

國立臺北科技大學自動化所 嵌入式工業機器視覺

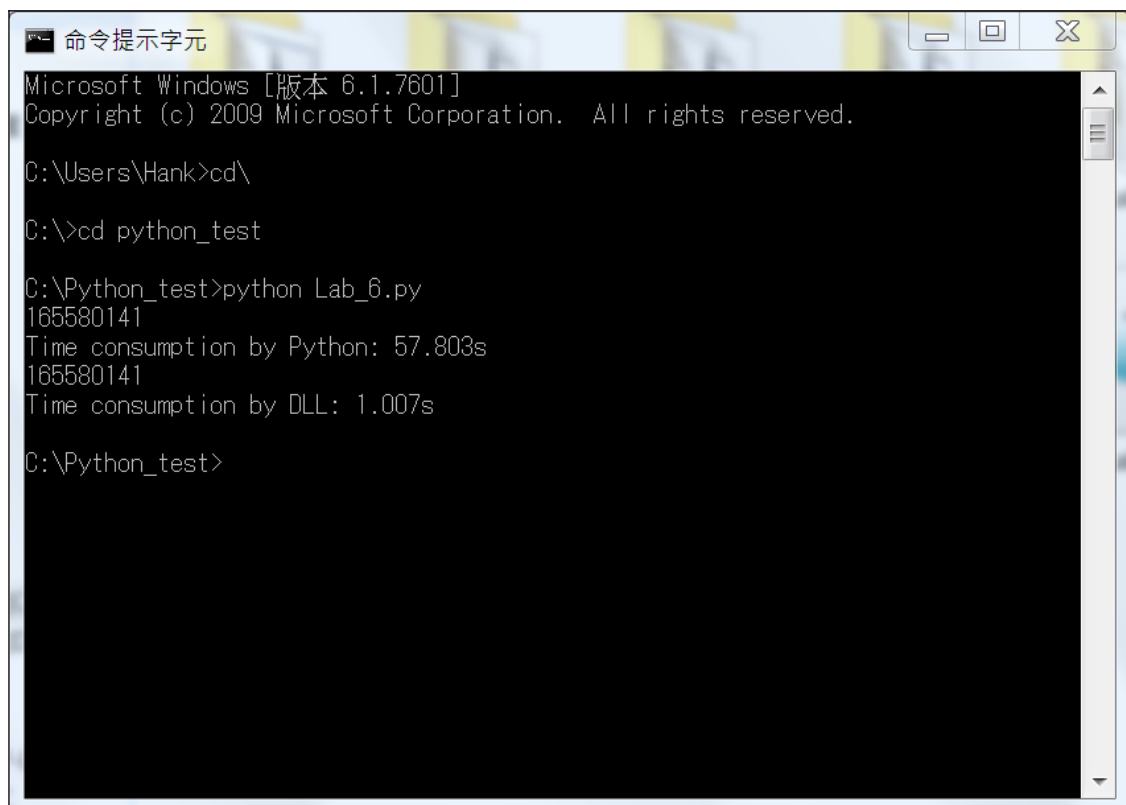
Lab 6 Using C code In Python

[Performance comparison between C and Python]

內容目錄

1. 這個練習會得到的成果圖
 2. 本練習的範例程式碼
 3. 安裝
 - 選擇官網安裝 Python 3.8 版以上
 - 選擇 Anaconda + vscode
 4. 範例程式碼
 - 複製或直接用範例程式碼，直接跑
 - 執行
 - 結果
 - 說明
 - ✓ 修改 Lab_1 的 C-type function DLL 專案，新增斐波那契（Fibonacci）數列計算函式
 - ✓ 新建 Lab_1.py 檔案 (NotePad++ 或 vscode 都可以)
 - ✓ 加入 python 版本的斐波那契（Fibonacci）數列計算函式
 - ✓ 利用 ctypes library 呼叫 DLL 檔
 - ✓ 比較 C 及 Python 執行斐波那契（Fibonacci）數列的計算效率
 5. 小結
- Reference

1. 這個練習會得到的成果圖



```
命令提示字元
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Hank>cd\
C:\>cd python_test
C:\Python_test>python Lab_6.py
165580141
Time consumption by Python: 57.803s
165580141
Time consumption by DLL: 1.007s
C:\Python_test>
```

2. 本練習的範例程式碼

MyDLL.h、MyDLL.cpp：

包裝成 DLL 檔的斐波那契（Fibonacci）數列 C 語言版本

Lab_6.py:

透過 ctypes 呼叫 DLL，並與 python 版本比較計算速度

3. 安裝

選擇官網安裝 Python 3.8 版以上 <https://www.python.org/downloads/>

安裝時要記得將 Python 路徑加入環境變數。

選擇 Anaconda + vscode <https://www.woodowlab.com/python-tutorial-0-anaconda/>

(optional，看想不想要管理 python 的 IDE 開發環境)

4. 範例程式碼

複製以下程式碼，直接跑：

應該能正常執行，不然就是沒成功安裝 Python 或是 DLL 有問題，記住 DLL 的檔案路徑要給正確。

```
1  import time
2  #from ctypes import cdll
3  from ctypes import *
4
5  def Fibo_Python(n):
6      if n<2:
7          return 1
8      else:
9          return Fibo_Python(n-1) + Fibo_Python(n-2)
10
11  start_time = time.time()
12  a = Fibo_Python(40)
13  end_time = time.time()
14
15  duration = end_time-start_time
16  print(a)
17  print(f'Time consumption by Python: {round(duration,3)}s')
18
19
20  #MyDLL = cdll.LoadLibrary('C:\Python_test\MyDLL.dll')
21  MyDLL = CDLL('C:\Python_test\MyDLL.dll')
22
23  start_time = time.time()
24  b = MyDLL.Fibo_C(40)
25  end_time = time.time()
26
27  duration = end_time-start_time
28  print(b)
29  print(f'Time consumption by DLL: {round(duration,3)}s')
```

執行：

python Lab_6.py

結果：

```
165580141
Time consumption by Python: 58.836s
165580141
Time consumption by DLL: 1.113s
```

說明：

(1) 修改 Lab_1 的 C-type function DLL 專案，新增斐波那契（Fibonacci）數列計算函式

```
#include <stdio.h>
#include <windows.h>

#ifdef MYDLL_EXPORTS
#define MYDLL_API __declspec(dllexport)
#else
#define MYDLL_API __declspec(dllimport)
#endif

#ifdef __cplusplus
extern "C" {
#endif

MYDLL_API int __cdecl Fibo_C(int n);

#ifdef __cplusplus
}
#endif
```

```
#include "MyDLL.h"

// 這是匯出函式的範例。
MYDLL_API int __cdecl Fibo_C(int n)
{
    if (n < 2)
        return 1;
    else
        return Fibo_C(n - 1) + Fibo_C(n - 2);
}
```

(2) 新建 Lab_6.py 檔案 (NotePad++ 或 vscode 都可以)

(3) 加入 python 版本的斐波那契（Fibonacci）數列計算函式

```
def Fibo_Python(n):
    if n<2:
        return 1
    else:
        return Fibo_Python(n-1) + Fibo_Python(n-2)

start_time = time.time()
a = Fibo_Python(40)
end_time = time.time()

duration = end_time-start_time
print(a)
print(f'Time consumption by Python: {round(duration,3)}s')
```

(4) 利用 ctypes library 呼叫 DLL 檔

```
from ctypes import *
MyDLL = CDLL('C:\Python_test\MyDLL.dll')
```

'C:\Python_test\MyDLL.dll' 是 DLL 的檔名及所在路徑，如果編譯 DLL 時用的是 64 位元，在執行 Python 時就必須採用 64 位元的版本。

ctypes 只能夠支援 C 的 interface，想要在 python 呼叫函式庫還有以下方式及其優缺點：

	優點	缺點
Python/C API	最原始，最大的控制權。	Reference count 很煩，比較需要 C 的基礎，只針對 CPython。
ctypes	使用簡單，不需編譯，可直接使用現成 library，基本使用上不需會 C。	Type 轉換比較麻煩，尤其是 struct、union、array 這種。
SWIG	支援多種語言。	要寫一份煩人的 interface file，Overhead 高。
Cython	兼顧開發與執行效能。	跟 Python 還是不太一樣，需要學新東西。

(5) 比較 C 及 Python 執行斐波那契（Fibonacci）數列的計算效率

```
165580141
Time consumption by Python: 58.836s
165580141
Time consumption by DLL: 1.113s
```

5. 小結

由於我們先前的練習都是用 C++/C# 來實作，有時用 python 來做 API 的整合測試相對方便；考慮到後續要移植到嵌入式系統(樹莓派 3)，因此後續會繼續實作 python / Qt 來對應 C++ / C# 的控制邏輯與介面。

Reference

- [1] <https://pyliaorachel.github.io/blog/tech/python/2018/01/26/python-c-mixing.html>
- [2] <https://blog.ez2learn.com/2009/03/21/python-evolution-ctypes/>
- [3] <https://lkm543.medium.com/dll%E6%AA%94%E6%98%AF%E7%94%9A%E9%BA%BC-%E8%AE%93dll%E6%8A%8Apython%E5%8A%A0%E9%80%9F75%E5%80%8D-994e35efa38d>
- [4] https://yodalee.me/2017/03/2017_python_ctype/