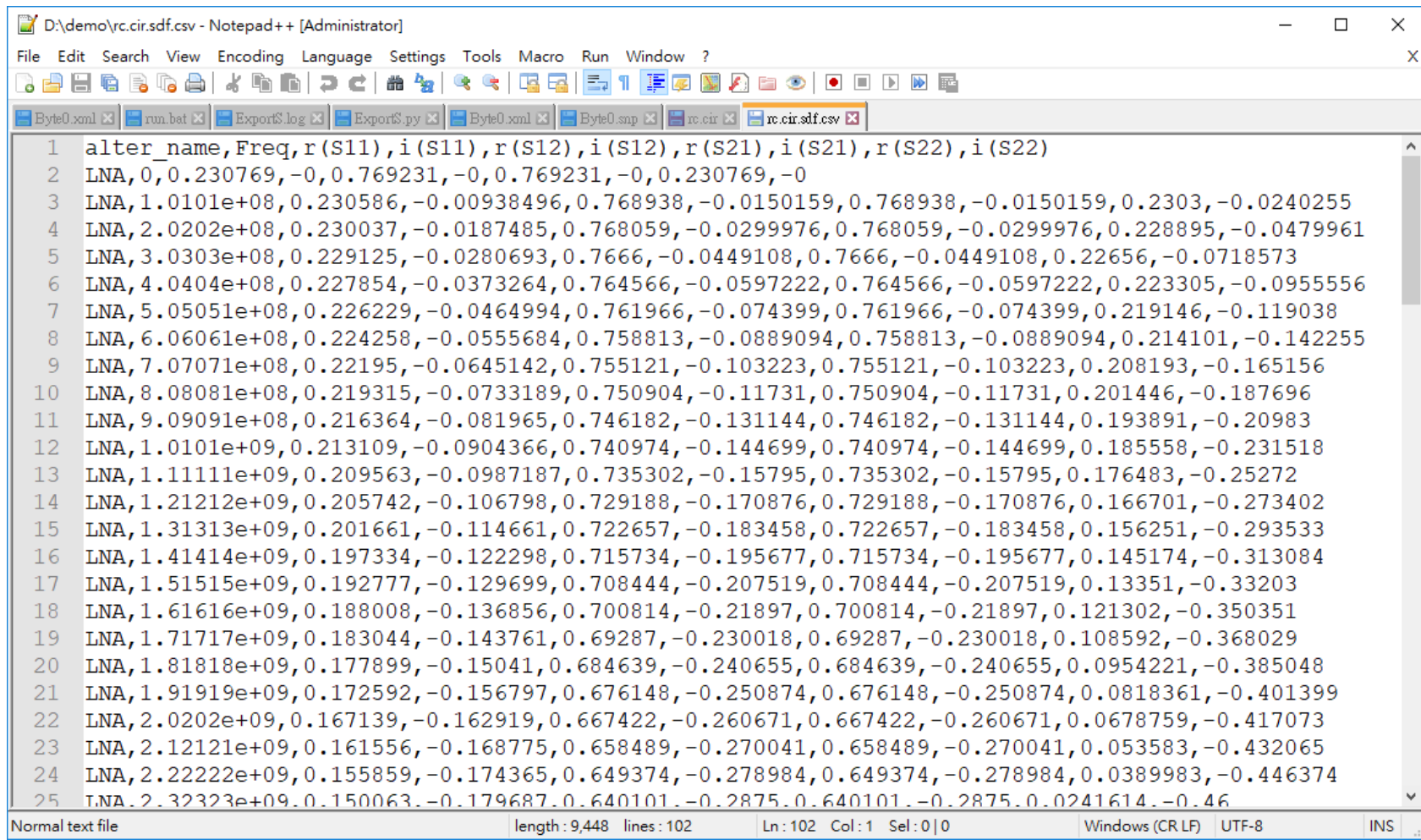
- 需求
 - 快速更改電路設定
 - 輸出模擬結果
- 輸入
 - 網表
 - 參數
- 輸出
 - S參數



修改網表並執行模擬及檔案轉換

```
1 import os
2 text = ''.option PARHIER='local'
3 .option max_messages=1
4
5 R2 Port1 Port2 {r}
6 C4 0 Port2 {c}
7 RPort1 Port1 0 PORTNUM=1 RZ=50 IZ=0
8 .PORT Port1 0 1 RPort1
9 RPort2 Port2 0 PORTNUM=2 RZ=50 IZ=0
10 .PORT Port2 0 2 RPort2
11
12 .LNA
13 + LIN 100 0 10000000000
14 + FLAG='LNA'
15
16 .end''.format(r=30, c=1e-12)
17
18 with open('rc.cir','w') as f:
19     f.write(text)
20
21 os.environ['path'] += ';C:\Program Files\AnsysEM\AnsysEM20.2\Win64'
22 os.system('nexxim rc.cir')
23 os.system('sdf2csv rc.cir.sdf')
```


輸出之CSV檔



```
D:\demo\rc.cir.sdf.csv - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Window ?
Byte0.xml x run.bat x Export$.log x Export$.py x Byte0.xml x Byte0.snp x rc.cir x rc.cir.sdf.csv x
1 alter_name,Freq,r(S11),i(S11),r(S12),i(S12),r(S21),i(S21),r(S22),i(S22)
2 LNA,0,0.230769,-0,0.769231,-0,0.769231,-0,0.230769,-0
3 LNA,1.0101e+08,0.230586,-0.00938496,0.768938,-0.0150159,0.768938,-0.0150159,0.2303,-0.0240255
4 LNA,2.0202e+08,0.230037,-0.0187485,0.768059,-0.0299976,0.768059,-0.0299976,0.228895,-0.0479961
5 LNA,3.0303e+08,0.229125,-0.0280693,0.7666,-0.0449108,0.7666,-0.0449108,0.22656,-0.0718573
6 LNA,4.0404e+08,0.227854,-0.0373264,0.764566,-0.0597222,0.764566,-0.0597222,0.223305,-0.0955556
7 LNA,5.0505e+08,0.226229,-0.0464994,0.761966,-0.074399,0.761966,-0.074399,0.219146,-0.119038
8 LNA,6.0606e+08,0.224258,-0.0555684,0.758813,-0.0889094,0.758813,-0.0889094,0.214101,-0.142255
9 LNA,7.0707e+08,0.22195,-0.0645142,0.755121,-0.103223,0.755121,-0.103223,0.208193,-0.165156
10 LNA,8.0808e+08,0.219315,-0.0733189,0.750904,-0.11731,0.750904,-0.11731,0.201446,-0.187696
11 LNA,9.0909e+08,0.216364,-0.081965,0.746182,-0.131144,0.746182,-0.131144,0.193891,-0.20983
12 LNA,1.0101e+09,0.213109,-0.0904366,0.740974,-0.144699,0.740974,-0.144699,0.185558,-0.231518
13 LNA,1.1111e+09,0.209563,-0.0987187,0.735302,-0.15795,0.735302,-0.15795,0.176483,-0.25272
14 LNA,1.2121e+09,0.205742,-0.106798,0.729188,-0.170876,0.729188,-0.170876,0.166701,-0.273402
15 LNA,1.3131e+09,0.201661,-0.114661,0.722657,-0.183458,0.722657,-0.183458,0.156251,-0.293533
16 LNA,1.4141e+09,0.197334,-0.122298,0.715734,-0.195677,0.715734,-0.195677,0.145174,-0.313084
17 LNA,1.5151e+09,0.192777,-0.129699,0.708444,-0.207519,0.708444,-0.207519,0.13351,-0.33203
18 LNA,1.6161e+09,0.188008,-0.136856,0.700814,-0.21897,0.700814,-0.21897,0.121302,-0.350351
19 LNA,1.7171e+09,0.183044,-0.143761,0.69287,-0.230018,0.69287,-0.230018,0.108592,-0.368029
20 LNA,1.8181e+09,0.177899,-0.15041,0.684639,-0.240655,0.684639,-0.240655,0.0954221,-0.385048
21 LNA,1.9191e+09,0.172592,-0.156797,0.676148,-0.250874,0.676148,-0.250874,0.0818361,-0.401399
22 LNA,2.0202e+09,0.167139,-0.162919,0.667422,-0.260671,0.667422,-0.260671,0.0678759,-0.417073
23 LNA,2.1212e+09,0.161556,-0.168775,0.658489,-0.270041,0.658489,-0.270041,0.053583,-0.432065
24 LNA,2.2222e+09,0.155859,-0.174365,0.649374,-0.278984,0.649374,-0.278984,0.0389983,-0.446374
25 LNA,2.3232e+09,0.150063,-0.179687,0.640101,-0.2875,0.640101,-0.2875,0.0241614,-0.46
Normal text file length: 9,448 lines: 102 Ln: 102 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS
```

**範例解說：使用ansysedt.exe
加上PinToPinSetup.exe完成
PCB模擬設定**

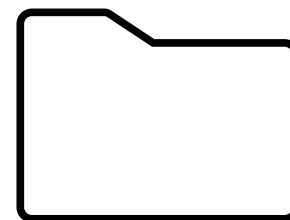
定義需求

- 需求
 - 快速完成不同匯流排的模擬設定
 - 批次完成模擬
- 輸入
 - PCB版
 - 模擬設定條件
- 輸出
 - 匯流排的S參數模型

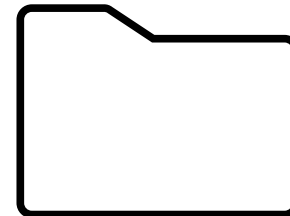


程式開發流程

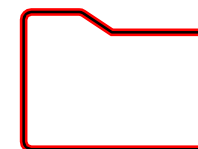
- 將brd轉換成全版aedt檔案
- 修正Stackup
- 針對不同匯流排準備不同.xml設定檔
- 執行PinToPinSetup根據不同設定檔產生aedt
- 執行ansysedt.exe模擬
- 輸出S參數模型



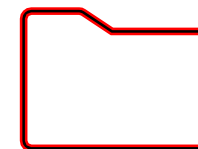
run.bat
pcb.aedb
ExportS.py



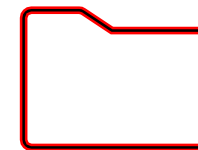
A.xml
B.xml
C.xml



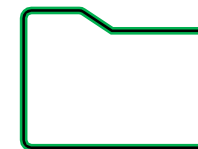
A.aedb



B.aedb



C.aedb



A.SNP
B.SNP
C.SNP

批次檔(run.bat)內容

```
path = %path%;C:\Program Files\AnsysEM\AnsysEM20.1\Win64  
PinToPinSetup.exe .\TestCase .\TestCase\Byte0.xml .\TestCase\Byte0.aedb  
ansysedt.exe -ng -monitor -batchsolve -machinelist  
num=4 .\TestCase\Byte0.aedb  
ansysedt.exe -ng -batchextract ExportS.py TestCase\Byte0.aedt
```

ExportS.py


```
import ScriptEnv, os
ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")
oDesktop.RestoreWindow()
oProject = oDesktop.GetActiveProject()
oDesign = oProject.GetActiveDesign()
name=oProject.GetName()
prj_dir=oProject.GetPath()
data_dir=prj_dir+'/'+'data'
if not os.path.exists(data_dir):
    os.makedirs(data_dir)


try:
    oDesign.ExportNetworkData("", ["HFSS Setup 1:Sweep1"], 3, "{}/{}.snp".format(data_dir,name), ["All"], True, 50, "S", -1, 0, 15,
    True, True, False)
    oDesign.ExportProfile("HFSS Setup 1", "", "{}/{}.prof".format(data_dir,name))
except:
    oDesign.ExportNetworkData("", ["SIwave Setup 1:Sweep1"], 3, "{}/{}.snp".format(data_dir,name), ["All"], True, 50, "S", -1, 0, 15,
    True, True, False)
    oDesign.ExportProfile("SIwave Setup 1", "", "{}/{}.prof".format(data_dir,name))
```

Python基本語法簡介(4/4)

 Python - Functions

 Python - Modules

 Python - Files I/O

 Python - Exceptions

Python Advanced Tutorial

 Python - Classes/Objects

- Defining a Function
 - Scope of Variables
 - Global vs. Local variables
- Calling a Function
 - Pass by reference vs value
 - Function Arguments
 - Keyword arguments
 - Default arguments
- Overview of OOP Terminology
 - Creating Classes
 - Creating Instance Objects
 - Accessing Attributes

使用函式的時機為何？

函式將一組完整功能的代碼包裹起來，僅透過參數傳遞資料，並在完成計算之後回傳結果。其功能主要有下列幾點：

- 需要重複使用某一段程式碼時
- 需要分隔功能，提高程式碼的可讀性時
- 需要限縮變量的範圍時（可簡化變數命名工作）
- 以def定義函式，return返回計算結果，例如：

```
def myadd(x, y):  
    return x+y
```



函式參數

參數可以是數值，字串，list，tuple，dictionary，函式或是自定義物件。此外參數可以有不同宣告方式：

- **Keyword arguments**

- 使用函式時，參數名可連同參數值一同輸入。可提高代碼的可讀性

- **Default arguments**

- 不常修改的參數可事先賦予預設值。在呼叫時便可以省去輸入的功夫。

```
>>> def mysub(x, y):  
...     return x-y  
...  
>>> mysub(3, 1)  
2  
>>> mysub(x=3, y=1)  
2  
>>> mysub(y=1, x=3)  
2  
>>> def mysub(x, y=1):  
...     return x-y  
...  
>>> mysub(3)  
2  
>>> mysub(3, 2)  
1  
>>> |
```

區域變數與全域變數

- 全域變數定義在函式之外，區域變數定義在函式之內。
- 函式裡面讀取的變數若沒有定義在函數裡面，則會嘗試到函式外部尋找全域變數
- 函式內部如果要修改全域變數需先宣告為 `global`

```
>>> x =100
>>> def foo():
...     print(x)
...
>>> foo()
100
>>> def foo2():
...     x = 30
...     print(x)
...
>>> foo2()
30
>>> x
100
>>> def foo3():
...     global x
...     x =50
...     print(x)
...
>>> foo3()
50
>>> x
50
>>> |
```

/ First-Class Citizens

在 Python 中，函數是一個一級公民（first-class citizen）。這意味著，函數與任何其他對象（例如：整數、字符串、列表）一致，既可以動態地創建或銷毀，也可以傳遞給其他函數，或作為值進行返回。

```
>>> import math
>>> def foo(func):
...     return func(math.pi)
...
>>> foo(math.sin)
1.2246063538223773e-16
>>> foo(math.cos)
-1.0
>>>
```

```
>>> for func in [math.sin, math.cos, math.tan]:
...     func(math.pi/6)
...
0.49999999999999994
0.86602540378443871
0.57735026918962573
>>> |
```

/ 物件導向設計[WIKI]

物件導向程式設計（英語：Object-oriented programming，縮寫：OOP）是種具有物件概念的程式設計典範，同時也是一種程式開發的抽象方針。它可能包含資料、屬性、程式碼與方法。物件則指的是類別的實例。它將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充性，物件裡的程式可以存取及經常修改物件相關連的資料。在物件導向程式設計裡，電腦程式會被設計成彼此相關的物件。

支援物件導向程式語言通常利用繼承其他類達到代碼重用和可擴展性的特性。而類有兩個主要的概念：

- 類：定義了一件事物的抽象特點。類的定義包含了資料的形式以及對資料的操作。
- 物件：是類的實例。

Python資料結構都是物件

Python裡面所有的資料結構都是類別，當建立變數時即產生了物件。物件除了可以儲存資料，還可以透過逗號取用當中的值或對應的**“方法”**來對資料作處理，像是：

- `x=0.33; y=x.is_integer()`
- `x=3+4j; y=x.real`
- `x='abc'; y=x.upper()`
- `x=[1,2,3]; x.append(4)`
- `x={'mm':1e-3, 'um':1e-6, 'nm':1e-9}; y=x.keys()`

物件可以被建立(Create)，被修改(Update)，被讀取(Read)，被刪除>Delete)。這四個動作簡稱CURD。

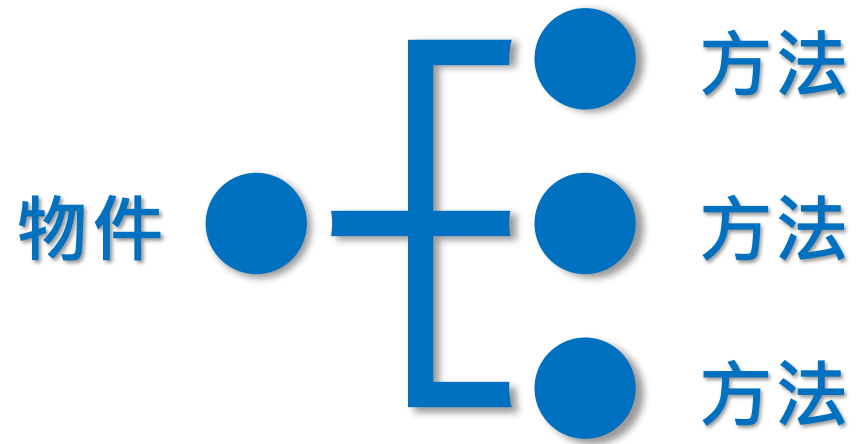
物件的方法(method)

方法與函數略有不同：

- 方法依附在物件之上
- 方法可以存取物件內的資料

特性：

- 物件可以支援多個方法
- 透過物件名之後加上逗號來使用
- 方法可以返回值或更新物件而不返回值
- 注意的是逗號引用並不表示就是方法，比方說`math.sin()`是函數，`math`是模組，不是物件。



/ 定義一個類別

```
7 import math
8 class vector():
9     def __init__(self, x, y, z):
10         self.x = x
11         self.y = y
12         self.z = z
13
14     def __add__(self, other):
15         x = self.x + other.x
16         y = self.y + other.y
17         z = self.z + other.z
18         return vector(x, y, z)
19
20     def mag(self):
21         return math.sqrt(pow(self.x,2)+pow(self.y, 2)+pow(self.z,2))
22
23     def __repr__(self):
24         return '({},{},{})'.format(self.x, self.y, self.z)
```

/ 用類別宣告物件並做運算

```
26 u = vector(1,2,3)
27 v = vector(3,2,1)
28 w = u + v
29 print(w)
30 z = w + u
31 print(z)
32 print(z.mag())
```

```
In [27]: runfile('C:/Users/mlin/AppData/Roaming/SPB_Data/
untitled7.py', wdir='C:/Users/mlin/AppData/Roaming/SPB_Data')
(4,4,4)
(5,6,7)
10.488088481701515
```

魔術方法(進階)

將兩個座標物件相加，我們可以有兩種操作方法：

- `Z = X.add(Y)`
- `Z = X + Y`

第二種方法顯然更直覺也更容易輸入。這時候我們可以在類別當中定義 `__add__(self, other)` 方法之後，就可以利用運算符號 `+` 來操作物件。除了 `+` 以外，還有許多的運算符號可以使用，這些統稱魔術方法。右邊顯示的是”部分”的魔術方法。

Magic Methods

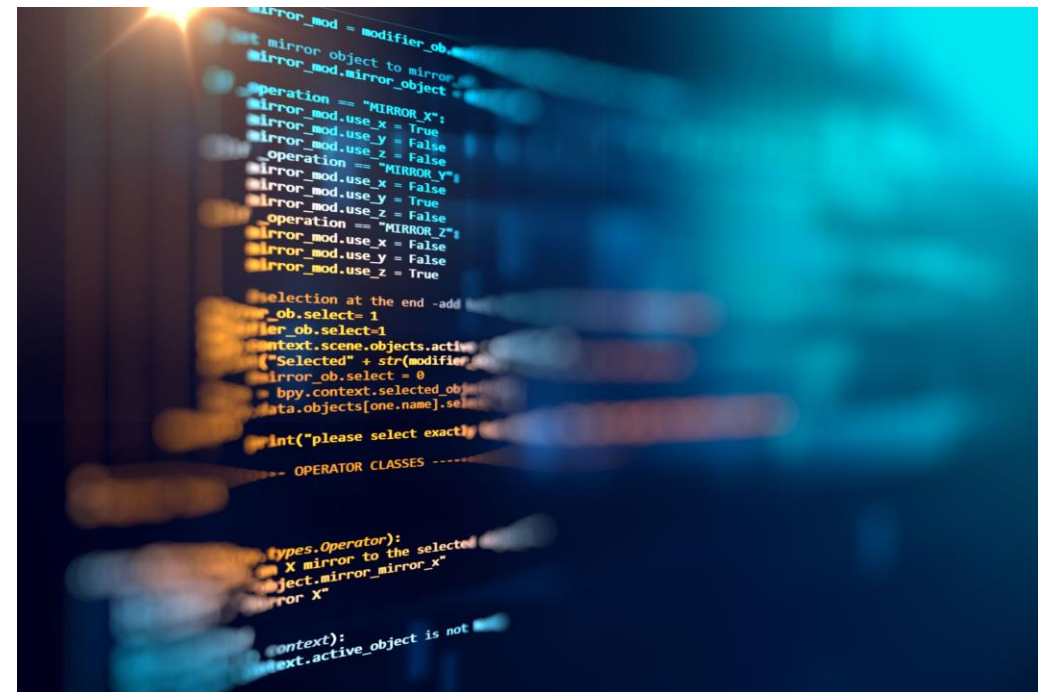
Python Syntax

Method Call

<code>a + b</code>	<code>a.__add__(b)</code>
<code>a - b</code>	<code>a.__sub__(b)</code>
<code>a * b</code>	<code>a.__mul__(b)</code>
<code>a / b</code>	<code>a.__truediv__(b)</code>
<code>a // b</code>	<code>a.__floordiv__(b)</code>
<code>a % b</code>	<code>a.__mod__(b)</code>
<code>a ** b</code>	<code>a.__pow__(b)</code>
<code>a == b</code>	<code>a.__eq__(b)</code>
<code>a != b</code>	<code>a.__ne__(b)</code>
<code>a < b</code>	<code>a.__lt__(b)</code>
<code>a > b</code>	<code>a.__gt__(b)</code>
<code>a <= b</code>	<code>a.__le__(b)</code>
<code>a >= b</code>	<code>a.__ge__(b)</code>

PEP8 PYTHON 編碼規範手冊

PEP8 是 Python 社群共通的風格指南，一開始是 Python 之父 Guido van Rossum 自己的撰碼風格，慢慢後來演變至今，目的在於幫助開發者寫出可讀性高且風格一致的程式。許多開源計畫，例如 Django、OpenStack 等都是以前 PEP8 為基礎再加上自己的風格建議。



專題討論

期末報告

- 是否製作專題由各組自行決定
- 要製作者請於下周三之前在臉書社團提出
- 其餘可選擇心得報告，像是：
 - 未來計畫如何使用所學在工作上
 - 學習到哪些有用的技巧
 - 課程改善建議

