

## **Week 2-自動化問題分析**

# Python基本語法簡介(2/4)

Python - Basic Operators

Python - Decision Making

Python - Loops

Python - Numbers

Python - Strings

# / 常用內建函式

- `abs()`
- `complex()`
- `dict()`
- `dir()`
- `enumerate()`
- `float()`
- `int()`
- `len()`
- `list()`
- `max()`
- `min()`
- `open()`
- `pow()`
- `print()`
- `range()`
- `round()`
- `str()`
- `sum()`
- `type()`
- `zip()`

Built-in Functions				
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

- Math Library
- Import Module
- Types of Operator
- Strings
  - Accessing Values in String
  - String Special Operators
  - Built-in String Methods
- Lists
- Tuples
- Dictionary
- Data Type Conversion
- Decision Making
- Loops

# / math module & cmath module

- 基本數學函式庫：<https://docs.python.org/3/library/math.html>
- 複數函式庫：<https://docs.python.org/3/library/cmath.html>

```
IronPython Command Window

=====
ElectronicsDesktop 2020.1.0
IronPython 2.7.0.40 on .NET 4.0.30319.42000
=====
- With Tab completion
- dir()      - lists all available methods and objects
- dir(obj)   - lists all available attributes/methods on obj
- help(obj)  - provides available help on a method or object
- tutorial() - provides more help on using the console
=====
try executing "dir(oDesktop)" or dir_sig(oDesktop,"ver")
=====
>>> import math
>>> math.sin(math.pi/4)
0.70710678118654746
>>> math.cos(math.pi/4)
0.70710678118654757
>>> dir(math)
['_doc_', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'loglp', 'modf', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> |
```

# / import指令

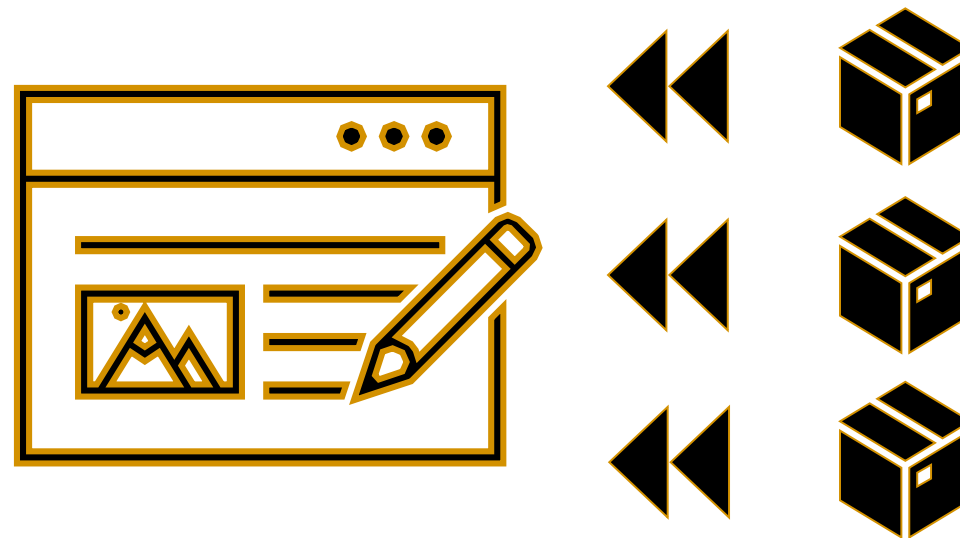
- import指令用來匯入模組
- 自動化常用模組
  - math, os, sys, re, tkinter, matplotlib
- 匯入指令

```
import math
```

```
from math import sin, cos
```

```
from math import sin as mysin
```

```
from math import *
```



# 字串的索引與切片

切片是相當方便的字串處理工具之一，利用中括號來取出部分字串或倒置

- `x[start:stop:step]`

```
In [8]: x='channel.s4p'
```

```
In [9]: x[0]
```

```
Out[9]: 'c'
```

```
In [10]: x[1:7]
```

```
Out[10]: 'hannel'
```

```
In [11]: x[-3:]
```

```
Out[11]: 's4p'
```

```
In [12]: x[:-4]
```

```
Out[12]: 'channel'
```

```
In [13]: x[::-1]
```

```
Out[13]: 'p4s.lennahc'
```

```
In [14]: x[::2]
```

```
Out[14]: 'canlsp'
```



# Escape Characters

```
>>> print('Hello World')
Hello World
>>> print('Hello\nWorld')
Hello
World
>>> print('Hello\\World')
Hello\World
>>> print('1\t2\t3')
1      2      3
>>> |
```

Code	Result	Try it
\'	Single Quote	<a href="#">Try it »</a>
\\	Backslash	<a href="#">Try it »</a>
\n	New Line	<a href="#">Try it »</a>
\r	Carriage Return	<a href="#">Try it »</a>
\t	Tab	<a href="#">Try it »</a>
\b	Backspace	<a href="#">Try it »</a>
\f	Form Feed	
\ooo	Octal value	<a href="#">Try it »</a>
\xhh	Hex value	<a href="#">Try it »</a>

## / 常用字串運算

- 除了切片，Python還提供了許多的字串處理函數。這裡展示其中幾個。

```
In [19]: x='Hello World'
In [20]: len(x)
Out[20]: 11
In [21]: 'or' in x
Out[21]: True
In [22]: y = x + "!!!"
In [23]: y
Out[23]: 'Hello World!!!'
In [24]: x.replace('World', 'Guys')
Out[24]: 'Hello Guys'
In [25]: x.split(' ')
Out[25]: ['Hello', 'World']
In [26]: x.upper()
Out[26]: 'HELLO WORLD'
In [27]: x.lower()
Out[27]: 'hello world'
```

## 邏輯判斷 if else elif

Python 提供了 if 、 else 、 elif 這三種語法來協助我們實現各種條件判斷和流程控制。Python 程式語言是一行一行執行的，所以當我們想要所寫的程式在某些條件下跳過某幾行敘述、不再照單全收的時候，就可以使用條件判斷。也就是說，如果要讓程式可以自動檢查所處理資料的內容，而且根據資料內容來決定是否執行某一個敘述或指令，那就需要用到條件判斷式來控制流程。

求解公倍數

```
for i in range(1000):  
    if i%21 == 0 and i%24 == 0:  
        print(i)
```

## 三元運算子(ternary operator)

- 三元運算子將變量指定及判斷整合在一行當中
- 好處是程式碼比較簡短。

- 一般

if  $x > y$ :

$z = x$

else:

$z = y$

- 三元運算子

$z = x$  if  $x > y$  else  $y$

## / 迴圈控制 for...in...:

- 單純的for迴圈會依序對列表當中每個元素做運算，直到所有元素都完成處理。加上break直接脫離迴圈，而continue敘述則是中斷這個元素後續工作進到下一個元素。這兩個敘述通常會搭配if-else使用。這裡用兩個例子來說明break以及continue的使用時機。

```
In [40]: for i in range(1, 1000):  
...:     if i%21==0 and i%24==0:  
...:         print(i)  
...:         break  
...:  
168
```

```
In [42]: for i in range(1, 1000):  
...:     if i%21==0 and i%24==0:  
...:         continue  
...:     else:  
...:         print(i)
```

# 函數range()與enumerate()

這兩個函數經常與for合併使用

- 函數range()產生等差的整數數列。
- enumerate多用於在for循環中得到計數，利用它可以同時獲得索引和值，即需要index和value值的時候可以使用enumerate。

```
In [44]: list(range(10))  
Out[44]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [45]: list(range(2, 10))  
Out[45]: [2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [46]: list(range(2, 10, 2))  
Out[46]: [2, 4, 6, 8]
```

```
In [50]: names = ['John', 'Mary', 'Anna', 'Ken']
```

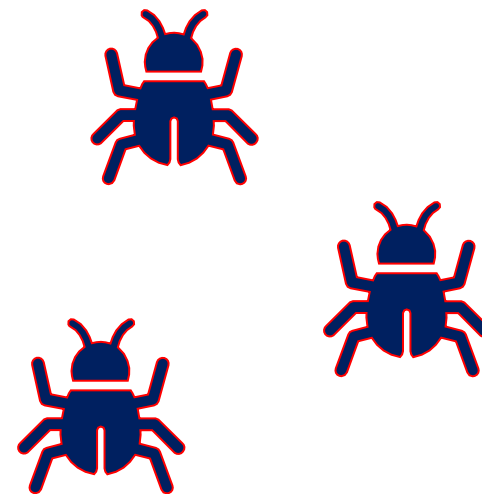
```
In [51]: for n, i in enumerate(names):  
...:     print(str(n) + ':' + i)  
...:
```

```
0:John  
1:Mary  
2:Anna  
3:Ken
```

# 除錯 (Debug)

寫程式是不斷的嘗試及除錯，程式錯誤基本上可分成兩種：

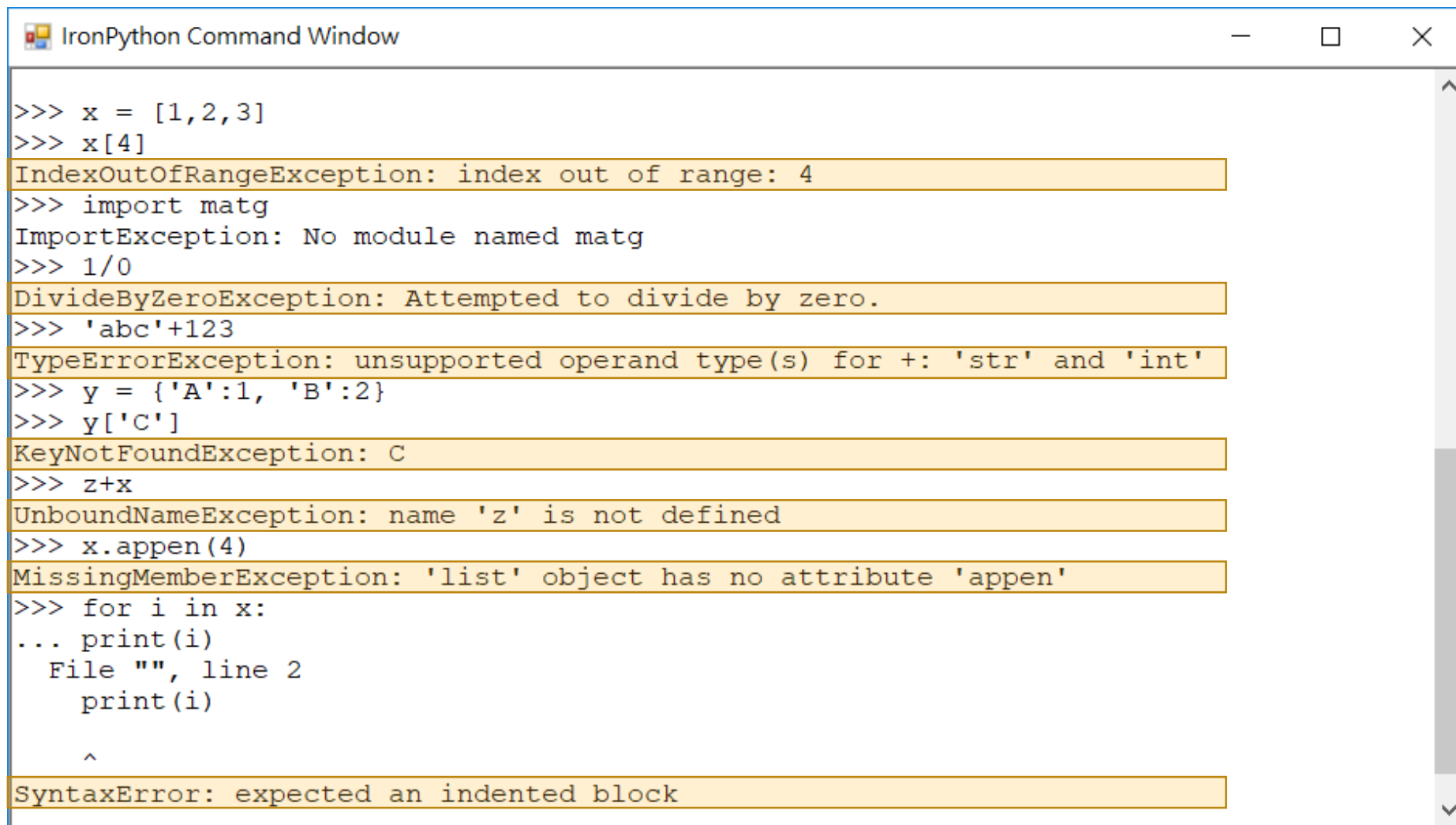
- Syntax Error(語法錯誤)
- Logic Error(邏輯錯誤)



Syntax error在執行過程便會被檢查出來，可以透過Exception推測發生的問題。修正之後既可繼續向下執行。

Logic Error則是輸出結果不如預期，這個錯誤要修正則是困難許多。一般需要設定斷點觀察變量的變動來追蹤可能的邏輯錯誤。在AEDT當中只能透過輸出變數值或log檔來協助除錯。

## 常見錯誤型態



```
IronPython Command Window

>>> x = [1,2,3]
>>> x[4]
IndexOutOfRangeException: index out of range: 4

>>> import matg
ImportException: No module named matg

>>> 1/0
DivideByZeroException: Attempted to divide by zero.

>>> 'abc'+123
TypeErrorException: unsupported operand type(s) for +: 'str' and 'int'

>>> y = {'A':1, 'B':2}
>>> y['C']
KeyNotFoundException: C

>>> z+x
UnboundNameException: name 'z' is not defined

>>> x.append(4)
MissingMemberException: 'list' object has no attribute 'appen'

>>> for i in x:
...     print(i)
File "", line 2
    print(i)
    ^
SyntaxError: expected an indented block
```



# 自動化程式開發流程

## / 找出模擬工作的瓶頸並定義題目

首先試著在問題中加入明確的輸入及輸出。舉例來說，“如何更快的畫出3D結構”就是屬於一個大問題，沒有明確的輸入及輸出。加入輸出入的問題會像是這樣子：

- “如何給定一條曲線（輸入），沿著線產生等間距的via（輸出）”，或是
- “如何給定座標點（輸入），將物件（輸入）複製到座標點上面（輸出）”或是，
- “選擇焊片（輸入），如何在上面自動產生錫球（輸出）”。

加入了輸入及輸出的描述讓整個問題變得具體也更容易理解。

## 分析題目並拆解為更小的問題

接下來要試著將上面的問題變成是（輸入—輸出）—（輸入—輸出）—（輸入—輸出）的格式，拆解問題的同時也是在回答問題，如同抽絲剝繭一般，這時候已經開始進入了程式開發的工作。舉例來說，“如何給定座標點”變成是“如何將儲存在csv檔案的座標點讀到陣列當中”以及“如何將選擇的物件輸出名稱”和“如何將輸出物件根據陣列的座標位置進行複製”。

要注意的是，除非你已經是有經驗的開發者，否則這些問題並不是你在程式開發初期就可以想的出來的，而是在程式碼開發過程當中，透過不斷嘗試去摸索出來的。過程當中答案也可能會一變再變，不斷嘗試不同的小程式碼片段去解決其中一個小問題，直到找到答案為止。

# 將所有的代碼拼湊出來

等到把所有的拼圖碎片找齊，最後便是拼圖的工作。這個階段不斷進行拼湊程式碼，測試，除錯的工作循環。就像拼圖一樣，先拼出邊邊角角比較容易的部分。將四個邊角完成之後在設法將這些大塊連結起來。

開發自動化程式很少是一氣呵成，一筆而就的。實際的過程更像是摸著石子過河，走到半途退回去另尋他路更是常有的事。就算是百轉千折，只要最終能到達對岸就是成功的達成任務，這也是學習程式開發的必經階段。至於對有開發經驗的設計師來說，可以快速的找出最短路徑，加上手上現有的工具，很快就能完成開發的工作。



# 範例解說：如何根據CSV 紀錄的座標複製物件

# 專題討論

## / 個人專題分享

- 分享想要透過自動化解決的題目及緣由，並設想輸入及輸出。
- 每位5分鐘時間並利用板書搭配口頭說明。
- 講師針對題目給出建議及方向，並評估難易度。
- 分享供所有學員參考，不一定為最後專題題目。
- 待第四周所有學員完成分享之後，再從當中選擇最後的專題題目。

題目:

緣由:

輸入:

輸出: