

國立虎尾科技大學

機械設計工程系

計算機程式 bg8 期末報告

PyQt5 事件導向計算器

PyQt5 Event-Driven Calculator Project

學生：

設計一乙 40623244 林俊鎧

設計一乙 40623240 何冠均

設計一乙 40623241 郭祐齊

設計一乙 40623242 高宇辰

設計一乙 40623243 盧逸誠

設計一乙 40623245 練峪愷

指導教授：嚴家銘

2018.01.08

## 摘要

這是第八組的計算機概要，利用各自的 `leo` 檔分工合作，用協同的方式來做出計算機

- 以老師原本 `user.leo` 為底。
- 另存為學號.`leo` 來進行編輯
- 使用 `github` 協同。
- Q&A 用報告 `SciTE` 來做文字檔編輯，進行討論

本研究的重點在於練習計算機編排、`python` 練習、`pyqt5` 練習

# 目錄

摘要	i
目錄	ii
表目錄	iii
圖目錄	iv
第一章 數字按鍵處理	1
1.1 數字按鍵	1
1.2 心得	2
第二章 加減運算按鍵處理	3
2.1 加減運算	3
第三章 乘除運算按鍵處理	4
3.1 乘或除按鍵處理	4
3.2 計算機程式期末心得	4
第四章 小數點和變號按鍵處理	5
4.1 小數點	5
4.2 心得	5
第五章 退格按鍵處理以及清除按鍵處理	7
5.1 全部清除按鍵	7
5.2 退格按鍵	7
5.3 清除按鍵	8
5.4 心得	9
第六章 記憶體按鍵處理與直接運算 (運算方式)	10
6.1 記憶體按鍵處理	10
6.2 直接運算	11
6.3 運算方式	13
6.4 心得	14
第七章 參考文獻	15

# 表目錄

## 圖目錄

# 第一章 數字按鍵處理

## 1.1 數字按鍵

按下數字鍵 1 即出現 1 以此類推

而 0 按下一次就只會出現一次

所以 if self.display.text() == '0' and value == 0.0:

number=[self.one, self.two, self.three, self.four, self.five,

self.six, self.seven, self.eight, self.nine, self.zero]

for i in number:

i.clicked.connect(self.digitClicked)

程式碼

```
button = self.sender()
value = int(button.text())
if self.display.text() == '0' and value == 0.0:
    return
    self.wait = True
if self.wait :
    self.display.clear()
    self.wait = False
self.display.setText(self.display.text() + str(value))
```

## 1.2 心得

我的部分是設計計算機之中最簡單的部分，所以很快就可以完成，但是途中還是會遇到許多困難和不了解的地方，我會去詢問同學及組長，他們也很快的幫我解決問題，我學習起來並且運用在其他的部分。

## 第二章 加減運算按鍵處理

### 2.1 加減運算

確定運算為加或減, 在按下運算之前在 `display` 的數字為運算數納入乘除後的先乘除後加減, 且連續按下可累計運算

運算式:

```
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 = ''
```

納入連續按下加減, 可以目前的運算數及運算子累計運算

運算式:

```
if self.pendingAdditiveOperator: if not self.calculate(operand, self.pendingAdditiveOperator):  
self.abortOperation() return 顯示運算結果 self.display.setText(str(self.sumSoFar))
```

心得 —

經過這次協同完成計算機程式, 比以前更了解程式, 也透過分組方式來分工合作, 每個人也都有屬於自己的工作及目標, 遇到困難時也互相幫忙, 雖然過程不是很順利, 但也因此學到了如何解決及處理問題, 也知道在寫程式時會遇到的困難, 對於個人來說, 也算是接觸了較不一樣的領域, 對於自己也有所提升了



## 第三章 乘除運算按鍵處理

### 3.1 乘或除按鍵處理

乘和除按鍵設定

```
multiply_divide = [self.timesButton, self.divisionButton]
```

乘或除按鍵處理

1. 當按下乘或除按鍵時, 程式設定以 `multiplicative&Division OperatorClicked` 處理
2. 進入 `multiplicative&DivisionOperatorClicked` 後, 不需檢查是否有尚未運算的加或減運算子, 因為乘除有優先權
3. 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

乘和除按鍵舉例

$$10+2\times 5 = 20$$

先計算  $2\times 5$  的地方, 計算完的結果和另個 10 加總起來等於 20 等號按鍵處理 —

等號按鍵設定

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

等號按鍵處理

將各加減乘除完之後的運算結果做個整合顯現出來.

### 3.2 計算機程式期末心得.

心得: 從一開始分組, 選出組長, 建立一個合作的倉儲, 讓大家能把自己完成的工作能推到一個地方在整合推上去, 然後組長分工下去, 大家完成自己的工作, 不會的就互相討論, 沒有分工合作是會做比較慢的, 有大家的努力才能如期地完成

## 第四章 小數點和變號按鍵處理

### 4.1 小數點

當按下小數點的按鍵之後, 以 `pointClicked()` 方法處理, 並且直接在 `display` 字串中加上 “.” 字串

程式碼

```
if self.point : self.display.setText(self.display.text() + '.') self.point = False self.wait = False
def changeSignClicked(self): 變號按鍵 —
```

按下變號按鍵後, 由 `changeSignClicked()` 處理, 如果顯示幕上顯示為正值, 就在 `display` 字串最前面, 加上 “-” 字串

如果顯示幕上顯示負值, 就會移除 `display` 上字串最前面的 “-” 字元

程式碼

```
text = self.display.text() value = float(text)
```

```
    if value > 0.0:
```

```
        text = "-" + text
```

```
    elif value < 0.0:
```

```
        text = text[1:]
```

```
    self.display.setText(text)
```

```
def backspaceClicked(self):
```

### 4.2 心得

這次協同的目的為六個人合力編輯出一套計算機程式, 我在這方面學得比較不佳, 而且因為家裡有事所已請了很多次假, 很可惜都沒聽到課, 但在同學有耐心的指導下有追上進度的趨勢, 雖然蠻後面才把自己該完成的部分搞定, 但也總算是大

功告成了，也深深感受到合作的重要性，很多都是大家一起討論，讓我覺得這次的合作很愉快。

## 第五章 退格按鍵處理以及清除按鍵處理

### 5.1 全部清除按鍵

全部清除按鍵應用

\* 按下計算機上的 `clearall` 鍵後, 所有的運算重置設為 0

全部清除按鍵處理

\* 以 `clear()` 方法處理, 進入函式後, 將現有的運算數重置為 0

\* 離開 `clear()` 前, 將 `waitingForOperand` 起始設為 `True`, 表示等待新運算數中

\* 程式碼

```
def clear(self):  
    if self.waiting:  
        return  
    self.display.setText("")  
    self.waiting = True
```

### 5.2 退格按鍵

退格按鍵應用

\* 按下計算機上的 `backspace` 鍵後, 保留除了最後一個字元的字串。

退格按鍵處理

\* 由 `backspaceClicked()` 處理, 這時可以利用 Python 字串數列中的 `[:-1]`, 保留除了最後一個字元的字串

\* 離開 `backspaceClicked()` 前, 將顯示幕中原有字串的 `[:-1]` 字串, 顯示在 `display` 上

\* 若退格後 display 上為空字串, 則顯示 0, 並且將 waitingForOperand 起始設為 True, 表示等待新運算數中

\* 程式碼

```
def backspaceClicked(self):
    if self.wait:
        return

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

    self.display.setText(text)
```

## 5.3 清除按鍵

清除按鍵應用

\* 按下計算機上的 clear 鍵後, 把加、減、乘、除後的數值重置

清除按鍵處理

\* 以 clear( ) 方法處理, 進入函式後, 將現有的運算數重置為 0

\* 離開 clear( ) 前, 將 waitingForOperand 起始設為 True, 表示等待新運算數中

\* 程式碼

```
def clearAll(self):
    self.sumSoFar = 0.0
    self.factorSoFar = 0.0
    self.pendingAdditiveOperator = ""
    self.pendingMultiplicativeOperator = ""
    self.display.setText('0')
    self.waitingForOperand = True
```

## 5.4 心得

這是一個計算機程式，六個人一起努力的結果，不管結果重不重要，都希望大家分工合作，一點一滴的貢獻，雖然過程中很多問題，但大家都討論很熱烈，這是一個很難得的機會，接受別人的意見，在加上自己的想法，做出最好的決定。

## 第六章 記憶體按鍵處理與直接運算 (運算方式)

記憶體按鍵處理與直接運算 (運算方式) 的概要

1. 記憶體按鍵的用法，使我們在做計算時，將我們計算的數值儲存在記憶體中，方便我們更快速運算出所需要的數值。
2. 直接運算用法，利用數學上特殊計算法，來運算，例如：開根號、平方、倒數。
3. 計算方式，將我們知道的數學運算邏輯，告訴電腦，讓電腦替我們做運算而符合我們的邏輯運算

### 6.1 記憶體按鍵處理

記憶體按鍵處理的內容

#### 1. 邏輯概念：

`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

#### 2. 設定相對應的按鍵：

`self.clearMemoryButton.clicked.connect(self.clearMemory)` - “MC”鍵

`self.readMemoryButton.clicked.connect(self.readMemory)` - “MR”鍵

```
self.setMemoryButton.clicked.connect(self.setMemory) - "MS"鍵
```

```
self.addToMemoryButton.clicked.connect(self.addToMemory) - "M+"鍵
```

### 3. 邏輯運用:

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

```
    self.sumInMemory = 0.0
```

```
    self.display.setText(str(self.sumInMemory))
```

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

```
    self.display.setText(str(self.sumInMemory))
```

```
    self.wait = 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 直接運算

直接運算的內容

### 1. 邏輯概念:

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 顯示幕上

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, 所以需要 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

2. 設定相對應的按鍵:

```
unaryOperator = [self.squareRootButton, self.powerButton, self.reciprocalButton]
for i in unaryOperator:
    i.clicked.connect(self.unaryOperatorClicked)
```

\*squareRootButton - ”Sqrt鍵(開根號)”

\*powerButton - ”x^2鍵(平方)”

\*reciprocalButton - ”1/x鍵(倒數)”

3. 邏輯運用:

```
def unaryOperatorClicked(self):
    button = self.sender()
    clickedOperator = button.text()
    operand = float(self.display.text())
    if clickedOperator == "Sqrt":
        if operand < 0.0:
            self.abortOperand()
            return

    result = math.sqrt(operand)
    elif clickedOperator == "X^2":
        result = math.pow(operand, 2.0)
    elif clickedOperator == "1/x":
```

```
if operand == 0.0:
    self.sabortOperand()
    return
```

```
result = 1.0 / operand
```

```
self.display.setText(str(result))
self.wait = True
```

\* 中斷運算用:

```
def abortOperation(self):
    self.clearAll()
    self.display.setText("####")
```

## 6.3 運算方式

運算方式的內容

1. 邏輯概念:

`calculate()` 方法中的運算, 以 `rightOperand` 為右運算數執行加或減運算時, 左運算數為 `sumSoFar` 執行乘或除運算時, 左運算數為 `factorSoFar` 若運算過程出現除以 0 時, 將會回傳 `False`

2. 邏輯運用:

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

## 6.4 心得

內容:

這個計算機程式，是一組六個人共同製作，這考驗著六個人的默契，以及組員之間遇到問題如何解決，不過這是很難得經驗，合作本來就是一件不容易的事，常常會有意見相左的時候，不過這就學習的一部分，發現別人的優點加以學習，並修正自己錯誤的觀念，這樣的合作可以碰撞出很棒的火花。

## 第七章 參考文獻