

國立虎尾科技大學

機械設計工程系

計算機程式 bg5 期末報告

PyQt5 事件導向計算器

PyQt5 Event-Driven Calculator Project

學生：

設計一乙 40623219

設計一乙 40623220 蔡崇廷

設計一乙 40623221 蔡和勳

設計一乙 40623228 陳永錫

設計一乙 40623229 陳宥安

設計一乙 40623230 陳柏亦

指導教授：嚴家銘

2017.12.25

# 目錄

# 表目錄

## 圖目錄

# 第一章 前言

計算器程式期末報告前言

## 第二章 可攜程式系統介紹

### 2.1 啟動與關閉

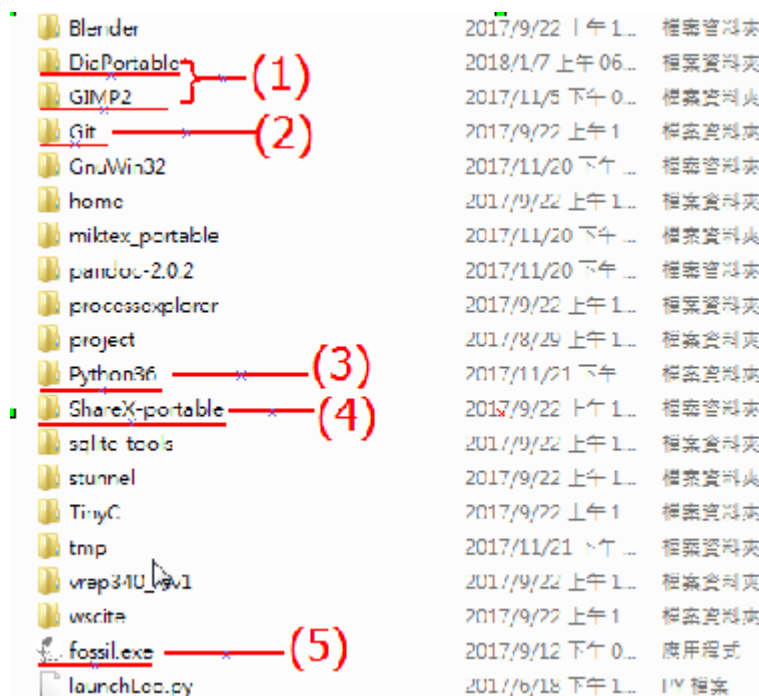


圖 2.1: system-1

可攜程式: 因為在不同的電腦擁有的程式也會有所不同所以使用可攜程式的話可以方便在任何電腦執行自己熟悉的程式也可使用建立自己習慣的開發環境

(1)GIMP2-可以做修剪圖片或是裁切圖片

DiaPortable-可繪製圖形幫助註解圖片

(2)GitHub-SCM(組態管理系統) 的一種, 特點多人協同,gh-pages, 公開 (不公開要花錢)

(3)Python36-在不同電腦都可以進行 Python 的程式開發

(4)ShareX-可截取螢幕畫面, 與錄製影片

(5)Fossil-SCM(組態管理系統) 的一種, 特點 Totally control 完全可以自己控制伺服到客戶端

### 2.2 啟動與關閉 2

(1)miktex\_portable-包含了 TeX 及其相關程式, 這些工具是以 TeX/LaTeX 所構成的






















 Blender	2017/9/22 上午 1...	檔案資料夾
 DiaPortable	2018/1/7 上午 06...	檔案資料夾
 GIMP2	2017/11/5 下午 0...	檔案資料夾
 Git	2017/9/22 上午 1...	檔案資料夾
 GnuWin32	2017/11/20 下午 ...	檔案資料夾
 home	2017/9/22 上午 1...	檔案資料夾
 <u>miktex_portable</u> (1)	2017/11/20 下午 ...	檔案資料夾
 <u>pandoc-2.0.2</u> (2)	2017/11/20 下午 ...	檔案資料夾
 PNG	2018/1/7 下午 11...	檔案資料夾
 processexplorer	2017/9/22 上午 1...	檔案資料夾
 project	2017/8/29 上午 1...	檔案資料夾
 Python36	2017/11/21 下午 ...	檔案資料夾
 ShareX-portable	2017/9/22 上午 1...	檔案資料夾
 sqlite-tools	2017/9/22 上午 1...	檔案資料夾
 stunnel	2017/9/22 上午 1...	檔案資料夾
 TinyC	2017/9/22 上午 1...	檔案資料夾
 tmp	2017/11/21 下午 ...	檔案資料夾
 vrep340_rev1	2017/9/22 上午 1...	檔案資料夾
 wscite	2017/9/22 上午 1...	檔案資料夾
 .fossil.exe	2017/9/12 下午 0...	應用程式
 launchLeo.py	2017/6/18 下午 1...	PY 檔案

圖 2.2: system-2

(2)pandoc-2.0.2-以命令列形式實現與用戶的互動，可支援多種作業系統  
可攜程式系統介紹

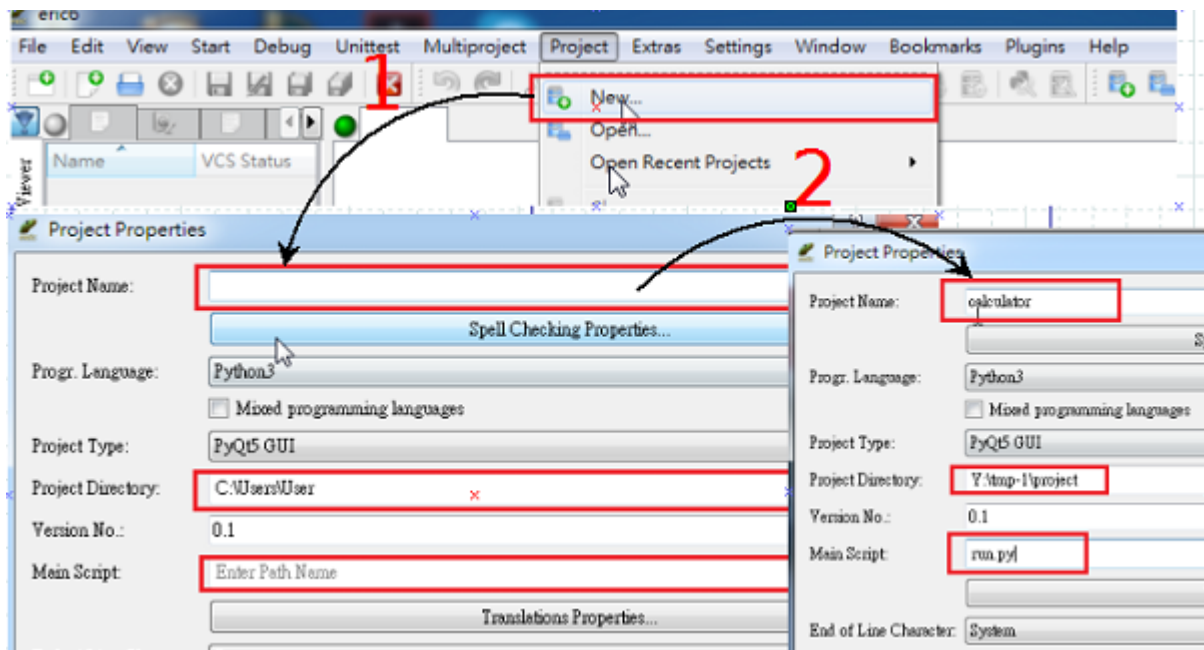


## 第三章 Calculator 程式

Calculator 程式細部說明

### 3.1 建立對話框

step1



step2

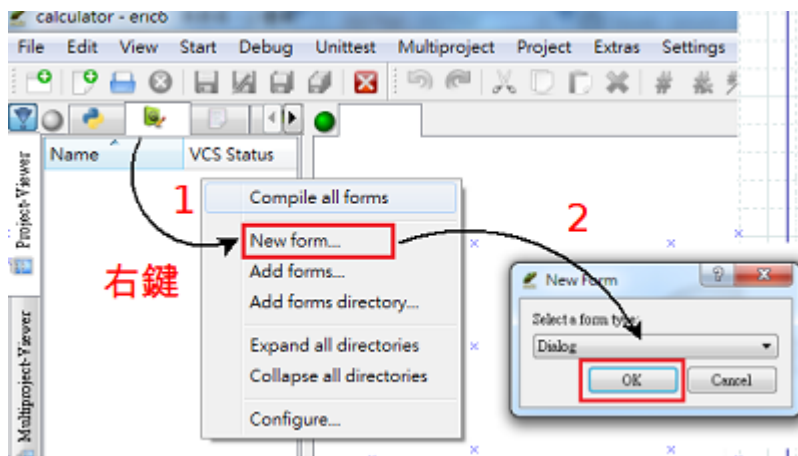
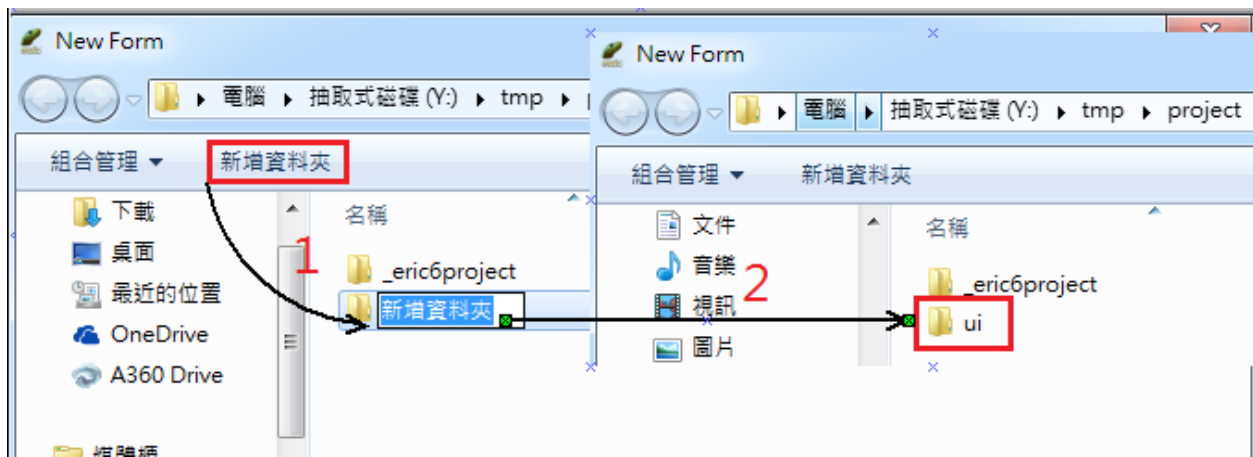


圖 3.1: newform

step3



step4

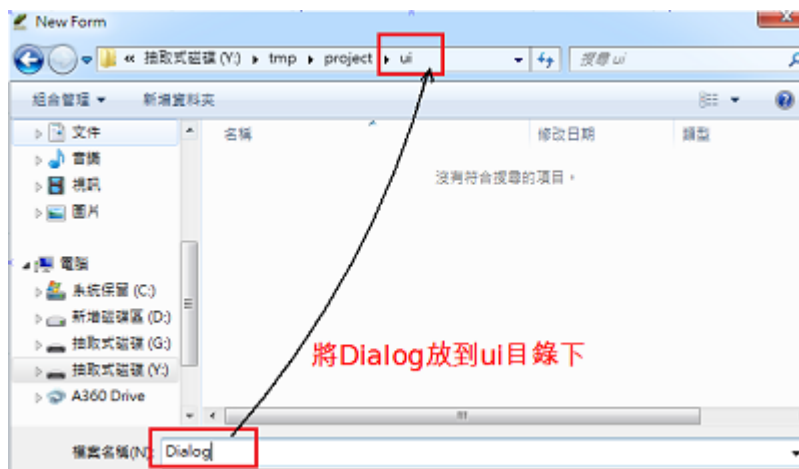


圖 3.2: Dialog into ui

step5

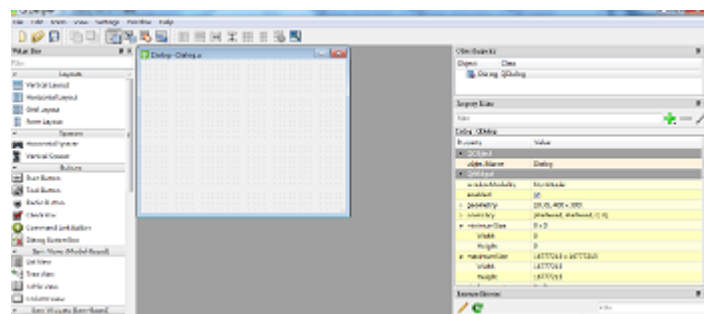


圖 3.3: qtdesigner

## 3.2 建立按鈕

step1

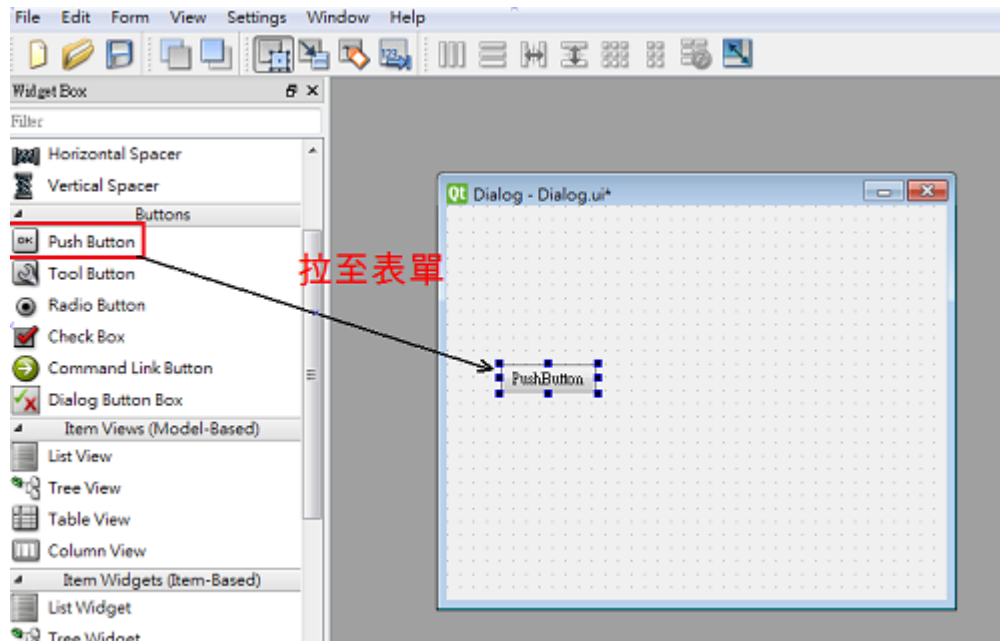


圖 3.4: button

step2

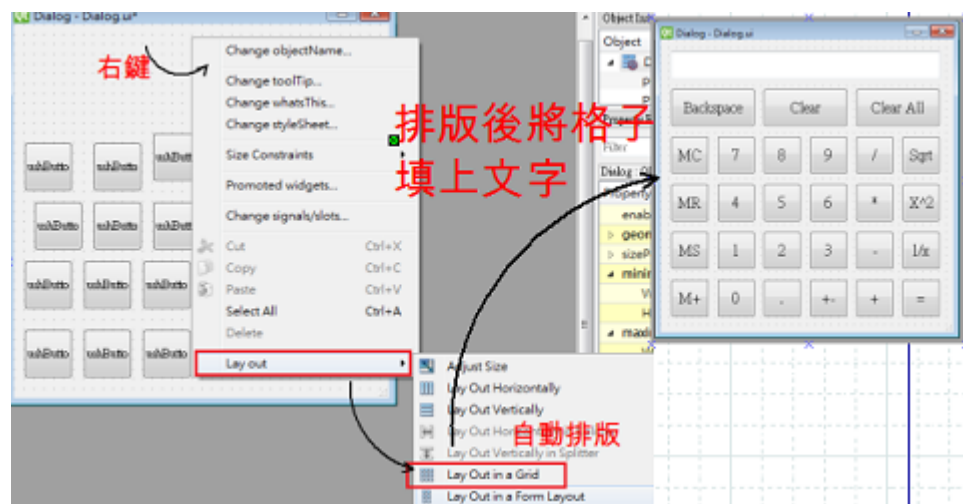


圖 3.5: grid

以上是由 Qt designer 製作  
Qt designer 詳細請查閱第五章

### 3.3 建立程式碼

40623220

乘除

```
def multiplicativeOperatorClicked(self):  
    """乘或除按下後進行的處理方法"""  
    # pass  
    clickedButton = self.sender()  
    clickedOperator = clickedButton.text() 變數定義  
    operand = float(self.display.text()) 浮點數  
  
    if self.pendingMultiplicativeOperator:  
        if not self.calculate(operand, self.pendingMultiplicativeOperator):  
            self.abortOperation()  
            return 如果按的按鍵不符calculator跳至abortOperation  
    else:  
        self.display.setText(str(self.factorSoFar)) 將目前累計運算數  
        self.factorSoFar = operand 顯示在 display 上  
        self.pendingMultiplicativeOperator = clickedOperator  
        self.waitForOperand = True  
        等待運算為真
```

圖 3.6: mult

變號

```
def changeSignClicked(self):  
    """變號鍵按下後的處理方法"""  
    #pass  
    text = self.display.text()  
    value = float(text) 浮點數  
  
    if value > 0.0: 如果數值小於零的話在前面  
        text = "-" + text 加-，如果大於零則為正數  
    elif value < 0.0:  
        text = text[1:]  
  
    self.display.setText(text)  
    數值顯示在display
```

圖 3.7: change

計算

中斷運算

40623221

```
def calculate(self, rightOperand, pendingOperator):
```

```
    """計算""" 右運算數與等待運算子當作輸入
```

```
    #pass
```

```
    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
```

進入計算時, 用目前輸入的運算數值  
與 self.sumSoFar 執行計算

圖 3.8: calculator

```
def abortOperation(self):
```

```
    """中斷運算""" 清除後, display顯示erro
```

```
    # pass
```

```
    self.clearAll()
```

```
    self.display.setText("erro")
```

圖 3.9: abo

```

def unaryOperatorClicked(self):
#40623221 ''' 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

```

說明:按下1/x若分母為0則需中斷運算,若按下Sqrt且數字小於0也是中斷運算,  
按下X^2若數字為2則需運算2的平方..

```

def pointClicked(self):
#40623221 ''' 小數點按下. 後的處理方法''' #pass if self.waitingForOperand: self.display.setText('0')

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

    self.waitingForOperand = False

```

說明:若出現display,且在數字0後面沒出現小數點則將小數點顯示出來..

```

def clearAll(self):
#40623221 ''' 全部清除鍵按下後的處理方法''' #pass self.sumSoFar = 0.0 self.factorSoFar = 0.0
self.pendingAdditiveOperator = self.pendingMultiplicativeOperator = self.display.setText('0')
self.waitingForOperand = True

```

說明:按下clearall則把全部運算停止並將全部清除最後出現0

## 40623228

數字邏輯

```

self.display.setText('0')

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

for i in num_button:
    i.clicked.connect(self.digitClicked)
def digitClicked(self):
#40623228
    """
    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
    """
    #pass
    clickedButton = self.sender()
    digitValue = int(clickedButton.text())
    if self.display.text() == '0' and digitValue == 0.0:
        return
    if self.waitingForOperand:
        self.display.clear()
        self.waitingForOperand = False
    self.display.setText(self.display.text() + str(digitValue))

```

按鍵連接

加減邏輯

等號邏輯

40623229 ''' 回復鍵按下的處理方法''' #pass if self.waitingForOperand: return

```

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

self.display.setText(text)
if self.waitingForOperand:
    return
def clear(self):

```

'''清除鍵按下後的處理方法'''

```

plus_minus = [self.plusButton, self.minusButton]
for i in plus_minus:
    i.clicked.connect(self.additiveOperatorClicked)

    self.pendingAdditiveOperator = ""

def calculate(self, rightOperand, pendingOperator):
    """計算"""
    #pass
    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

    self.sumSoFar = 0.0

def additiveOperatorClicked(self):
    """等號按下後進行的處理方法"""
    #pass
    clickedButton = self.sender()
    clickedOperator = clickedButton.text()
    operand = float(self.display.text())

    if self.pendingMultiplicativeOperator:
        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:
        if not self.calculate(operand, self.pendingAdditiveOperator):
            self.abortOperation()
            return
        self.display.setText(str(self.sumSoFar))
    else:
        self.sumSoFar = operand

    self.pendingAdditiveOperator = clickedOperator
    self.waitForOperand = True

```

按鍵連接

判斷式

圖 3.10: additiveOperatorClicked

```

def equalClicked(self):
    """等號按下後進行的處理方法"""
    #pass
    operand = float(self.display.text())

    if self.pendingMultiplicativeOperator:
        if not self.calculate(operand, self.pendingMultiplicativeOperator):
            self.abortOperation()
            return
        operand = self.factorSoFar
        self.factorSoFar = 0.0
        self.pendingMultiplicativeOperator = ""

    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.waitForOperand = True

```

判斷式子中是否有乘除

判斷式子中是否有加減

圖 3.11: equalClicked



```
#pass
if self.waitingForOperand:
    return
```

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

```
self.waitingForOperand = True
```

#### **40623230**

```
def clearMemory(self):
```

```
#40623230 ''' 清除記憶體鍵按下後的處理方法''' #pass self.sumInMemory = 0.0
```

說明:按下MC鍵後將記憶的數字變為0

```
def readMemory(self):
```

```
#40623230 ''' 讀取記憶體鍵按下後的處理方法''' #pass self.display.setText(str(self.sumInMemory))
```

```
self.waitingForOperand = True
```

說明:按下MR鍵後把記憶的數字顯示出來

```
def setMemory(self):
```

```
#40623230 ''' 設定記憶體鍵按下後的處理方法''' #pass self.equalClicked() self.sumInMemory =
float(self.display.text())
```

說明:按下MS鍵後會把當前的數字取代記憶的數字

```
def addToMemory(self):
```

```
#40623230 ''' 放到記憶體鍵按下後的處理方法''' #pass self.equalClicked() self.sumInMemory +=
float(self.display.text())
```

說明:按下M+鍵會把當前數字與記憶的數字相加後並記憶

## 第四章 Python 程式語法

### Python 程式語法

#### 4.1 變數命名

##### Python3 變數命名規則與關鍵字

###### 一、Python 英文變數命名規格

1. 變數必須以英文字母大寫或小寫或底線開頭
2. 變數其餘字元可以是英文大小寫字母, 數字或底線
3. 變數區分英文大小寫
4. 變數不限字元長度
5. 不可使用關鍵字當作變數名稱

###### 二、Python3 的程式關鍵字, 使用者命名變數時, 必須避開下列保留字.

1. 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']

###### 2. 選擇好的變數名稱:

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

###### 計算機中的迴圈

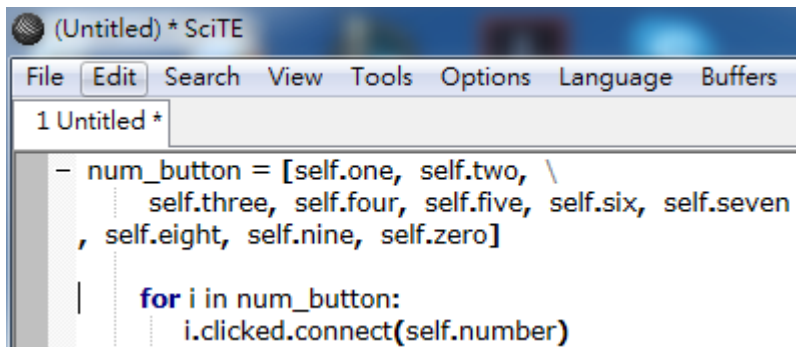


圖 4.1: for

## 4.2 print 函式

## 4.3 重複迴圈

## 4.4 判斷式

if 判斷式 1:

要處理的指令1

elif 判斷式 2:

要處理的指令2

else: 要處理的指令 3

注意事項

1. 每個判斷式的結束要加:
2. 要處理的指令不可以用 {} 括起來
3. 最後的 else 可以不用加

## 4.5 數列

python 的數列是一個 []

[] 是容器中能放容器也能放物件和字串

容器例如:list、set、dict、tuple

計算機中的數列



圖 4.2: s

## 第五章 PyQt5 簡介

說明 PyQt5 基本架構與程式開發流程

### 5.1 PyQt5 架構

PyQt5-GUI frame work , 圖形使用者介面軟體框架, 可以快速製做 GUI 界面程式, 是由一系列 Python 組成。超過 620 個類, 6000 和函數和方法

Qt5 原本是 C++ 語法之後用 Python 製作而成 PyQt

Qt 採用了 signal 和 slot 的概念來處理 GUI 程式中的用戶事件。PyQt 同樣支援這種方法。任何 Python 類型都可以定義 signal 和 slot, 並與 GUI 控制項的 signal 和 slot 相連線。

## 第六章 心得

期末報告心得

### 6.1 Fossil SCM

40623220 蔡崇廷

學習到如何用指令維護倉儲還有 wiki 和 timeline 使用

40623221 蔡和勳

fossil 是比較簡單的一些指令所以一開始我們已 fossil 當基礎練習了半學期將指令弄熟了其實跟 github 的指令是大同小異

40623228 陳永鋈 Fossil 使用起來比之前用雲端軟體的效率及感覺有很大的不一樣以版次來說能夠在每次都能做成一種版次就比雲端更可以處存自己設計的東西

40623229 陳宥安剛開始第一次接觸 fossil 完完全全不知道在搞啥也不知道做甚麼用的用在哪裡摸了兩個禮拜才懂得一些皮毛而已，後來經過老師講解才得以將前後串起來

40623230 陳柏亦 fossil 學習到的近遠端的倉儲維護還有版次處理

### 6.2 網誌心得

40623220 蔡崇廷

將自己每周的學習進度記錄在網誌上, 能夠為自己的學習能夠有個回想

40623221 蔡和勳

將平常上課所學的指令及程式碼運用讓自己的網誌可以更新

40623228 陳永鋈在每次上課都做一次網誌想筆記一樣有效的將遇到的問題及學過的東西記錄下來

網誌心得 40623229 陳宥安將每個禮拜老師的上課內容上傳到自己的網誌

40623230 陳柏亦可以將上課的內容記錄下來之後要尋找看便很容易

### 6.3 Github 協同倉儲

40623220 蔡崇廷學到如何在 github 建立倉儲, 也學到如何與其他人協同做報告跟計算機的專題, 也知道有衝突時該如何解決

40623221 蔡和勳透過與組員一起寫計算機程式能更快解決問題, 協同報告可以自動編排也很方便, 不用再將文字檔拉來拉去

40623228 陳永鋆雖然說跟 Fossil 一樣是 SCM 的東西可是加上了多人協同就有很大的不能像是怎麼解決衝突等等

Github 協同倉儲 40623229 陳宥安方便的討論如何製作一個報告不會因為不在同一個地方而無法討論印證了天涯若比鄰這句話

40623230 陳柏亦可以共同協同很方便, 更新檔案只要上傳下載就好, 不過更新時容易衝突, 需要去解決

## 6.4 學員心得

40623220 蔡崇廷從一開始的不知道這節課要幹嘛漸漸的進入狀況也能每周把自己的學習做個紀錄

40623221 蔡和勳雖然程式碼很多很嚇人, 但是如果有看老師教學影片就可以很快進入狀況可以學很快, 自己在花點時間就可以將程式完成

40623228 陳永鋆在經過整學期的課程我覺得, 程式會是必學的一項技能, 有了程式的幫忙才可以有效率的完成設計上的事情

40623229 陳宥安這 16 個禮拜一直處在懵懵懂懂的狀態下, 不懂的地方還是很多, 反覆看了影片我相信總有一天會摸出個所以然, 下學期希望也一直保持像這學期這樣的好狀態

40623230 陳柏亦剛開始完全不會到學期末, 已經把這學期的都理解的差不多, 比起開始已經不太討厭程式了

## 6.5 說明各學員任務與執行過程

說明各學員任務與執行過程

40623220 蔡崇廷做計算機和報告不像前七周是自己做自己的而現在是必須與同學協同所以更需要討論也必須更注意排版不然一定會亂掉

40623221 蔡和勳組員所負責的計算機程式內容不一樣若前面的按鈕和排版沒做好後面就沒辦法接下去做, 還有按鈕名稱也很重要, 錯誤的話會沒辦法執行計畫

40623228 陳永鋆必須將流程已有條理的方式逐步的完成不然很容易亂了手腳

40623229 陳宥安裏頭的程式語言其實不是很懂, 如何能將一整串的文字組在一塊, 加上一些符號竟然能夠成為一台計算機, 需要的是不停的動腦不停的失敗再試反覆的操作直到完成

40623230 陳柏亦自己的程序必須要做好, 不然可能因為自己的部分有出錯, 導致其他人無法完成, 所以需要討論好, 不然很容易出錯

## 第七章 結論

### 期末報告結論

#### 結論與建議

透過這次的小組期末報告合作，大家學會了使用共同倉儲來協同，理解了許多的語法，在操作上很熟練很多，了解到分工時每一個地方都很重要，必須討論好，若是自己部分有錯誤，會影響到其他人，再進行作業時必須確保不會出錯，這樣才能完成期末報告。

## 第八章 參考文獻