National Yang Ming Chiao Tung University, Department of Electrical Engineering

嵌入式系統設計
Embedded System Design

Lab 3: Table View, Data Source and Delegate

蕭安紘助教製作

一. 實驗目的

了解 table view 的使用方法及原理，並試著使用 table view 呈現資料、增加與刪除資料，完成簡單的記帳程式。

二. 實驗需求

環境：macOS 13.4 或以上
IDE：Xcode 14.0 或以上
語言：Swift 5

三. Array

1. Array 的表示方法

Array<Type> 或 [Type] 表示

```
var stringArray1: Array<String>
var stringArray2: [String]
```

- stringArray1 和 stringArray2 的 type 都一樣，常用的表示法為 [Type]

2. 創建 Array

(1) 空陣列

```
stringArray = [String]()
stringArray = Array<String>()
```

(2) 直接給定Array的值

```
stringArray = ["A","B","C","D"]
```

(3) 使用Range Operators給定Array的值

```
var intArray = Array(1...5) //[1, 2, 3, 4, 5]
var intArray2 = Array(1..<5) //[1, 2, 3, 4]
```

(4) 使用重複的 element 創建 array

```
var stringArray = Array(repeating: "A", count: 3) //["A", "A", "A"]
```

3. Array 的存取與修改

(1) 取值

```
var stringArray = ["A","B","C"]
stringArray[1]      // "B"
stringArray.first   // "A"
stringArray.last    // "C"
```

(2) 長度

```
var stringArray = ["A","B","C"]
stringArray.count   // "3"
```

(3) 檢查是否為空

```
stringArray.count == 0
stringArray.isEmpty
```

- 兩個 statement 等價

(4) 修改 element

```
var stringArray = ["A","B","C","D"]
stringArray[1] = "b"              //["A", "b", "C", "D"]
stringArray[1...2] = ["1","2"]  //["A", "1", "2", "D"]
```

- 可以使用 range operator 來一次修改多個項目

(5) 新增加 element

```
var stringArray = ["A"]
stringArray.append("B")              //["A", "B"]
stringArray = stringArray + ["C"]    //["A", "B", "C"]
stringArray += ["D","E"]             //["A", "Z", "B", "C", "D", "E"]
```

- 注意！ 用 + 或 += ，後面接的是 Array 不是單一 Element

(6) 插入 element

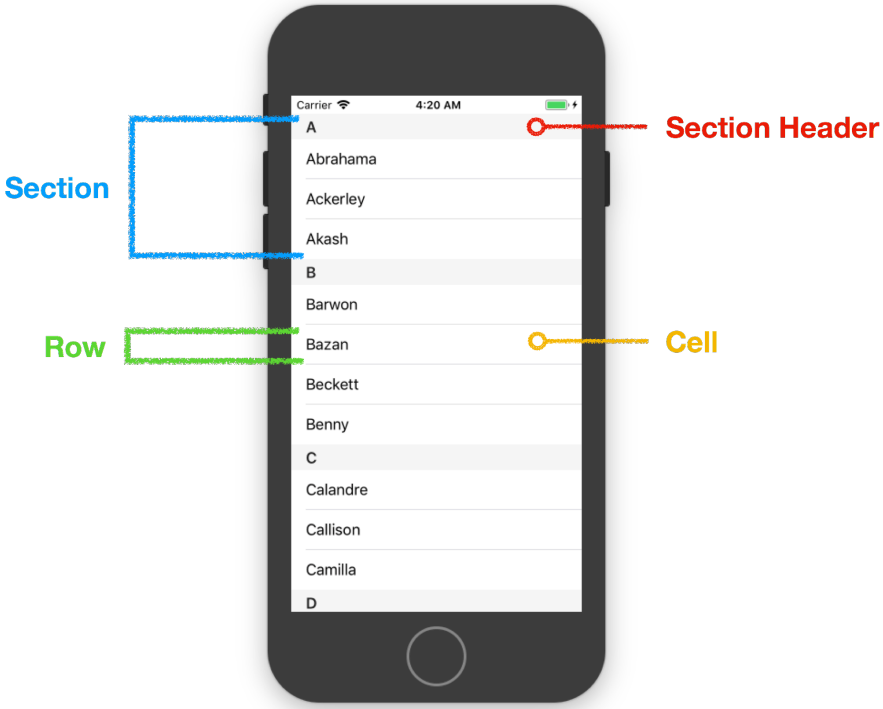
```
var stringArray = ["A","E"]
stringArray.insert("B", at: 1)                    //["A","B","E"]
stringArray.insert(contentsOf: ["C","D"], at: 2)//["A","B","C","D","E"]
```

- index 為插入的項目最後會在的位置

(7) 移除 element

```
var stringArray = ["A", "B", "C", "D", "E"]
stringArray.removeFirst()   //["B", "C", "D", "E"]
stringArray.removeLast()    //["B", "C", "D"]
stringArray.remove(at: 1)   //["B", "D"]
stringArray.removeAll()     //[]
```

四. Table View



我們可以使用 table view 來顯示列表型的資料，並且利用 UITableView 內建的一些 功能來讓使用者與資料做互動：新增、刪減、修改

1. Protocol

Protocol-oriented programming 為 swift 語言的其中一個核心設計，protocol 基本的概念就是提供一個不同 Type 等定義之間的一個協定。協定內定義了要符合這個協定必須要有的 variable 或是 method。只要 Type 符合特定的協定，就可以用來做特定的事，不需要像 class inheritance 一樣，一定要繼承同樣的 parent class ，才能用在相同的地方。 以實際的例子來說，智慧型手機、筆電、印表機、遊戲機，只要符合Wi-Fi傳輸的 protocol (802.11x)，就可以無線的互相傳遞資料，只要 protocol 有定義的事情都可以做到，就可以互相溝通。但我們不會說智慧型手機是筆電的一種。

當我們在定義要用的 variable 時，除了可以用他的 type 來定義之外，也可以用 protocol 來定義它。例如說，假設我們想用一個變數來表示智慧型手機目前連上網路的Wi-Fi 裝置，我們就可以用「符合 "網路提供protocol" 的 type 」來定義它，這樣一來，這個變數除了可以儲存 **Wi-Fi基地台**，也可以儲存開熱點分享網路的**智慧型手機**，或是任何其他可以提供網路的裝置。

2. Protocol Extension

如果有一個已經被定義好的型別，而我們無法更改它(例如官方提供的 Type)，但又想要讓他符合某個 protocol，我們可以使用 protocol extension 來額外提供它可以符合 protocol 的 variable 或 method 定義。舉例來說，桌上型電腦的主機板原本沒有Wi-Fi的功能，這時候我們可以另外用一個外接Wi-Fi網卡(Extension) 來擴充他的功能，讓他符合 Wi-Fi 傳輸 protocol 的定義，使他也可以跟其他 Wi-Fi 裝置溝通。

3. UITableView 定義的 Protocol

為了將資料與介面分開，而介面又得將使用者的互動傳遞給真正決定程式要怎麼做反應的物件，因此 UITableView 定義了常用的兩種 protocol：data source, delegate，也同時是 UITableView 中最重要的兩個 protocol

- Data source：
  View 要顯示資料時，從符合其定義的 data source protocol 的物件中抓取需要呈現的資料。protocol 中的 requirement 通常是「要資料」的 function，function 會 return view 需要的值。

- UITableViewDataSource 常用的 method:

```
//回傳 table view 有多少個 section
func numberOfSections(in tableView: UITableView) -> Int

//回傳 table view 中每個 section 有多少個 row
public func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int

//針對每個位置(index path)回傳此位置的Cell要長什麼樣子
//可用indexPath.section與indexPath.row確定現在 table view要的是哪個位置的cell
public func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell

//回傳section的header的文字
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int)
-> String?

//回傳section的footer的文字
func tableView(_ tableView: UITableView, titleForFooterInSection section: Int)
-> String?

//回傳此位置(index path)的row可不可以被編輯
func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) ->
Bool

//row被編輯的過後的動作
func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath)
```

- Delegate：
  View 做了某個動作時(通常是跟使用者操作有關)，去 call 其在 delegate 定義好的 function ，告訴別的物件自己發生了什麼事，protocol 中的 requirement 通常是「發生什麼事」 的 call back function，通常不會有 return 值，而且是 optional 的 不一定要定義。

- UITableViewDelegate 常用的 Delegate：

```
//tableview某個位置的row被選了會call這個function
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)

//tableview某個位置的cell將要出現的時候會call這個function
func tableView(_ tableView: UITableView, willDisplay cell: UITableViewCell,
forRowAt indexPath: IndexPath)

//tableview某個位置的cell出現在螢幕上後會call這個 function
func tableView(_ tableView: UITableView, didEndDisplaying cell:
UITableViewCell, forRowAt indexPath: IndexPath)

//回傳某個位置的row的高度
func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath)
-> CGFloat

//回傳某個section header的高度
func tableView(_ tableView: UITableView, heightForHeaderInSection section:
Int) -> CGFloat

//回傳某個section footer的高度
func tableView(_ tableView: UITableView, heightForFooterInSection section:
Int) -> CGFloat
```

五. 補充指令

1. 鍵盤打開，收合

```
//離開 text field 的輸入 (鍵盤會收回)
textField.resignFirstResponder()

//開始 text field 的輸入 (會打開對應text field設定的鍵盤)
textField.becomeFirstResponder()
```

六. 實驗步驟

1. 建立 iOS 專案 - Bookkeeping

(1) 開啟Xcode，左上角menubar，選擇 File -> New -> Project

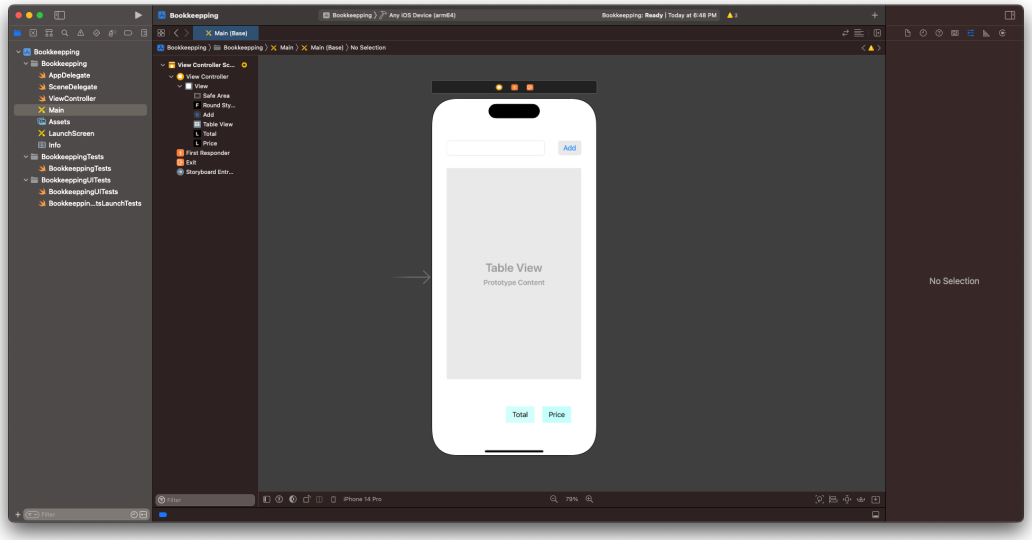
(2) 左方選擇 iOS 內的 Application ， 右方選擇 Single View Application，按 Next

(3) Product Name 輸入 **Bookkeeping**
Organization identifier 輸入 nctu.esd.”你的學號”
Language 選擇 Swift
按下 Next，存在桌面

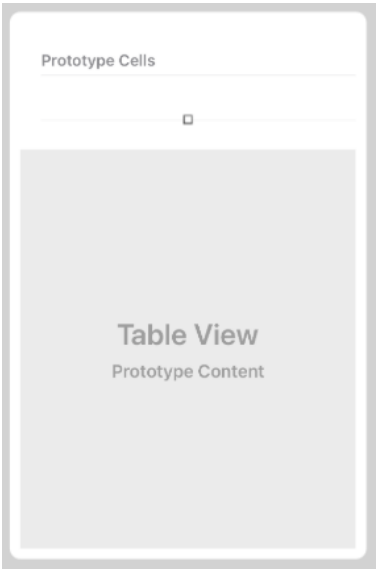
2. 完成Bookkeeping介面

(1) 從右下方的 Object library 拉出 Table View 、 textfield 、 button 各一，以及兩個 Label 到介面中

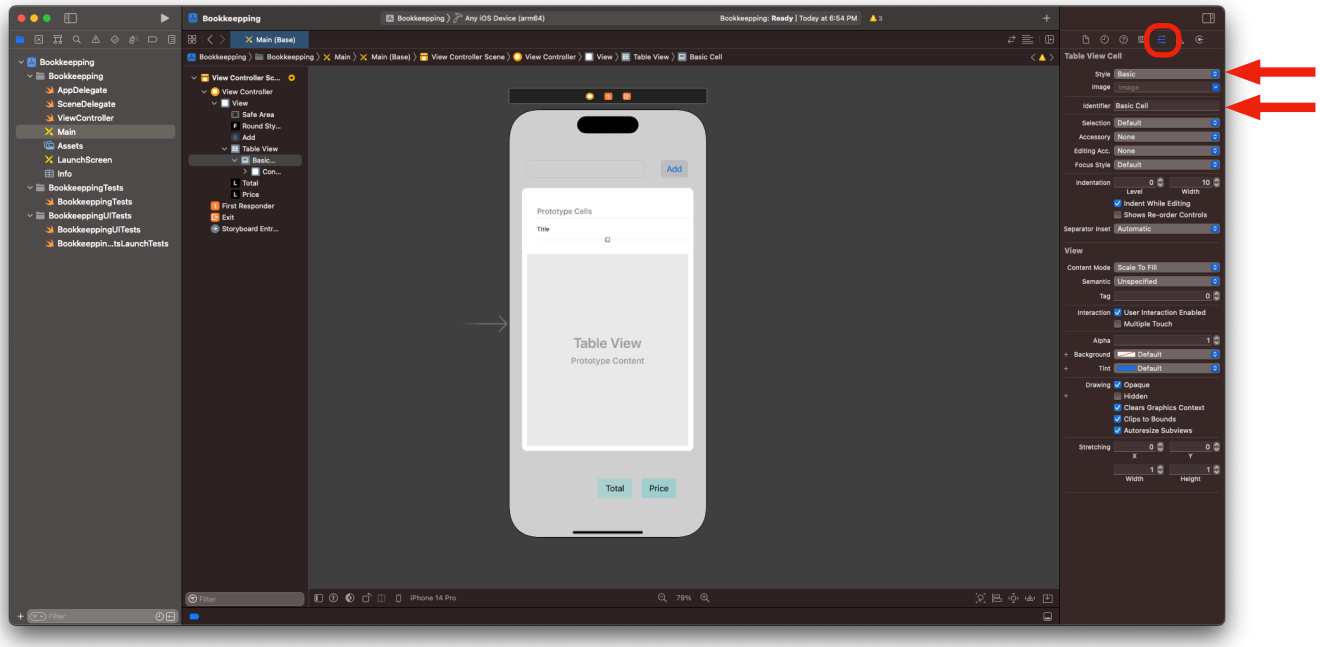
(2) 完成介面的的排列與文字修改 (不一定要照講義排)



(3) 從object library 中找到 table view cell，拉到畫面中的 table view 中



(4) 選取剛剛拉入的 table view cell ，從右方 Attribute Inspector 中把 table view cell 的 style 改成 basic，並且在 identifier 輸入 Basic Cell



(5) 利用右鍵拖移，將 text field 以及顯示總數值的 label 在 ViewController.swift 中建立 newCostField 以及 totalCostLabel 兩個 outlet。同樣，也建立名為tableView 的 table view 的 outlet。

```
@IBOutlet weak var tableView: UITableView!
@IBOutlet weak var totalCostLabel: UILabel!
@IBOutlet weak var newCostField: UITextField!
```

(6) 建立 add button 的 addData action

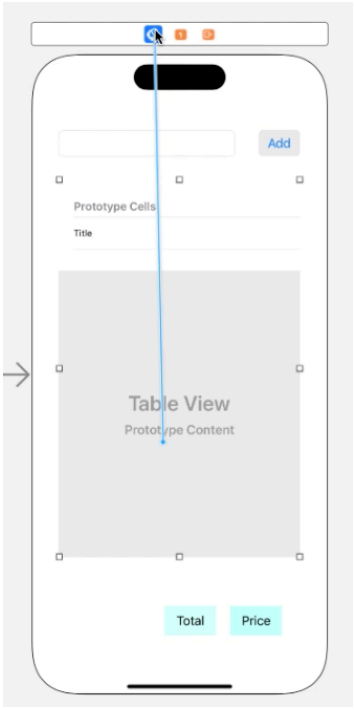
```
@IBAction func addData(_ sender: Any) {

}
```

3. 設定 table view 的 datasource 以及 delegate

(1) 由於我們需要利用 ViewController 來提供 table view 要顯示的 data，以及針對使用者對 table view 的互動做出反應，因此我們要把 ViewController 設定為 tableview 的 datasource 及 delegate。首先，先在 ViewController 中加入 UITableViewDataSource 及 UITableViewDelegate 這兩個 protocol，表示這個 class 會符合這兩個 protocol 的需求。

```
class ViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {
```

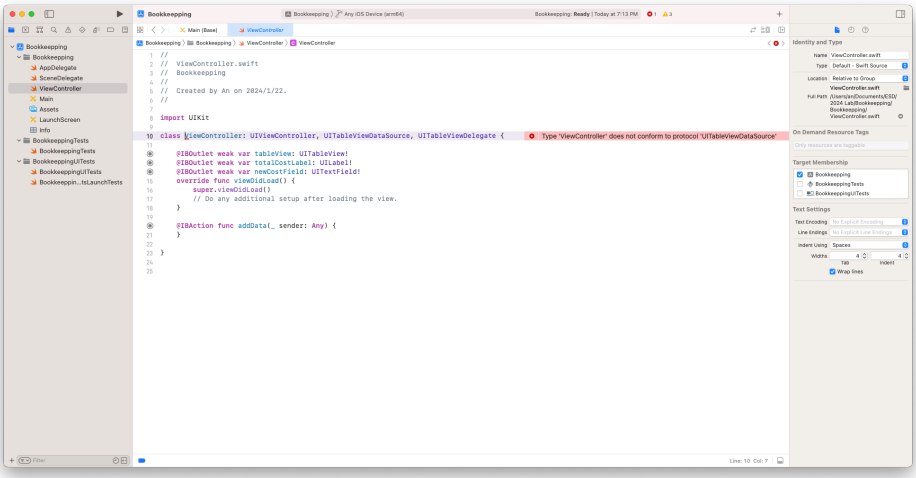
(2) 在 storyboard 中，用右鍵拖移 table view 至上方黃色的 view controller



點選 data source 與 delegate (需要再右鍵拖拉一次)

4. 完成必要的 datasource requirement，顯示資料

(1) 當我們在 class 的定義加上 UITableViewDataSource 後，可以發現 Xcode 出現 error。 因為 class 的內容並不符合這個 protocol 的 requirement。



(2) 由於 UITableViewDataSource 是一個提供資料的 protocol ，我們先在 Class 定義好要存資料的 array property，此實驗中我們使用 Double array

```
class ViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {

    var dataArray:[Double] = [123, 456, 789]
```

(3) 接著，加入 UITableViewDataSource 的其中一個 required 的 method，table view 的 row 的個數等於 array 的長度

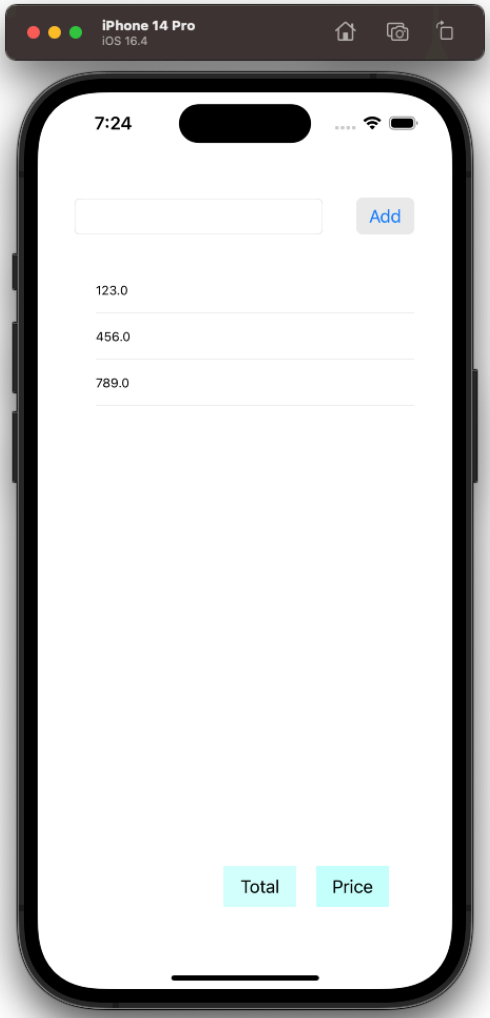
```
    func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        return dataArray.count
    }
```

(4) 加入另一個 UITableViewDataSource required 的 method，回傳 tableview 顯示到這個 row 的時候，要顯示的 cell 物件。 row 的位置可以從 function 的 indexPath得到 （ indexPath.section, indexPath.row)

```
    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

//從storyboard中的table view尋找有 identifier 為"Basic Cell"的 cell 範例
//且如果之前有相同identifier的Cell被宣告出來且沒有在用的話，重複使用，節省記憶體
        let cell = tableView.dequeueReusableCell(withIdentifier: "Basic
Cell", for: indexPath)

        //設定cell的內容
        cell.textLabel?.text = "\(dataArray[indexPath.row])"
        return cell
    }
```

※因為需要節省記憶體的關係 cell 物件是重複使用的(會拿沒有在顯示中的cell重複使用)，請務必每次都更新 cell 所有會變動的 property

(5) 當上述兩個 method 都 implement 完後，可以發現 Xcode 沒有在跳錯誤的警告。執行程式，可以看到 array 的數值已經可以被顯示在 table view 上。



(6) 到此，我們並沒有 implement 任何 UITableViewDelegate 的 method，因為其 protocol 定義的 method 都是 optional 的，也就是說，不管有沒有 implement 這些 function ，還是可以符合 protocol。

5. 新增與刪除 Table View 的資料

(1) 按下 add button 後，我們要**新增資料到 array** ，並且讓 table view 上顯示的資料跟著更新

```
    @IBAction func addData(_ sender: Any) {
        //檢查輸入匡有沒有文字，如果沒有，離開function
        guard let newCostString = newCostField.text, !
newCostString.isEmpty else { return  }

        //檢查輸入的文字可不可以轉成 Double，如果不能，離開function
        guard let newCost = Double(newCostString) else { return }

        //將新的數值加入array
        dataArray.append(newCost)

        //叫table view 重新讀取一次資料
        tableView.reloadData()

        //準備下次輸入，將輸入匡清空
        newCostField.text = ""

        //將鍵盤收起
        newCostField.resignFirstResponder()

    }
```

(2) 上述的方法是直接更新整個 table veiw 的資料，如果想要 table view 比較流暢的更新資料，也可以讓 table view 只更新特定範圍的 row ，前提是你必須知道資料更新的確切位置，我們這邊使用 insertRows

```
//創立指向新增資料所在的table位置的IndexPath物件
let refreshIndexPath = IndexPath.init(row: dataArray.count-1, section:
0)

//告訴table view要插入這些位置(array)的row，使用從上插入的動畫
tableView.insertRows(at:[refreshIndexPath], with: .top)
```

※這邊需要非常注意提供給 tableView 的新插入位置的數量，以及位置，如果算錯 index 或是數量不對，都會導致 crash

(3) **讓TableView可以刪除資料**，我們使用 table view 本身提供的 swipe action。首先，利用 delegate function 讓 tableview 知道我們要起用 edit row 這個功能，在class中加入

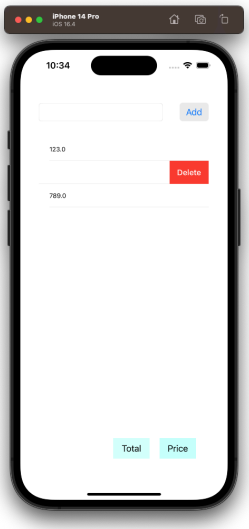
```
    func tableView(_ tableView: UITableView, canEditRowAt indexPath:
IndexPath) -> Bool {

        //我們讓 edit 的功能在每個位置的row都啟用。
        return true
    }
```

(4) 接著，implement 處理 edit row 動作的 delegate method，當使用者在 table view 做 edit row 動作時， table view 會呼叫這個 delegate method

```
    func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {

        switch editingStyle {

        case .delete: //如果是 commit delete 的動作

            dataArray.remove(at: indexPath.row ) //從 array 移除資料

            //告訴table view要刪掉這些位置(array)的資料，的row，使用往上刪除的
動畫
            tableView.deleteRows(at: [indexPath], with: .top)

        default: //其他edit的動作不做任何事
            break
        }
    }
```

※請自行填入???所表示資料在array中的位置

(5) 執行程式，測試刪除資料的功能 (從右往左滑)



6. 總額的更新

(1) 最後我們要讓最下方的總額可以隨著資料的變動而更新，我們先新增一個叫做 updateTotal 的 method

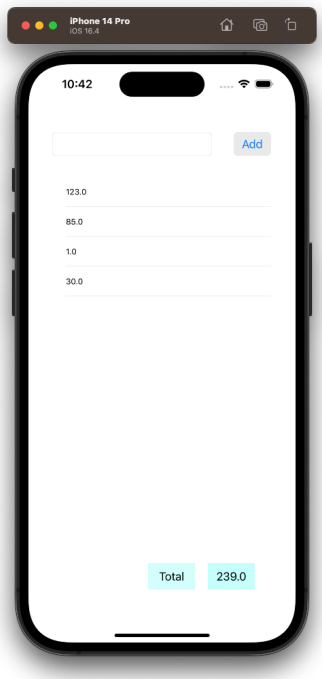
```
func updateTotal(){
    var total:Double = 0

    TODO:計算總額存到 total 變數

    TODO:更新總額到 totalCostLabel
}
```

※請自行完成函數的內容

(2) 在所有該更新總額的地方呼叫 updateTotal() 函數

※請自行完成

(3) 最後的結果如下，下方的Total會隨著資料而變動



(4) 我們會在下次 Lab 中實做更完整的 Bookkeeping 功能

七. Demo

1. 請 Demo 最後的執行結果，**可新增**、**刪除資料**，總額也會隨時隨著資料更新，排版不用跟講義一樣。