

國立虎尾科技大學
計算機程式 ag3 期末報告

PyQt5 事件導向計算器
PyQt5Event-DrivenCalculatorProject

學生：

設計一甲 40623107 簡稜雅
設計一甲 40623108 林郁涵
設計一甲 40623109 李如芳
設計一甲 40623116 楊子毅
設計一甲 40623117 楊智傑
設計一甲 40623118 楊秉澤

指導教授：嚴家銘

2018.01.10

摘要

機械設計一甲-計算機程式第三組的期末簡報，透過 **Github** 網站協同，分工合作完成計算機介面與程式。

- 各學員根據抽籤結果決定處理之按鈕
- 編譯各種按鍵之處理方式
- 期末計算機程式之心得感想

本簡報重點在於練習計算機的邏輯與 **PyQt5** 程式的運用以及分工合作之重要性。

Ag3 計算機程式分配：

40623107 簡稜雅: 前言、可攜系統的開始

40623108 林郁涵: 可攜系統的開始、關閉、python

40623109 李如芳: python

40623116 楊子毅: Calculator 程式

40623117 楊智傑: 結論與建議

40623118 楊秉澤: 倉儲系統

目錄

第一章 前言	3
第二章 可攜程式系統介紹	4
第三章 <i>Python</i> 程式語法	8
第四章 倉儲系統	14
第五章 Calculator 程式	20
第六章 心得	34
第七章 結論與建議	39

前言

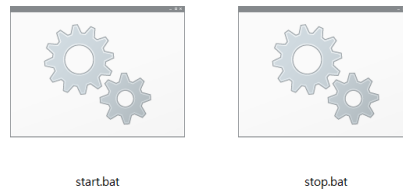
本次報告目的，在於統整這學期的課程內容，並加以檢視及複習，參照上課日期依序介紹教學內容：可攜程式系統介紹、**Python** 程式語法、倉儲系統、**Calculator** 程式。為多加練習多人協同合作，本報告由各組學員分段同時編寫，以利形式之效率。根據結果，多人協同有助於減少個人作業的壓力，並能與協同者有較多交流、討論及修改錯誤，使時間能在有效的運作下，縮短整體時間。最後歸納報告結論，並整合各學員討論及建議，作為後續相關課程的參考資料。

第二章-可攜程式系統介紹

2-1 啟動

1. 啟動

點選兩下在 `python2017fall_36` 裡的 `start.bat`，就可開啟

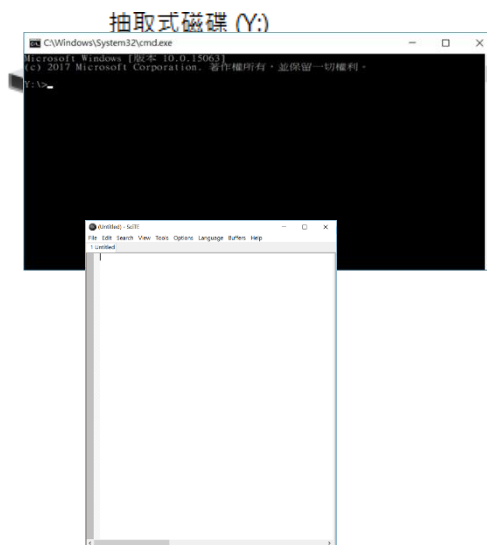


命令提示列及 **SciTE**，就可以處理後續作業。

(圖表 1.)可攜系統的開啟與關閉。

(圖表 2.)開啟後所產生的 Y 槽。

(圖表 3.)命令提示列。



(圖表 4)SciTE。

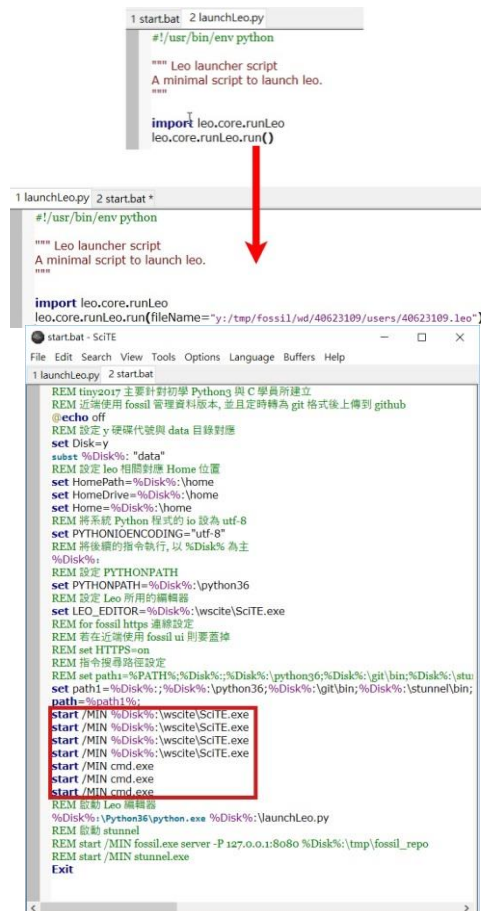
2-1-1 可攜式系統客製化解說

1. 為因應現在克制化趨勢，我們將啟動時間縮短，並達到快速的使用。以下是 `launchLeo.py` 中的縮減步驟。

```
REM 啟動 Leo 編輯器
REM %Disk%\Python36\python.exe %Disk%\launchLeo.py
...
REM 啟動 Leo 編輯器
%Disk%\Python36\python.exe %Disk%\launchLeo.py
```



2. 為了讓程式知道我要開啟的檔案，所以必須要在 `launchLeo.py` 中打下所開的檔案位置。
3. 開啟數量就由行數決定。



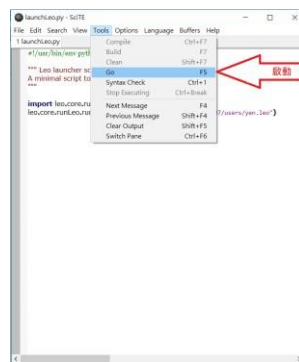
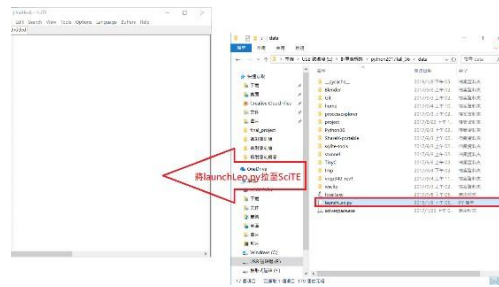
```
1 start.bat 2 launchLeo.py
#!/usr/bin/env python
""" Leo launcher script
A minimal script to launch leo.
"""
import leo.core.runLeo
leo.core.runLeo.run()

1 launchLeo.py 2 start.bat *
#!/usr/bin/env python
""" Leo launcher script
A minimal script to launch leo.
"""
import leo.core.runLeo
leo.core.runLeo.run(fileName="y:/tmp/fossil1109/users/40623109.Leo")

start.bat - SciTE
File Edit Search View Tools Options Language Buffers Help
1 launchLeo.py 2 start.bat
REM 2017 主要針對初學 Python3 與 C 學員所建立
REM 近端使用 fossil 管理資料版本，並且定時轉為 git 格式後上傳到 github
@echo off
REM 設定 y 硬碟代號與 data 目錄對應
set Disk=y
subst %Disk%: "data"
REM 設定 leo 相關對應 Home 位置
set HomePath=%Disk%\home
set HomeDrive=%Disk%\home
set Home=%Disk%\home
REM 將系統 Python 模式的 so 改為 utf-8
set PYTHONIOENCODING="utf-8"
REM 將後續的指令執行，以 %Disk% 為主
%Disk%:
REM 設定 PYTHONPATH
set PYTHONPATH=%Disk%\python36
REM 設定 Leo 所用的編輯器
set LEO_EDITOR=%Disk%\wscite\SciTE.exe
REM for fossil https 連線設定
REM 若在近端使用 fossil ui 則要蓋掉
REM set HTTPS=on
REM 指令搜尋路徑設定
REM set path1=%PATH%;%Disk%;%Disk%\python36;%Disk%\git\bin;%Disk%\stunnel\bin;
set path1=%path1%;
start /MIN %Disk%\wscite\SciTE.exe
start /MIN %Disk%\wscite\SciTE.exe
start /MIN %Disk%\wscite\SciTE.exe
start /MIN cmd.exe
start /MIN cmd.exe
start /MIN cmd.exe
REM 啟動 Leo 編輯器
%Disk%\Python36\python.exe %Disk%\launchLeo.py
REM 啟動 stunnel
REM start /MIN fossil.exe server -P 127.0.0.1:8080 %Disk%\tmp\fossil_repo
REM start /MIN stunnel.exe
Exit
```

2-1-2 Leo 開啟的步驟

步驟 1

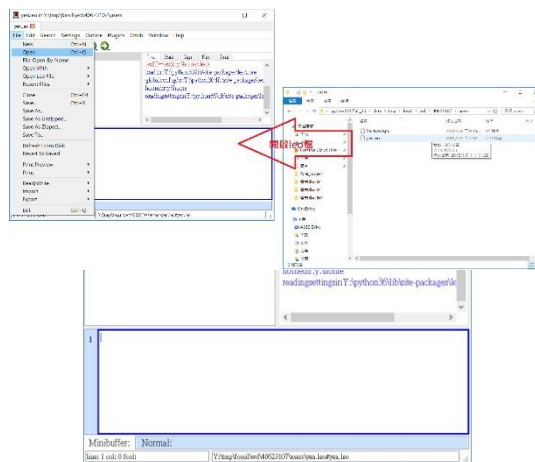


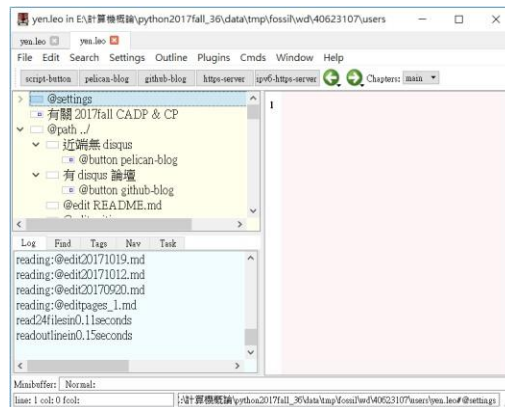
步驟 2

步驟 3

(原始 Leo)

步驟 4



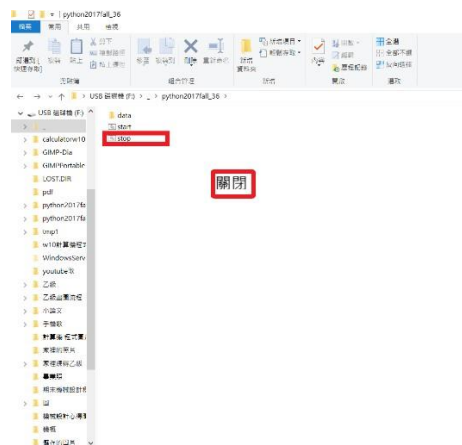


步驟 5.

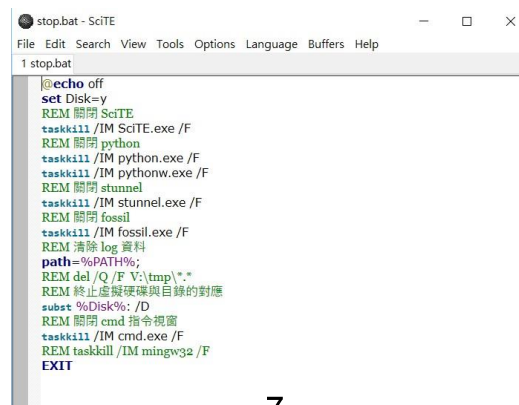
(開啟檔案後的 Leo 介面)

2-2 關閉

1. 關閉。



2. 關閉的程式碼。



第三章- Python 程式語法

3-1 命名規則

變數必須以英文字母大寫或小血或底線開頭。

變數其餘字元可以是英文大小寫字母，數字或底線。

變數區分英文大小寫。

變數不限字元長度。

不可使用關鍵字當作變數名稱。

Python3 的程式關鍵字, 使用者命名變數時, 必須避開下列保留字.

Python keywords: ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

使用有意義且適當長度的變數名稱， 例如: 使用 `length`

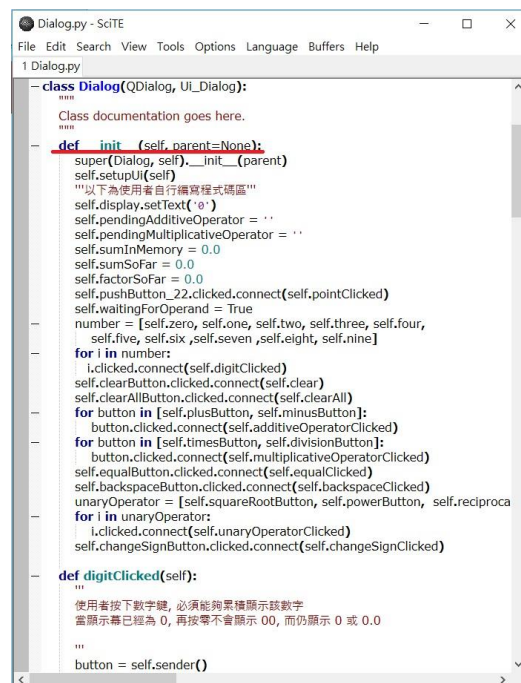
代表長度, 不要單獨使用 `l` 或 `L`, 也不要使用

`this_is_the_length` 程式前後變數命名方式盡量一致, 例如:

使用 `rect_length` 或 `RectLength` 用底線開頭的變數通常具有特殊意義

3-2 print() 函式用法

`print()` 為 Python 程式語言中用來列印數值或字串的函式，其中有 `sep` 變數定義分隔符號，`sep` 內定為 `" "`，`end` 變數則用來定義列印結尾的符號，`end` 內定為跳行符號。函式的定義：在 Python 中定義函式，是用 `def` 來定義，`def` 是個陳述句，Python 執行到 `def` 時，會產生一個函式物件，為 `function` 實例，即然函式是個物件，它可以指定給其他的變數。



```
Dialog.py - QtConsole
File Edit Search View Tools Options Language Buffers Help
1 Dialog.py

class Dialog(QDialog, Ui_Dialog):
    """
    Class documentation goes here.
    """
    def __init__(self, parent=None):
        super(Dialog, self).__init__(parent)
        self.setupUi(self)
        """以下為使用者自行撰寫程式碼區"""
        self.display.setText('0')
        self.pendingAdditiveOperator = ''
        self.pendingMultiplicativeOperator = ''
        self.sumInMemory = 0.0
        self.sumSoFar = 0.0
        self.factorSoFar = 0.0
        self.pushButton_22.clicked.connect(self.pointClicked)
        self.waitForOperand = True
        number = [self.zero, self.one, self.two, self.three, self.four,
                  self.five, self.six, self.seven, self.eight, self.nine]
        for i in number:
            i.clicked.connect(self.digitClicked)
        self.clearButton.clicked.connect(self.clear)
        self.clearAllButton.clicked.connect(self.clearAll)
        for button in [self.plusButton, self.minusButton]:
            button.clicked.connect(self.additiveOperatorClicked)
        for button in [self.timesButton, self.divisionButton]:
            button.clicked.connect(self.multiplicativeOperatorClicked)
        self.equalButton.clicked.connect(self.equalClicked)
        self.backspaceButton.clicked.connect(self.backspaceClicked)
        unaryOperator = [self.squareRootButton, self.powerButton, self.reciproca
        for i in unaryOperator:
            i.clicked.connect(self.unaryOperatorClicked)
            self.changeSignButton.clicked.connect(self.changeSignClicked)

    def digitClicked(self):
        """
        使用者按下數字鍵，必須能夠累積顯示該數字
        當顯示器已經為 0，再按零不會顯示 00，而仍顯示 0 或 0.0
        """
        button = self.sender()
```

3-3

3-3 迴圈

- 迴圈說明:迴圈是計算機科學運算領域用語，一種常見的控制流程。迴圈在程式中只出現一次，但可能會連續執行多次程式碼。迴圈中的程式碼會執行特定次數，或是執行到特定條件成立時結束，又或者只針對某一集合中的所有項目都執行一次。

1. For 迴圈:

Python 的 for 迴圈 (for loop) ，用於取得複合資料型態(compound data type) 元素 (element) ，因此，具有重複結構的程式通常需要下列三項基本任務

- A. 控制變數初始設定
- B. 迴圈結束條件測試
- C. 調整控制變數的值

for 迴圈只需要取得元素的控制變數 (variable) ，其餘條件 (condition) 測試與調整控制變數的部份，迴圈中會自動完成。

```

1 #ex1 簡單的 for 迴圈範例
2 def dia(w):
3     for i in range(1,w):
4         print((w-i)*" "+"i**")
5     for i in range(w):
6         print(i*" "+"(w-i)*")
7 dia(4)

```

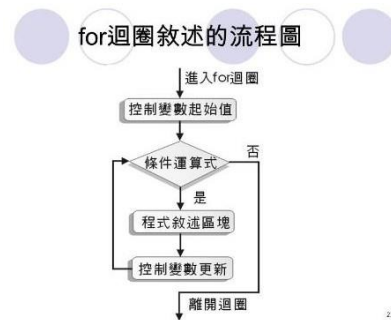
Filename: input file name .py Save

Run Output 清除 清除畫布

```

*
**
***
****
***
**
*
<completed in 64.00 ms>

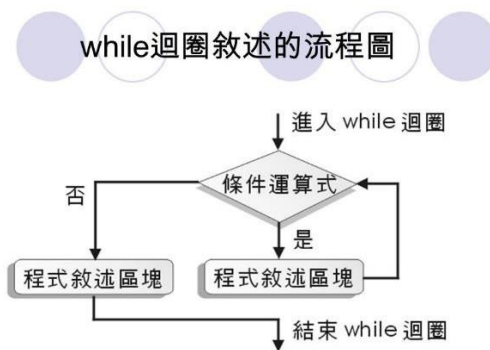
```



(For 迴圈)

2. While 迴圈

while 構成 Python 中迴圈的一種，常用於沒有確定重複次數的迴圈，同時 **while** 後到冒號間的運算式 (**expression**) 為迴圈結束的條件 (**condition**) 測試，即為迴圈開始前進行迴圈結束條件的測試。由於 **while** 陳述僅需迴圈的結束條件測試，所以有關控制變數 (**variable**) 的初始設定及調整，這都需要放在其它地方。



3-4 判斷式

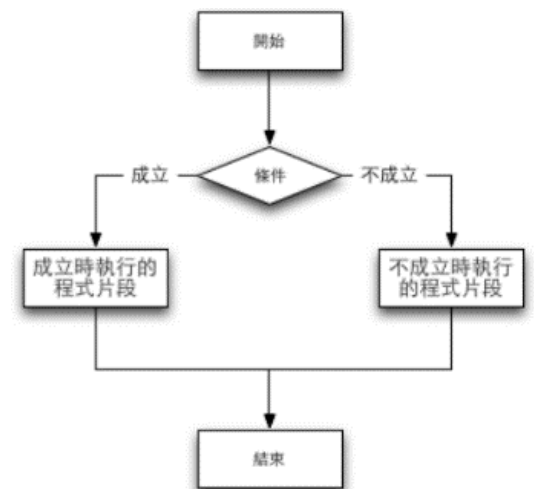
1. 比較運算 語法

- 相等 $a==b$
- 不相等 $a!=b$
- 大於 $a>b$
- 小於 $a<b$
- 大於等於 $a>=b$
- 小於等於 $a<=b$

2. if-else 判斷式

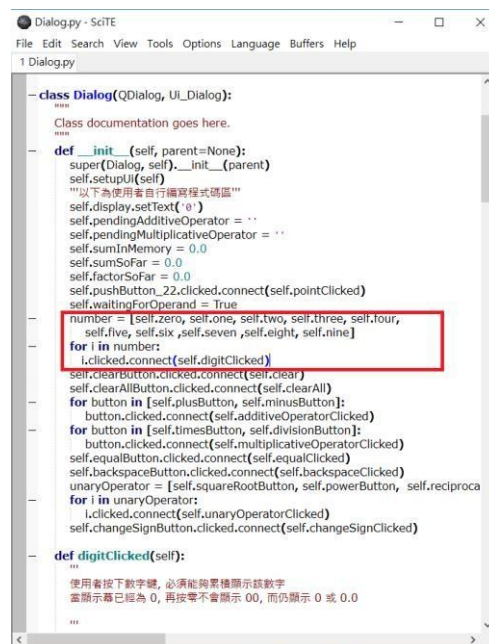
if 條件：成立時執行的程式片段

else：不成立時執行的程式片段



3-5 數列

就是我把我需要的資源全放一起，這樣讓握的後續作業就不用一個一個去找，而是只著重在我所放資源的地方找。

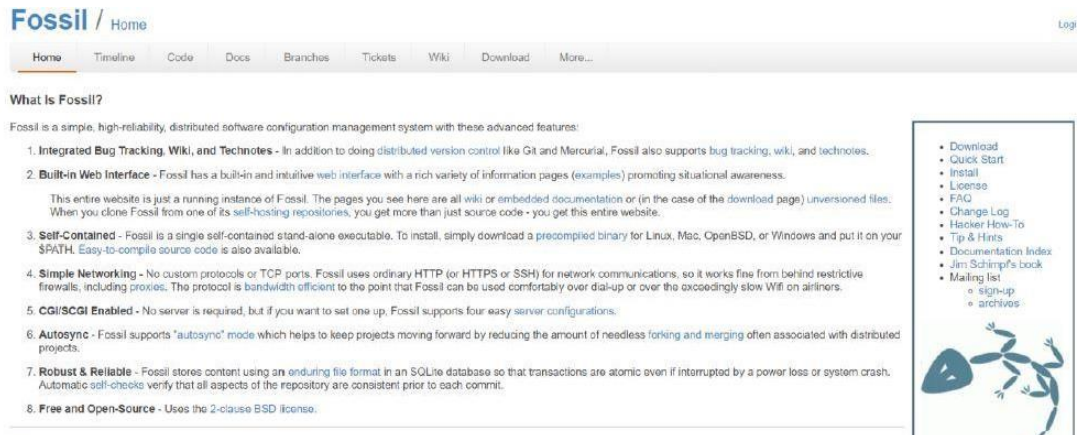


```
class Dialog(QDialog, Ui_Dialog):
    """
    Class documentation goes here.
    """
    def __init__(self, parent=None):
        super(Dialog, self).__init__(parent)
        self.setupUi(self)
        """以下為使用者自行編寫程式碼區"""
        self.display.setText('0')
        self.pendingAdditiveOperator = ''
        self.pendingMultiplicativeOperator = ''
        self.sumInMemory = 0.0
        self.sumSoFar = 0.0
        self.factorSoFar = 0.0
        self.pushButton_22.clicked.connect(self.pointClicked)
        self.waitingForOperand = True
        number = [self.zero, self.one, self.two, self.three, self.four,
                  self.five, self.six, self.seven, self.eight, self.nine]
        for i in number:
            i.clicked.connect(self.digitClicked)
        self.clearButton.clicked.connect(self.clear)
        self.clearAllButton.clicked.connect(self.clearAll)
        for button in [self.plusButton, self.minusButton]:
            button.clicked.connect(self.additiveOperatorClicked)
        for button in [self.timesButton, self.divisionButton]:
            button.clicked.connect(self.multiplicativeOperatorClicked)
        self.equalButton.clicked.connect(self.equalClicked)
        self.backspaceButton.clicked.connect(self.backspaceClicked)
        unaryOperator = [self.squareRootButton, self.powerButton, self.reciproca
        for i in unaryOperator:
            i.clicked.connect(self.unaryOperatorClicked)
        self.changeSignButton.clicked.connect(self.changeSignClicked)

    def digitClicked(self):
        """
        使用者按下數字鍵，必須能夠累積顯示該數字
        當顯示幕已經為 0，再按零不會顯示 00，而仍顯示 0 或 0.0
        """
```

第四章 倉儲系統

Fossil



Links For Fossil Users:

- Frequently Asked Questions
- The concepts behind Fossil. Another viewpoint.

Fossil scm 是本學期為使我們先熟悉管理版次及編寫 blog 所練習之程式。

1.使用 cmd 命令提示字元在 fossil 的目錄下打 fossil clone，並在後面打出自己所想要的網址。

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

Y:\>cd tmp
Y:\tmp>cd fossil
2:\tmp\fossil>fossil clone https://40623118@pca.kiml.info/40623118_40623118.fossil
```

```
Y:\tmp\fossil>mkdir wd_
```

3.在建立(wd)子目錄以方便管理資料，也讓自己方便找檔案。

```
Y:\tmp\fossil\wd>mkdir 40623118_
```

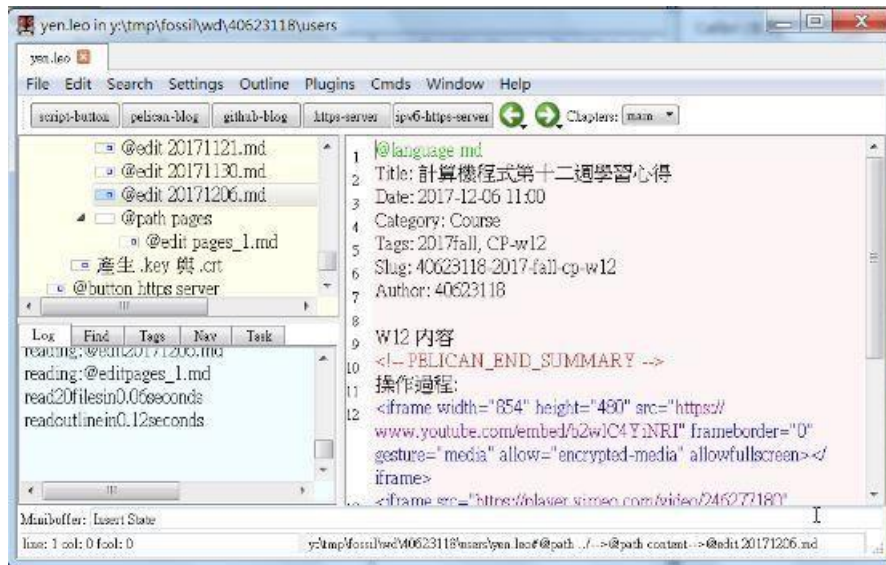
4. 在子目錄開啟前兩個目錄(為 fossil 目錄)的檔案。

```
Y:\tmp\fossil\wd\40623118>fossil open ../../40623118.fossil
```

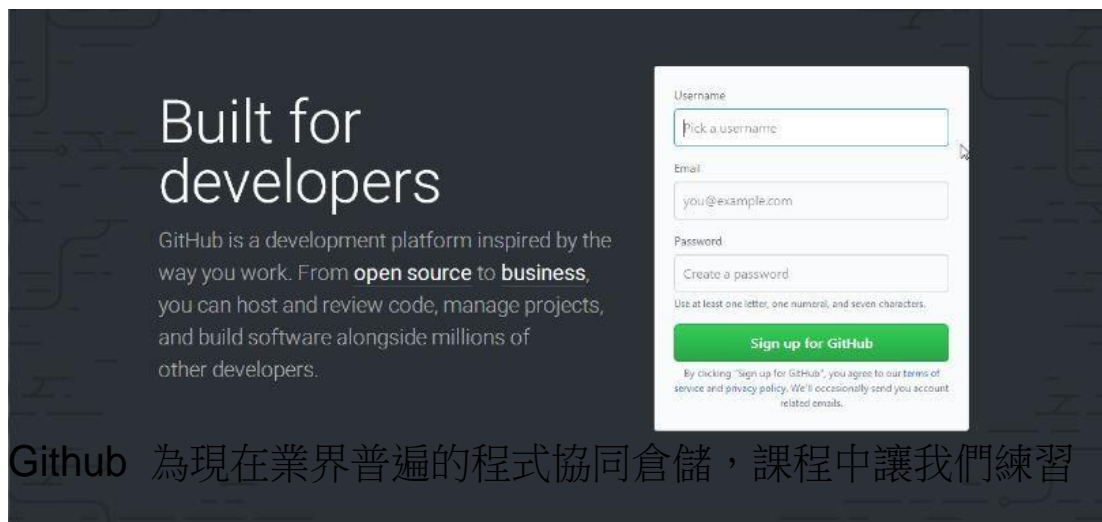
5. 在 SciTE 中開啟於 1. 目錄下的 launchLeo 啟動 Leo。



6. 開啟位於 user 的 .leo 檔以編寫 blog，利用 https-server 按鈕啟動近端檢視，並使用 peliccan-blog 按鈕將所編寫之文字轉換成近端程式碼，檢查無誤後再用 github-sever。



7. 最後再用 `fossil add .` 及 `fossil commit -m ""` 指令將完成編寫之 blog 推送。



Github 為現在業界普遍的程式協同倉儲，課程中讓我們練習協同計算機程式與報告，幫助同學熟悉未來開發程式或課程

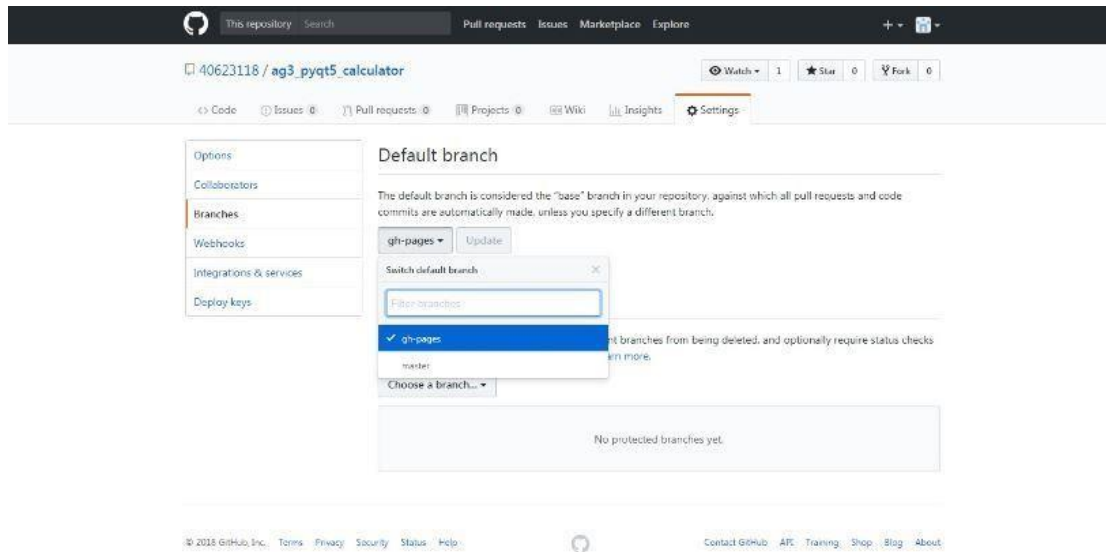
會遇到的作業方式。

The screenshot shows the GitHub 'Create a new repository' form. At the top, there's a search bar and navigation links like 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main heading is 'Create a new repository' with a subtext: 'A repository contains all the files for your project, including the revision history.' Below this, there are fields for 'Owner' (set to '40623118') and 'Repository name' (set to '專案名稱'). A hint suggests repository names should be lowercase with dashes. There's a 'Description (optional)' text area. Under 'Visibility', 'Public' is selected with the note 'Anyone can see this repository. You choose who can commit.' 'Private' is also an option. A checkbox 'Initialize this repository with a README' is checked, with a note: 'This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.' Below this are dropdowns for 'Add .gitignore: Python' and 'Add a license: GNU General Public License v3.0'. At the bottom is a green 'Create repository' button.

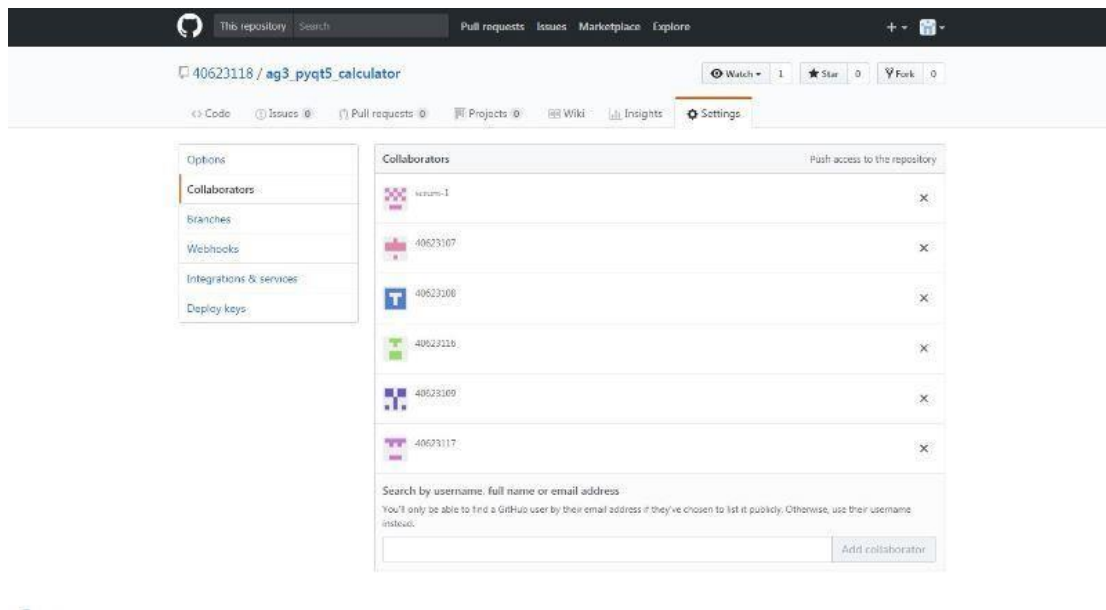
1.建立一個新的專案並參照格式設定。

The screenshot shows a GitHub repository page. At the top, it displays statistics: '35 commits', '2 branches', '0 releases', '7 contributors', and 'GPL-3.0' license. Below this is a 'Branches' section with a dropdown menu showing 'gh-pages' selected. To the right, there's a table of commit history. The table has two columns: the commit message and the time since the commit. The commits listed are: 'edit ag3leo name test1' (7 days ago), 'edit ag3leo name test1' (7 days ago), 'Initial commit' (28 days ago), 'Initial commit' (28 days ago), 'Initial commit' (28 days ago), and 'edit ag3.txt error' (22 days ago). At the bottom, there's a file list showing 'README.md' and 'ag3.txt'.

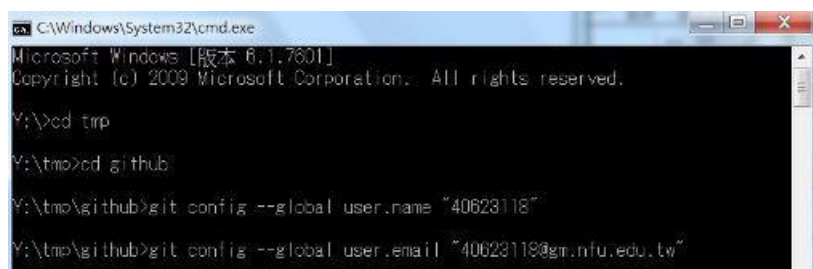
2.於Branch分支中新增gh-pegas。



3.進入Settings中的Branches將master修改成gh-pegas 。



4.進入 Collaborators 邀請協同學員 。



5.完成分配之作業如要推送需先設定帳號 email

```
Y:\tmp\github>git add .
```

```
Y:\tmp\github>git commit "填入自己所想要的名稱以方便尋找檔案"
```

```
Y:\tmp\github>git push
```

6.依序使用以下指令確認無誤後方可。

第五章 Calculator 程式

一、數字按鍵處理

1-1 數字按鍵程式

點按數字按鍵，將會送出該按鍵的訊號

儲存按鍵發出的訊號後，判斷視窗上是否有計算中的數字

如果無計算中的數字且訊號數字為字串 0 則不動作

如果無計算中的數字且訊號數字不為字串 0 則顯示並堆疊顯示所按下之數字

程式如下：

```
number = [self.zero, self.one, self.two, self.three, self.four,
self.five, self.six, self.seven, self.eight, self.nine]

for i in number:
    i.clicked.connect(self.digitClicked)

def digitClicked(self):
    button = self.sender()
    if self.display.text() == '0' and int(button.text()) == 0.0:
        return
    if self.waitingForOperand:
        self.display.clear()
        self.waitingForOperand = False
    self.display.setText(self.display.text() + button.text())
```

二、加減運算按鍵處理

按下加或減運算子按鍵時，程式設定以 `additiveOperatorClicked()` 處理。

進入 `additiveOperatorClicked()` 後，必須先查是否有尚未運算的乘或除運算子，因為必須先乘除後才能加減。

先處理乘與除運算後，再處理加或減運算後，將 `sumSoFar` 顯示在 `display` 後，必須重置 `sumSoFar` 為 0，表示運算告一段落。

程式如下：

```
for button in [self.plusButton, self.minusButton]:
    button.clicked.connect(self.additiveOperatorClicked)

    clickedButton = self.sender()
    clickedOperator = clickedButton.text()
    operand = float(self.display.text())
    if self.pendingMultiplicativeOperator:
        """
        計算：self.calculate(乘數或除數, 運算子)
        回傳 bool 以知道運算成功與否
        Python 文法：[if not 結果:] 當失敗時執行 self.abortOperation()。
        """
        if not self.calculate(operand, self.pendingMultiplicativeOperator):
            self.abortOperation()
            return
        self.display.setText(str(self.factorSoFar))
        operand, self.factorSoFar = self.factorSoFar, 0.0
        self.pendingMultiplicativeOperator = ''
    if self.pendingAdditiveOperator:
        """
        同上
        """
        if not self.calculate(operand, self.pendingAdditiveOperator):
            self.abortOperation()
            return
        self.display.setText(str(self.sumSoFar))
    else:
        self.sumSoFar = operand
    self.pendingAdditiveOperator = clickedOperator
    self.waitingForOperand = True
```

```
def calculate(self, rightOperand, pendingOperator):  
    """計算"""  
    if pendingOperator == "+":  
        self.sumSoFar += rightOperand  
    elif pendingOperator == "-":  
        self.sumSoFar -= rightOperand  
    elif pendingOperator == "*":  
        self.factorSoFar *= rightOperand  
    elif pendingOperator == "/":  
        if rightOperand == 0.0:  
            return False  
        self.factorSoFar /= rightOperand  
    return True
```

三、乘除運算按鍵處理

3-1 乘除運算程式

按下乘或除運算子按鍵時，程式設定以

`multiplicativeOperatorClicked()` 處理

進入 `multiplicativeOperatorClicked()` 後，無需檢查是否有尚未運算的加或減運算子，因為乘除運算有優先權

先處理乘與除運算後，再處理加或減運算，將 `sumSoFar` 顯示在 `display` 後，必須重置 `sumSoFar` 為 0，表示運算告一段落

程式如下：

```
'''乘或除按下後進行的處理方法'''
clickedButton = self.sender()
clickedOperator = clickedButton.text()
operand = float(self.display.text())
if self.pendingMultiplicativeOperator:
    '''
    同加減法
    '''
    if not self.calculate(operand, self.pendingMultiplicativeOperator):
        self.abortOperation()
        return
    self.display.setText(str(self.factorSoFar))
else:
    self.factorSoFar = operand
    self.pendingMultiplicativeOperator = clickedOperator
    self.waitForOperand = True
```


四、小數點與變號按鍵處理

4-1 小數點按鍵程式

使用者按下小數點按鍵後，以 `pointClicked()` 方法處理，直接在 `display` 字串中加上 "." 字串數值變號按鍵處理。

程式如下：

```
self.pushButton_22.clicked.connect(self.pointClicked)
```

```
if self.waitingForOperand:
    self.display.setText('0')

if "." not in self.display.text():
    self.display.setText(self.display.text() + ".")
```

4-2 變號按鍵程式

使用者按下變號按鍵後，由 `changeSignClicked()` 處理，若顯示幕上為正值，則在 `display` 字串最前面，疊上 "-" 字串。

假如顯示幕上為負值，則設法移除 `display` 上字串最前方的 "-" 字元。

程式如下：

```
self.changeSignButton.clicked.connect(self.changeSignClicked)
```

```
text = self.display.text()
value = float(text)
if value > 0.0:
    text = "-" + text
elif value < 0.0:
    text = text[1:]
self.display.setText(text)
```

五、上一步、清除與等號按鍵處理

AC 或 C 可以清除掉所有的記憶

AC

All Clear 全部清除鍵

按下 **AC** 按鍵，可將整個運算式清除。

範例： $235+882-762$ ，按下 **AC** 按鍵，則整個運算資料接清除

C

Clear 清除鍵

按下 **C** 按鍵，可清除運算式的最後數據

範例： $235+882-762$ ，按下 **C** 按鍵，則清除 **762** 這組數據

= 可以鍵算出最後結果

Equal 等於

按下等於按鍵，可使前面運算得出結果

範例： $235+882-762$ ，按下 **=** 按鍵，則會計算出 **355** 這組數據結果

```
def clear(self):
```

```
    """清除鍵按下後的處理方法"""
```

```
    #留著前面的數字
```

```
    if self.waitingForOperand:
```

```
        #下面不會執行
```

```
        return
```

```
    #清除
```

```
    self.display.setText('0')
```

```
    self.waitingForOperand = True
```

```
def clearAll(self):
```

```
    """全部清除鍵按下後的處理方法"""
```

```
    #重設預設值
```

```
    self.sumSoFar = 0.0
```

```
    self.factorSoFar = 0.0
```

```
    self.pendingAdditiveOperator = ''
```

```
    self.pendingMultiplicativeOperator = ''
```

```
    self.display.setText('0')
```

```
    self.waitingForOperand = True
```

```
def equalClicked(self):
```

```
    """等號按下後的處理方法"""
```

```
self.equalButton.clicked.connect(self.equalClicked)
```

```
'''等號按下後的處理方法'''
operand = float(self.display.text())
'''
同乘除
'''
if self.pendingMultiplicativeOperator:
    if not self.calculate(operand, self.pendingMultiplicativeOperator):
        self.abortOperation()
    return
operand = self.factorSoFar
self.factorSoFar = 0.0
self.pendingMultiplicativeOperator = ''
'''
同加減
'''
if self.pendingAdditiveOperator:
    if not self.calculate(operand, self.pendingAdditiveOperator):
        self.abortOperation()
    return
self.pendingAdditiveOperator = ''
else:
    self.sumSoFar = operand
self.display.setText(str(self.sumSoFar))
self.sumSoFar = 0.0
self.waitingForOperand = True
```

以上就是寫出這些功能的程式

六、倒退鍵

- 使數字能夠刪掉最右方一個

範例:45 按下 `backspace` = 4

```
self.backspaceButton.clicked.connect(self.backspaceClicked)

text = self.display.text()[::-1]
if not text:
    text = '0'
    self.waitingForOperand = True
self.display.setText(text)
```

七、記憶體按鍵與其他按鍵處理

6-1 記憶體按鍵程式

功能

MC：就是將目前記憶的數字歸零。

MR：將當前計算出來的數字呈現出來。

MS：無視目前記憶多少數字，直接以當前數字取代記憶中的數字。

M+：記憶當前的數字，「加入累加數字當中」。

按鍵處理

`clearMemory()` 方法與 "MC" 按鍵對應, 清除記憶體中所存

`sumInMemory` 設為 0

`readMemory()` 方法與 "MR" 按鍵對應, 功能為讀取記憶體中的數值, 因此將 `sumInMemory` 顯示在 `display`, 作為運算數

`setMemory()` 方法則與 "MS" 按鍵對應, 功能為設定記憶體中的數值, 因此取 `display` 中的數字, 存入 `sumInMemory`

`addToMemory()` 方法與 "M+" 按鍵對應, 功能為加上記憶體中的數值, 因此將 `sumInMemory` 加上 `display` 中的數值

因為 `setMemory()` 與 `addToMemory()` 方法, 都需要取用 `display` 上的數值, 因此必須先呼叫 `equalClicked()`, 以更新 `sumSoFar` 與 `display` 上的數值

導入的模組:

```
# 清除記憶
self.clearMemoryButton.clicked.connect(self.clearMemory)

# 讀取記憶
self.readMemoryButton.clicked.connect(self.readMemory)

# 設定記憶
self.setMemoryButton.clicked.connect(self.setMemory)

# 加入記憶體
self.addToMemoryButton.clicked.connect(self.addToMemory)
```

```
import math
```

*程式:

```

def clearMemory(self):
    self.sumInMemory = 0.0

def readMemory(self):
    self.display.setText(str(self.sumInMemory))
    self.waitingForOperand = True

def setMemory(self):
    self.equalClicked()
    self.sumInMemory = float(self.display.text())

def addToMemory(self):
    self.equalClicked()
    self.sumInMemory += float(self.display.text())

```

6-2 其他按鍵程式

功能：

- Sqrt : 開根號
- X^2 : 平方
- 1/x : 倒數

按鍵處理：

- Sqrt, x^2 與 1/x 等按鍵的處理方法為 unaryOperatorClicked(), 與數字按鍵的點按回應相同, 透過 sender().text() 取得按鍵上的 text 字串

- `unaryOperatorClicked()` 方法隨後根據 `text` 判定運算子後, 利用 `display` 上的運算數進行運算後, 再將結果顯示在 `display` 顯示幕
- 若進行運算 `Sqrt` 求數值的平方根時, 顯示幕中為負值, 或 `1/x` 運算時, `x` 為 `0`, 都視為無法處理的情況, 以呼叫 `abortOperation()` 處理
- `abortOperation()` 方法則重置所有起始變數, 並在 `display` 中顯示 "####"
- 直接運算子處理結束前, 運算結果會顯示在 `display` 中, 而且運算至此告一段落, 計算機狀態應該要回復到等待新運算數的階段, 因此 `waitingForOperand` 要重置為 `True`

導入的模組:

單一運算子

```
for i in unaryOperator:
    i.clicked.connect(self.unaryOperatorClicked)
self.changeSignButton.clicked.connect(self.changeSignClicked)
```

程式:

```

'''單一運算元按下後處理方法'''
#pass
clickedButton = self.sender()
clickedOperator = clickedButton.text()
operand = float(self.display.text())

if clickedOperator == "Sqrt":
    if operand < 0.0:
        self.abortOperation()
        return

    result = math.sqrt(operand)
elif clickedOperator == "X^2":
    result = math.pow(operand, 2.0)
elif clickedOperator == "1/x":
    if operand == 0.0:
        self.abortOperation()
        return

    result = 1.0 / operand

self.display.setText(str(result))
self.waitingForOperand = True

```

第六章 心得

1.fossil scm 心得

40623107 簡稜雅:

Fossil 是我第一個接觸的倉儲系統，一開始雖然覺得很陌生的點，但了解後才發現 fossil 非常的簡單又方便。有了 fossil 還能記錄在什麼時候更改了資料。

40623108 林郁涵:

雖然一開始對程式完全不了解，但經過多次練習和看影片，就能夠比較清楚了。

40623109 李如芳:

上網查詢後，知道最近企業都使用(資料)倉儲，存取實體上的及資訊系統上的檔案統整，只要有倉儲，它會幫我存取我所有做過的資料及時間點，讓我能查看並修改資料。

40623116 楊子毅

當初剛學習 **Fossil** 時，我完全不懂的到底怎麼用，回宿舍都要繼續問同學，我覺得非常的困難。

40623118 楊秉澤

雖然第一次接觸到 **Fossil** 這種倉儲，而且剛開始也感到困難，不過在同學的幫助下，讓我了解到原來 **Fossil** 如此簡單。

40623117楊智傑

我覺得**Fossil**好用的地方是，管理資料真的很方便，也可以隨時看到上一次推送檔案的時間，非常的方便。

2.網誌心得

40623107 簡稜雅:

Leo Edit 是我覺得比較容易上手的網誌編輯器，第一次寫網誌覺得完全無法理解，寫了很多次之後就像在寫日記一樣。

40623108 林郁涵:

一開始推網誌的時候根本不知道要怎麼推，但看過影片再加上多次練習就可以比較上手了。

40623109 李如芳:

我能將學習心得及一些影片圖片放在網上進行編輯及解說，讓它留存在網路上，並給予他人去瀏覽。

40623116 楊子毅

製作網誌的時候，總會遇到很多推不上去的問題，每次都要找很久才找到問題點在哪，我覺得非常的困擾，需要多加熟練。

40623118 楊秉澤

我認為它就像一個日記一樣，只是多了網路來編輯它，利用 **Leo** 來幫助就顯的簡單許多。

40623117楊智傑

Leo可以讓我們在沒有網路的狀態下工作，能將修改的資料推至近端，等待重新連接上網路時推送到遠端，讓我們工作不再受網路的限制，無時無刻都可以做。

3. Github 心得

40623107 簡稜雅:

Github 是個能夠多人協同進行調和的倉儲，讓我充分的理解團隊合作的重要。

40623108 林郁涵:

在 Github 倉儲中，能夠用協同系統一起完成程式，覺得很方便也能節省時間。

40623109 李如芳:

在使用 Github 時，會常使用到 git 的指令，當我對哪些指令覺得不清楚時，我會在網上查詢相關指令，讓我對 git 只了解更多，也學到新知識。

40623116 楊子毅

接觸 github 時，覺得指令的部分跟 fossil 很像，所以覺得蠻簡單的，加上大家一起共同做計算機，我認為這是非常好的一個方式，讓我們一起完成作業。

40623117 楊智傑

Github 協同倉儲是在現在這種分工作業的時代一個非常具有指標性的工具，隨時可以看到哪一位協同者在什麼時間修改了什麼程式，方便看到團隊工作的進度，以提高工作的效率。

40623118 楊秉澤

在老師的規劃下，先讓我們了解 **fossil** 的原理，再交 **Github** 就覺得簡單了很多，而小組討論也讓我們會尋找衝突的方，更學到了許多知識。

第七章 結論及建議

1.結論

透過github的協同倉儲讓我們知道，一個人的力量或許有限，但許多人一起協同做一件事情時，不但能事半功倍，還能透過互相開會討論，不斷的精進自己與團隊，未來不僅僅是計算機，若能將協同工作這個模式帶到職場，一定能追求到最好的工作效率。

2.建議

希望老師之後能交給我們更多方便的程式，讓我們能更有能力面對未來瞬息萬變的世界。