

Week 4-AEDT操作錄製與修改



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
- Try... Exception...

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。

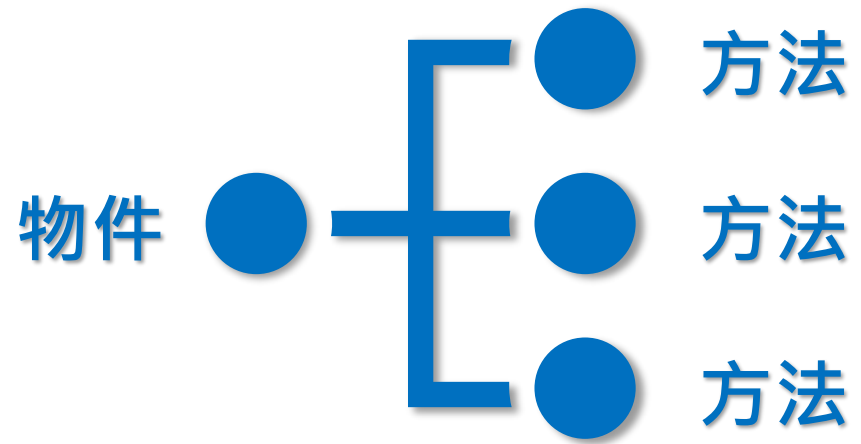
物件的方法(method)

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

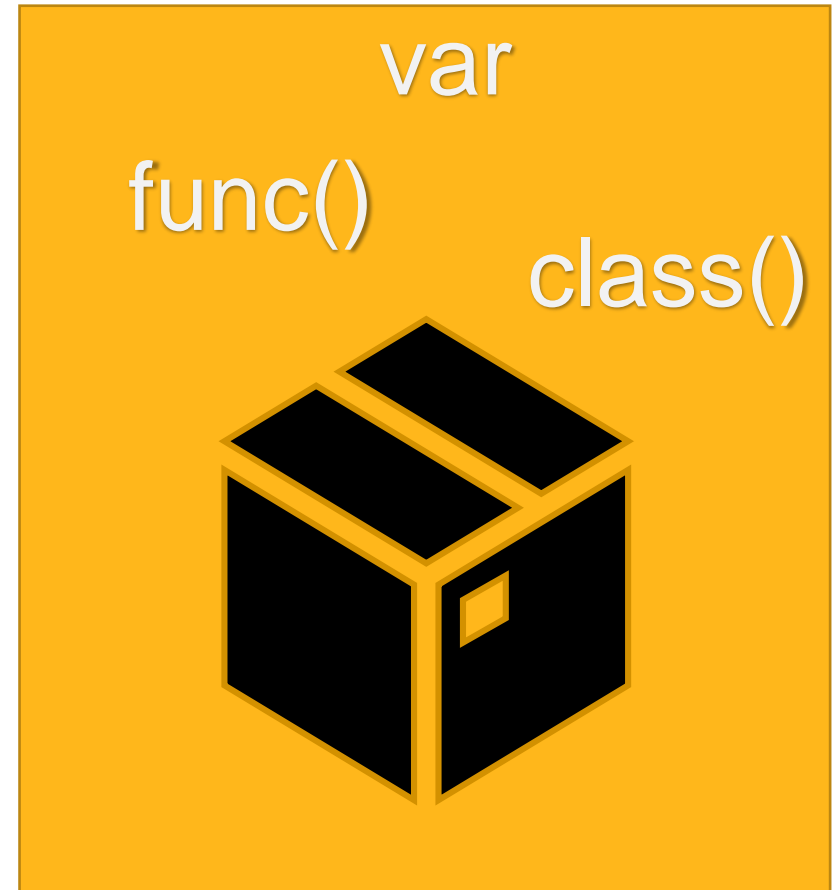
<code>a + b</code>	<code>a.__add__(b)</code>
<code>a - b</code>	<code>a.__sub__(b)</code>
<code>a * b</code>	<code>a.__mul__(b)</code>
<code>a / b</code>	<code>a.__truediv__(b)</code>
<code>a // b</code>	<code>a.__floordiv__(b)</code>
<code>a % b</code>	<code>a.__mod__(b)</code>
<code>a ** b</code>	<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>

/ 模組(Module)

只要你建立了一個原始碼檔案 `modu.py`，你就建立了一個模組 `modu`，原始碼主檔名就是模組名稱。

`import modu` 陳述句會在相同目錄下尋找 `modu.py`，如果沒找到，則會試著尋找在 `sys.path` 中遞迴地尋找 `modu.py`，如果還是沒有，則會引發 `ImportError` 例外。

模組提供了名稱空間。模組中的變數、函式與類別，基本上需透過模組的名稱空間來取得。在 Python 中，`import`、`import as` 與 `from import` 是陳述句，可以出現在程式中陳述句可出現的任何位置。



例外處理 v.s. 事先檢查

在程式開發階段，一般都是假設輸入參數會嚴格遵守規範，並基於這個假設開發演算法。如果格式出錯，就算這個錯誤無關緊要，程式也會終止運算並返回錯誤訊息，如果要對輸入的資料一一判斷是否合規，將導致程式複雜化並額外耗用運算資源。

一個解決思維是先做再說，發生了正常流程無法處理的狀況，也就是例外，再另行處理。這種設計思維就是所謂的例外處理。目前例外處理已經是主流程式語言所採取的設計方案。這種設計風格又稱之為EAFP(Easier to Ask for Forgiveness than Permission) ,有別於LBYL(Look Before You Leap)。

如何將下列不同csv檔讀入的字串處理成list of tuple ?

csv1=['1,5','2,6','3,7', '4,8']

csv2=['#comment', '1,5','2,6','3,7', '4,8']

csv3=['2020,06,05', '1,5','2,6','3,7', '4,8']

csv4=['ANSYS','HFSS','1,5','2,6','3,7', '4,8']

csv5=['time, volt','1,5','2,6','3,7', '4,8']

csv6=[' ', '1,5','2,6','3,7', '4,8']

csv7=['1,5','2,6','3,7', '4,8', '']

```
17 output = []
18
19 for i in csv1:
20     x, y = i.split(',')
21     output.append((float(x), float(y)))
22
23 print(output)
```

只能處理csv1, 處理其他
(csv2-csv7) 都會錯誤

```
In [6]: runfile('D:/Downloads/untitled5.py', wdir='D:/Downloads')
[(1.0, 5.0), (2.0, 6.0), (3.0, 7.0), (4.0, 8.0)]
```

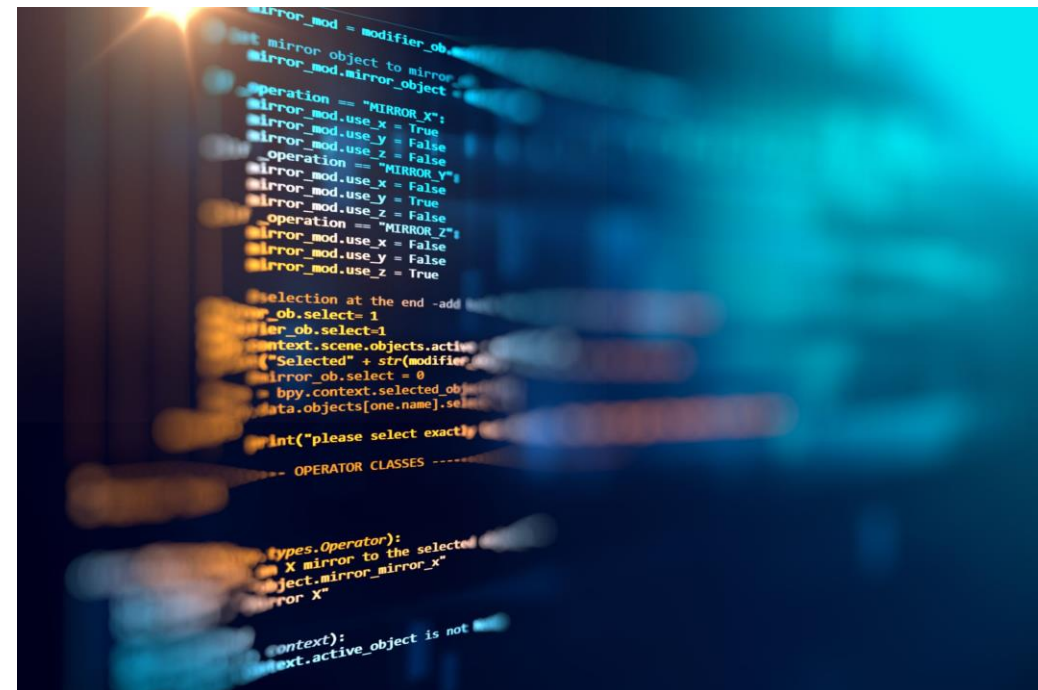
/ 解答:加上try...except...pass

```
16
17 output = []
18
19 for i in csv7:
20     try:
21         x, y = i.split(',')
22         output.append((float(x), float(y)))
23     except:
24         pass
25
26 print(output)
```

處理不來的就pass

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。



```
mirror_mod = modifier_ob
set mirror object to mirror
mirror_mod.mirror_object =
operation = "MIRROR_X";
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation = "MIRROR_Y";
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation = "MIRROR_Z";
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True

selection at the end -add
ob.select= 1
ler ob.select=1
context.scene.objects.active
("Selected" + str(modifier
mirror_ob.select = 0
= bpy.context.selected_ob
data.objects[one.name].sel

int("please select exactly

--- OPERATOR CLASSES ---

types.Operator):
X mirror to the selected
object.mirror_mirror_x"
error X"

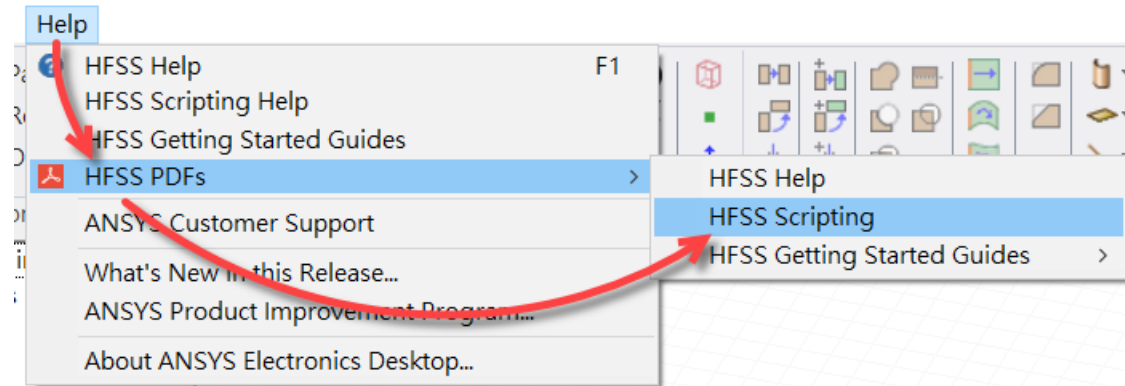
context):
context.active_object is not
```

AEDT函式庫架構

/ AEDT函式庫的完整文件

關於函式庫的Help檔可以在C:\Program Files\AnsysEM\AnsysEM20.1\Win64\Help底下找到ScriptingGuide.pdf當中詳細說明了函式的功能及使用方法。或是在Help底下也可以找到。

Modeler - [Project18 - HFSSDesign1 - Modeler]



Q3D Extractor S



ANSYS, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
<https://www.ansys.com>
(T) 724-746-3304
(F) 724-514-9494



HFSS Scripting Guide



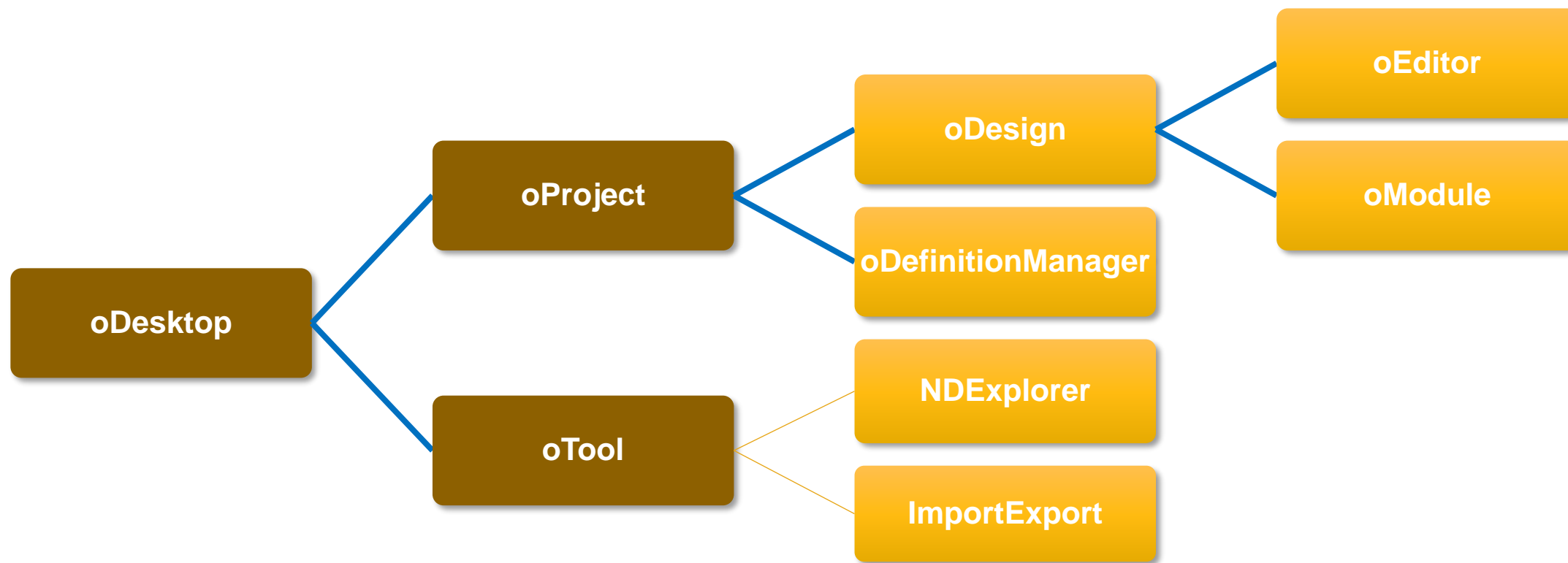
ANSYS, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
<https://www.ansys.com>
(T) 724-746-3304
(F) 724-514-9494

Release 2020 R1
January 2020

ANSYS, Inc. and
ANSYS Europe,
Ltd. are UL
registered ISO
9001:2015 com-
panies.

物件的階層

AEDT函式庫分成Desktop, Project, Design, Editor, Module等大類。當中Desktop, Project及Tool是共用的。之後的分類不同的產品會略有不同。



輸入dir()顯示最上層的函式與物件oDesktop

```
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
- help(obj)  - provides available help on a module
- tutorial() - provides more help on using the application
=====


try executing "dir(oDesktop)" or dir_sig(oDesktop)

=====
>>> dir()
['AddErrorMessage', 'AddFatalMessage', 'AddInfoMessage', 'AddWarningMessage',
'CreateObject', 'LogDebug', 'LogError', 'RunScriptCommand', 'RunScriptFile',
'SetScriptingLanguageToJavascript', 'SetScriptingLanguageToVBScript', '__builtins__',
'__doc__', '__name__', '__scopeID__', 'dir_sig', 'dir_sig_doc', 'oAnsoftApplication',
'oDesktop', 'onlinehelp', 'tutorial']
>>>
```

訊息顯示函式

oDesktop物件

輸入dir(oDesktop)顯示oDesktop的方法和屬性



```
IronPython Command Window
>>> dir(oDesktop)
['AddMessage', 'ClearMessages', 'CloseAllWindows', 'CloseProject',
'CloseProjectNoForce', 'DeleteProject', 'DoesRegistryValueExist',
'DownloadJobResults', 'EnableAutoSave', 'Equals', 'ExportOptionsFiles',
'GetActiveProject', 'GetAutoSaveEnabled', 'GetBuildDateTimeString',
'GetDefaultUnit', 'GetDistributedAnalysisMachines',
'GetDistributedAnalysisMachinesForDesignType', 'GetExeDir', 'GetGDIObjectCount',
'GetHashCode', 'GetLibraryDirectory', 'GetLocalizationHelper', 'GetMessages',
'GetPersonalLibDirectory', 'GetProcessID', 'GetProjectDirectory', 'GetProjectList',
'GetProjects', 'GetRegistryInt', 'GetRegistryString', 'GetScriptingToolsHelper',
'GetSysLibDirectory', 'GetTempDirectory', 'GetTool', 'GetType',
'GetUserLibDirectory', 'GetVersion', 'LaunchJobMonitor', 'MemberwiseClone',
'NewProject', 'OpenAndConvertProject', 'OpenMultipleProjects', 'OpenProject',
'OpenProjectWithConversion', 'PageSetup', 'PauseRecording', 'PauseScript', 'Print',
'QuitApplication', 'ReferenceEquals', 'RefreshJobMonitor', 'ResetLogging',
'RestoreProjectArchive', 'RestoreWindow', 'ResumeRecording', 'RunACTWizardScript',
'RunProgram', 'RunScript', 'RunScriptWithArguments', 'SelectScheduler',
'SetActiveProject', 'SetActiveProjectByPath', 'SetLibraryDirectory',
'SetProjectDirectory', 'SetRegistryFromFile', 'SetRegistryInt',
'SetRegistryString', 'SetTempDirectory', 'ShowDockingWindow', 'Sleep', 'SubmitJob',
'TileWindows', 'ToString', '__class__', '__delattr__', '__doc__', '__format__',
'__getattr__', '__hash__', '__init__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__']
```


/ oDesktop -> oProject -> oDesign

- AEDT的函式庫是以物件導向，階層式的方式建構而成。最上層為oDesktop。底下衍生出來的不同模塊有各自的函數可以操作。比方說oDesktop不存在函式讓我們可以加入新的design。而在oProject底下可以找到InsertDesign()的指令可以插入新的設計。所以正確的順序是先從oDesktop建立新的project並指定到變數oProject，接著再使用oProject.InsertDesign()加入設計並指定到oDesign變數當中。這個順序跟我們手動設定方法是一致的。

```
oProject = oDesktop.NewProject()  
oDesign = oProject.InsertDesign("HFSS", "HFSSDesign1", "DrivenTerminal", "")
```

/ oDesign -> oEditor

以此類推，假設我們要在設計當中加入圓柱，需要用到oEditor.CreateCylinder()指令，而oEditor需要從oDesign當中取得，這時候程式碼為：

```
oEditor = oDesign.SetActiveEditor("3D Modeler")
oEditor.CreateCylinder(
[
  "NAME:CylinderParameters",
  "XCenter:=" , "-0.4mm",
  "YCenter:=" , "-0.4mm",
  "ZCenter:=" , "0mm",
  "Radius:=" , "0.447213595499958mm",
  "Height:=" , "1.2mm",
  "WhichAxis:=" , "Z",
  "NumSides:=" , "0"
],
[
  "NAME:Attributes",
  "Name:=" , "Cylinder1",
  "Flags:=" , "",
  "Color:=" , "(143 175 143)",
  "Transparency:=" , 0,
  "PartCoordinateSystem:=" , "Global",
  "UDMId:=" , "",
  "MaterialValue:=" , "\"vacuum\"",
  "SurfaceMaterialValue:=" , "\"\"",
  "SolveInside:=" , True,
  "IsMaterialEditable:=" , True,
  "UseMaterialAppearance:=" , False,
  "IsLightweight:=" , False
])
```

/ 多個設計變量

以上的描述乍看有些複雜，實際使用過幾次之後，抓到訣竅之後其實並不難懂。這樣的架構好處是可以同時控制多個專案及設計，舉例來說，將某一個HFSS設計的局部結構複製到另一個Q3D的設計。我們可以用變量oHFSS及oQ3D分別代表HFSSDesign1及Q3DDesign1，並將HFSSDesign1當中的圓柱Cylinder1複製到Q3DDesign1設計當中。

```
oHFSS = oProject.SetActiveDesign("HFSSDesign1")  
oHFSSEditor = oHFSS.SetActiveEditor("3D Modeler")  
oHFSSEditor.Copy(  
[  
  "NAME:Selections",  
  "Selections:=" , "Cylinder1"  
])
```

```
oQ3D = oProject.SetActiveDesign("Q3DDesign1")  
oQ3DEditor = oQ3D.SetActiveEditor("3D Modeler")  
oQ3DEditor.Paste()
```

物件方法多以動詞+名詞來描述

- 建立(Create)
 - Add, New, Open,...
- 更新(Update)
 - Set, Show, Reset, Edit,...
- 讀取(Read)
 - Get, Is,...
- 刪除>Delete)
 - Close, Quit, Delete, Remove, ...

oDesktop

AddMessage
CloseProject
DoesRegistryValueExist
ExportOptionsFiles
GetBuildDateTimeString
GetDistributedAnalysisMachinesForDesignType
GetLibraryDirectory
GetPersonalLibDirectory
GetProjectList
GetRegistryString
GetTempDirectory
GetVersion
OpenAndConvertProject
OpenProjectWithConversion
PauseScript
RefreshJobMonitor
RestoreWindow
RunProgram
SelectScheduler
SetLibraryDirectory
SetRegistryInt
ShowDockingWindow
TileWindows

ClearMessages
CloseProjectNoForce
DownloadJobResults
GetActiveProject
GetDefaultUnit
GetExeDir
GetLocalizationHelper
GetProcessID
GetProjects
GetScriptingToolsHelper
GetTool
LaunchJobMonitor
OpenMultipleProjects
PageSetup
Print
ResetLogging
ResumeRecording
RunScript
SetActiveProject
SetProjectDirectory
SetRegistryString
Sleep

CloseAllWindows
DeleteProject
EnableAutoSave
GetAutoSaveEnabled
GetDistributedAnalysisMachines
GetGDIObjectCount
GetMessages
GetProjectDirectory
GetRegistryInt
GetSysLibDirectory
GetUserLibDirectory
NewProject
OpenProject
PauseRecording
QuitApplication
RestoreProjectArchive
RunACTWizardScript
RunScriptWithArguments
SetActiveProjectByPath
SetRegistryFromFile
SetTempDirectory
SubmitJob

讀懂函式說明

函式名稱

OpenProject

功能解說

Opens a specified project.

輸入參數

UI Access	Click File>Open		
Parameters	Name	Type	Description
	<FileName>	<string>	Full path of the project to open
Return Value	An object reference to the newly opened project.		

返回值

呼叫範例

Python Syntax	OpenProject(<Filename>)
Python Example	<code>oDesktop.OpenProject("C:/Projects/MyProject.aedt")</code>

如何理解AEDT的方法參數

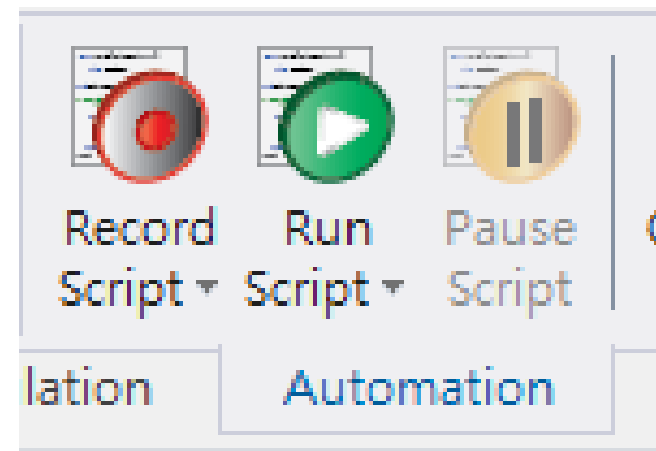
要理解函式，必須先從輸出入著手。一個函式可以有多個輸入參數，也可以沒有輸入參數。必須以正確的資料型態傳入參數，函式才能正常地完成運算或執行動作。假設參數x是以字串表示的數值，舉例來說“23”，那麼以整數型別傳入23是不被接受的。這是初學者經常犯的錯誤。同樣的，返回資料的型態可能是單純的整數或字串，也可能是物件。清楚返回值的資料格式才能正確的加以處理。在閱讀AEDT的函式說明時，先觀察輸入參數的型別，必要時可以編寫程式碼片段來測試。

如果要判斷返回資料的格式，最簡單的方式便是利用`type()`來得到資料型態，透過`dir()`來得到返回資料可以使用的方法。

錄製及修改

錄製腳本對於開發AEDT自動化程式有相對大的幫助。由於腳本的函式碼對應使用者的操作，減少了許多查詢及試錯的時間。要注意的是腳本的檔頭如果有中文字符，必須先移除。project與design的部分也要改成GetActiveProject()及GetActiveDesign()。使其可以使用在其他的專案。

如果要加上迴圈及判斷式，最好先將錄製的函式進行封裝簡化其輸出輸入。要留意保持正確的輸入參數型別。可以加上AddWarningMessage(str(X))適時的將執行過程中的變數值(X)輸出到訊息視窗，這有助於除錯。



範例解說：設定多組Eye Source及EyeProbe

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: