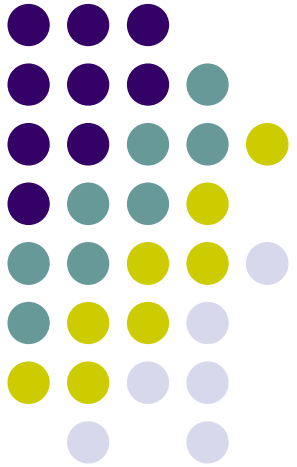
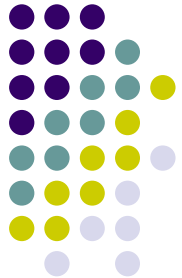
# Python程式設計入門
# 模組

葉難

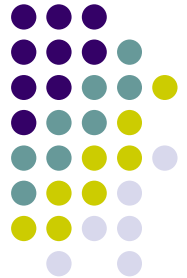


# 大綱



- 模組（module）
- 模組匯入述句：import、from
- 套件（package）：含子模組的模組
- 模組搜尋路徑
- 第三方模組
- 模組管理系統，虛擬環境

# 匯入模組，大致分爲兩部分

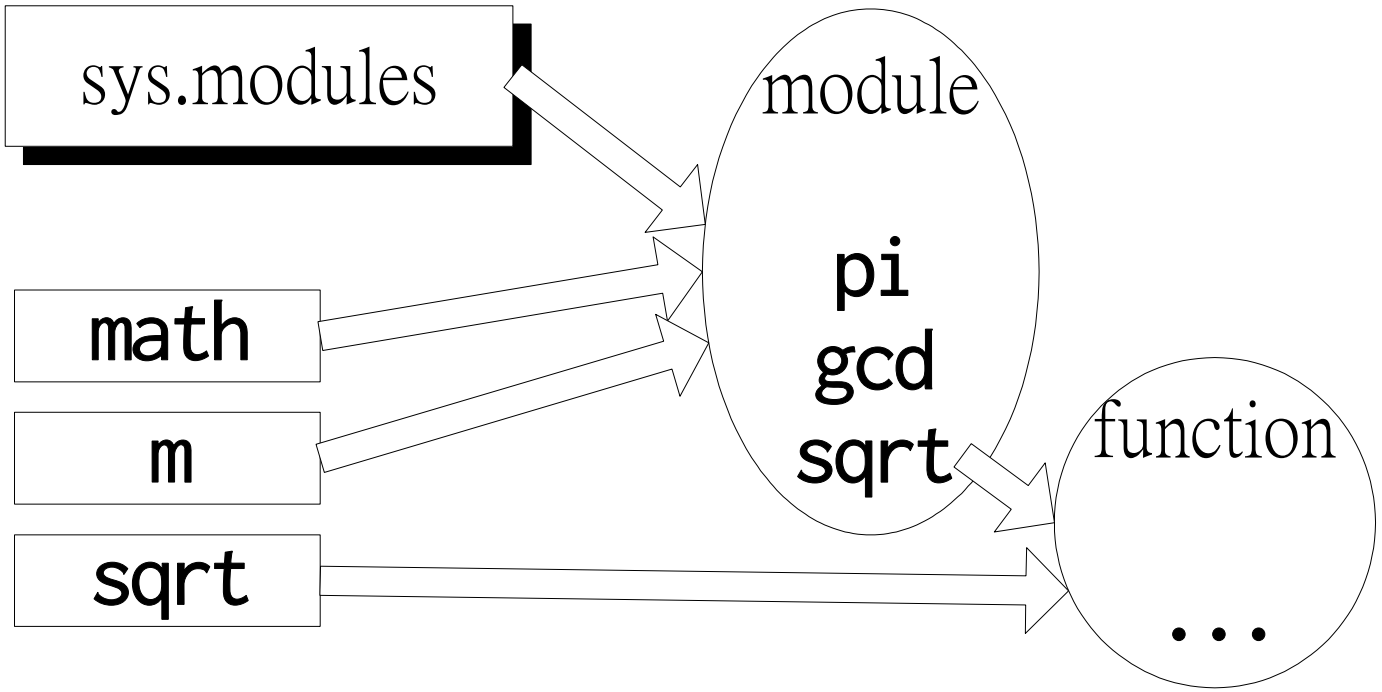


1) 尋找模組，載入（執行）
2) 名稱指派

- 模組搜尋路徑：sys.path
- 記錄已載入的模組：sys.modules

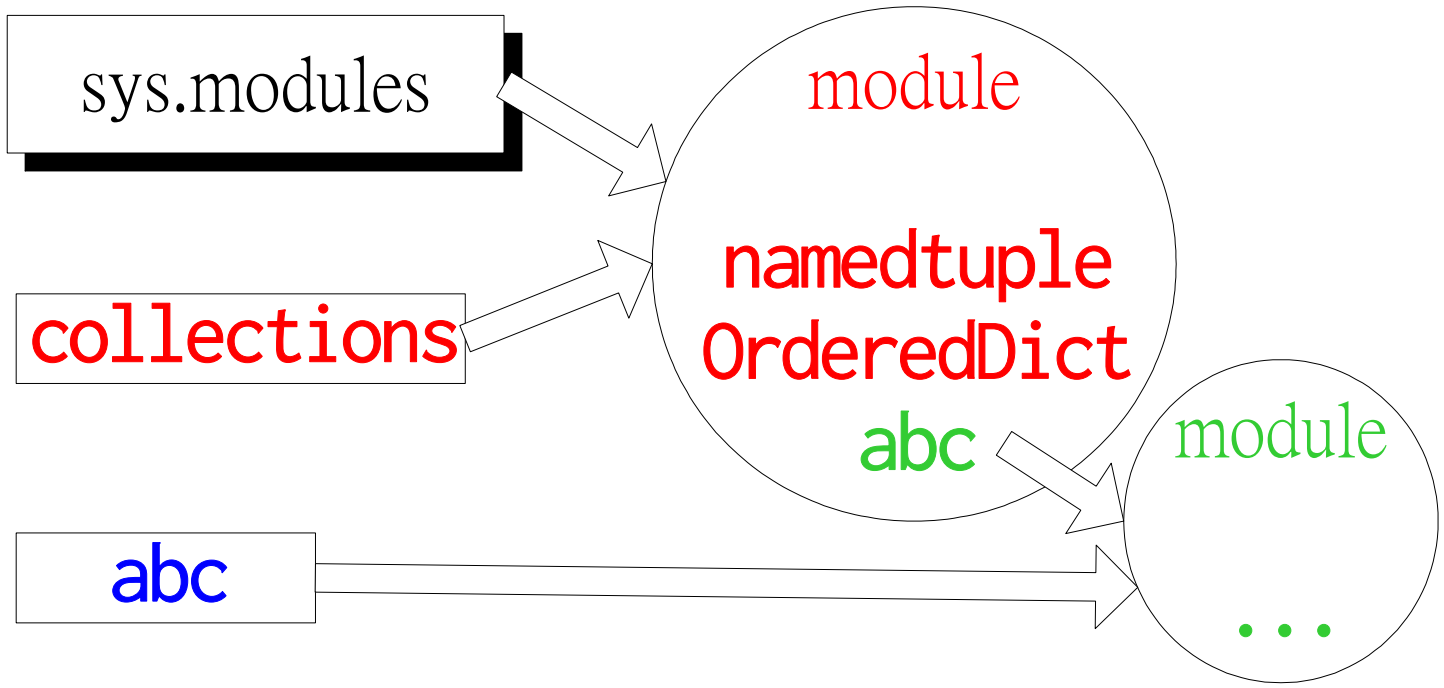
# 模組範例

```
import math
import math as m
from math import sqrt
```

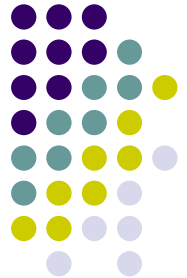


# 套件範例

```
import collections          # 注意，未匯入子模組abc
import collections.abc      # 注意，你尚未有名稱abc
from collections import abc   # 此時才有名稱abc
```



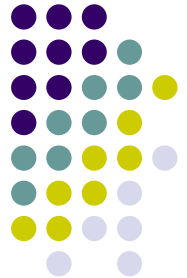
# 概念



- 模組就是管理程式碼的機制，管理名稱的機制
- 在Python裡，什麼都是物件

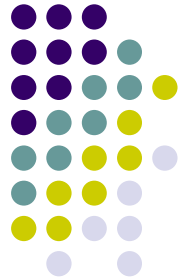
- 撞名、名稱衝突（name conflict）
- 不好的寫法「from module import *」，可能意外修改程式裡某個名稱的綁定關係
- 3.x版：禁止在函式內「import *」

# 模組



- 內建模組（標準程式庫）
- PyPI（Python Package Index）
  https://pypi.python.org/pypi
- 大型複雜模組：numpy、scipy、Django、Pygame、PyInstaller等

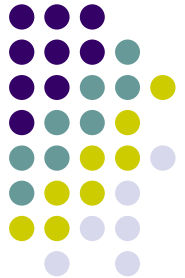
# 模組管理系統



- 手機App、Linux的deb與rpm、軟體商店
- 建置、散佈、下載、安裝、移除、版本更新

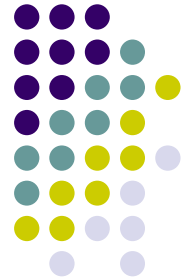
- Python自行制定一套標準，達到跨平台目標
- pip、setuptools
- 虛擬環境

# 簡史



- 模組distutils，「python setup.py install」
- PyPI：2003年創立
- 模組setuptools，格式Egg（.egg），指令easy_install，沒有解除安裝的能力
- 新格式Wheel（.whl）
- pip，以setuptools爲基礎
- site-packages，第三方模組存放處
- 虛擬環境virtualenv（或pyvenv）

# pip：安裝、更新



- 2.7.9、3.4版之後，已預設安裝
- 更新指令：

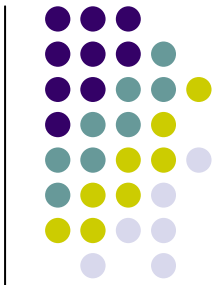
  Linux或Mac：pip install -U pip

  Windows：python -m pip install -U pip
- 自行安裝pip：

  下載get-pip.py（https://bootstrap.pypa.io/get-pip.py）

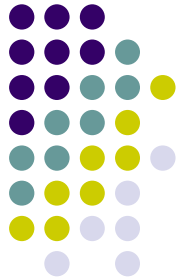
  然後執行「python get-pip.py」

# 路徑



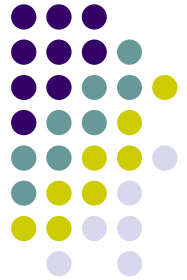
- 以Windows爲例
- Python直譯器：C:\Python34\
- pip與其他工具：C:\Python34\Scripts
- 第三方模組：C:\Python34\Lib\site-packages

# pip指令、參數



- pip install 模組名，安裝
- pip install 模組名==X.Y.Z，指定版本
- pip install 模組名>=X.Y.Z，指定最低版本
- pip install 模組檔名.whl，安裝已下載的模組
- pip install --upgrade 模組名，升級
- pip list，列出已安裝模組
- pip show 模組名，秀出詳細資訊

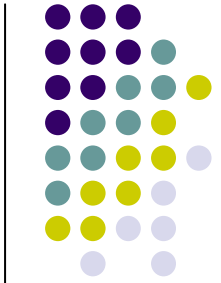
# 範例



- pip install psutil

```
>>> import psutil
>>> psutil.cpu_times()
scputimes(user=1966.15625, system=577.515625,
  idle=9777.109375)
>>> psutil.cpu_percent() # CPU運轉百分比
15.8
>>> psutil.cpu_count()   # 幾個邏輯CPU
1
```

# 範例



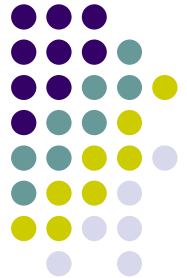
- pip install regex

```
>>> import regex
>>> p = regex.compile(r'Hello')
>>> p.match('Hello Python')
<regex.Match object; span=(0, 5),
  match='Hello'>
```

# 虛擬環境virtualenv

- pip install virtualenv，安裝
- 找個地方，virtualenv hellopy，建立
- cd hellopy，切換到目錄裡
- Linux
  啓動：source bin/activate
  終止：deactivate
- Windows：
  啓動： .\Scripts\activate.bat
  終止：deactivate.bat

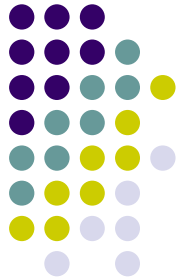
# 基本範例：主程式檔、模組檔



```
# myhello.py

import mymath as m

a, b = 26, 16

if __name__ == '__main__':
    print(__name__)
    print(m.__name__)
    print(m.pi)
    print(m.gcd(a, b))
```

```
# mymath.py

pi = 3.14

def gcd(a, b):
    ...

def factorial(n):
    ...
```

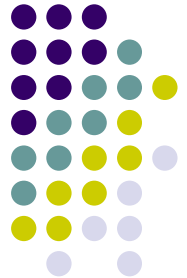
# 模組內私有名稱「_name」



```
# myhello2.py

from mymath2 import *
# print(_x)  # error

from mymath2 import _y
print(_y)    # ok
```

```
# mymath2.py

pi = 3.14
_x = 5
_y = 6
```

# 以「__all__」指定「import *」要產生的名稱



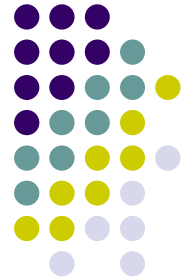
```
# myhello3.py

from mymath3 import *

# print(_x)  # error
print(pi)
# print(z)   # error
print(sq(9))
```

```
# mymath3.py

__all__ = ['pi', 'gcd',
   'sq']

pi = 3.14
_x = 5
_y = 6
z = 7
def gcd(a, b): pass
def sq(n): return n**2
```

# 再次載入模組

- 已載入的模組：sys.modules
- 主程式很大或不可關閉，正在開發某模組，想在修改後重新載入
- 使用模組importlib的reload，傳入已載入的模組物件
- 作用是「更新原來的模組物件」，以from/import匯入的名稱仍指向舊模組的各物件
- 許多C語言撰寫的模組，並未考量

# 模組搜尋路徑： **sys.path**



- 例：
- ['', 'C:\\WINDOWS\\system32\\python34.zip', 'C:\\Python34\\DLLs', 'C:\\Python34\\lib', 'C:\\Python34', 'C:\\Python34\\lib\\site-packages']
- 空字串代表當前目錄
- 模組可能是支.py檔、DLL檔、或放在壓縮檔裡

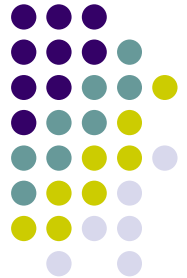
- 程式可在執行時動態修改sys.path

# sys.path從何而來

- Python實作的安裝設定
- 環境變數「PYTHONPATH」
- .pth檔

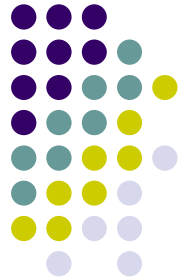
- Python直譯器初始化時，會自動載入模組site；能以「-S」關閉此行爲
- 會到某些目錄尋找.pth檔，增加模組搜尋路徑到sys.path

# 模組命名



- 不僅會是檔案名或目錄名，也會是Python程式裡的名稱
- 應遵守Python語言的名稱命名規則
- 建議全部採小寫字母
- 不要使用空格、中文、奇怪的符號
- 兩個單字的話，直接組合在一起，如xmlrpc
- 以C/C++開發的模組，慣例以「_」開頭

# 套件（**package**）

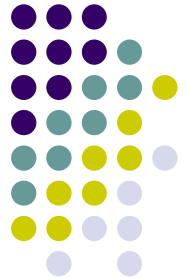


- 含子模組的模組
- 含`__init__.py`檔的目錄：成爲Python套件

- 套件目錄裡可含其他模組檔、套件目錄

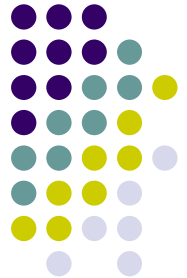
- 範例：package_example

# import：
# 匯入模組（套件）



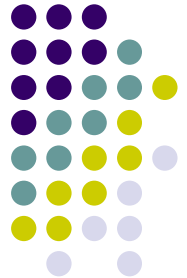
```
# 套件.模組      套件也會被載入
import package_exmple.info

# 套件.套件
import package_example.gui

# 套件.套件.模組
import package_example.gui.menu
import package_example.gui.menu as menu
```

# from/import：
# 模組（套件）或模組裡的名稱



```
# 套件的子模組或子套件
from package_example import info # 子模組
from package_example import gui  # 子套件

# 套件的子套件 的子模組
from package_example.tools import rotate

# 套件的子套件的子模組 的函式
from pacakge_example.tools.foo import foo
```

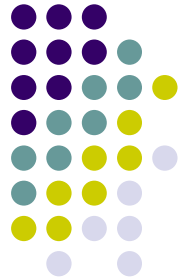
# from xxx import *



- 從一般模組：匯入該模組裡的名稱
- 從套件：匯入__init__.py裡的名稱，不含底下

```
# 子模組gui的全部名稱
from package_example.gui import *

# 先前已載入，之後也會產生名稱
import package_example.gui.menu
from package_example.gui import *
```

# from xxx import *：
# 自行列出想要匯入的名稱



```
# gui/__init__.py

# menu、canvas是子模組，
__all__ = ['menu', 'canvas', 'display',
  'draw']

def display(): pass
def draw(): pass
```

# 模組匯入其他模組

- 絕對式匯入

```
from package_example.tools import foo
```

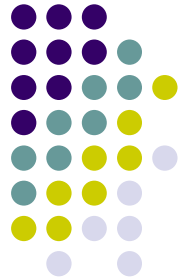
- 隱式相對式匯入（implicit relative import），僅適用2.x版：先到套件內搜尋，然後到sys.path搜尋

```
from foo import *
```

- 顯式相對式匯入（explicit relative import），

```
from ..gui import display
```

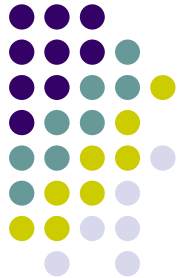
# 顯式相對式匯入



- 一個句點「.」代表一層

```
# 匯入同一層的模組bar
from . import bar
# 匯入同一層的模組（或套件）foo裡的所有名稱
from .foo import *

# 匯入上一層的子模組info
from .. import info
# 匯入上一層的子模組gui的函式display
from ..gui import display
```

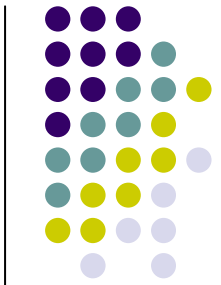
# 問題



- 一個句點「.」代表一層

```
from ..gui import menu

from ..gui.canvas import *

from ... import info
```

# 分拆模組檔



- 想讓一個模組分散到許多支檔案，但對外界而言猶如一個模組
- 把mymodule.py檔案改成目錄mymodule，模組檔案有sub1.py、sub2.py，__init__.py內容如下：

```
from .sub1 import *
from .sub2 import *
```

# 命名空間套件（**namespace package**）

- 3.3版
- 讓各支模組檔分散到不同目錄，但對外界而言猶如一個套件
- 沒有__init__.py的目錄，被當做「命名空間套件」，Python直譯器會把同名目錄組合成一個

```
path_1/
    mymodule/
        sub1.py
path_2/
    mymodule/
        sub2.py
```

```
# 使用時
import mymodule.sub1
import mymodule.sub2
```

# Q&A