ex0-1

創客•自造者
工作坊 WORKSHOP

STEM 教育•寫程式•玩科技
Science · Technology · Engineering · Mathematics

ESP32×Python AIoT 大應用

用 ESP32 帶你體驗 AIoT 智慧聯網

閃爍 LED 燈／門鎖控制／門鎖遙控器／量測溫度值／手機溫度監控站／
防盜收藏盒／防盜收藏盒 - LINE 通知／自製時鐘／自製警報器／
火車誤點提醒器／雲端溫度紀錄儀／無線網路門鎖遙控器／網頁門鎖遙控器／
遊戲室年齡監控站／智慧門鎖／讀取按壓開關狀態／自製藍牙截圖、音量遙控器

旗標創客 · 自造者工作坊
FB 粉絲專頁
專人線上服務

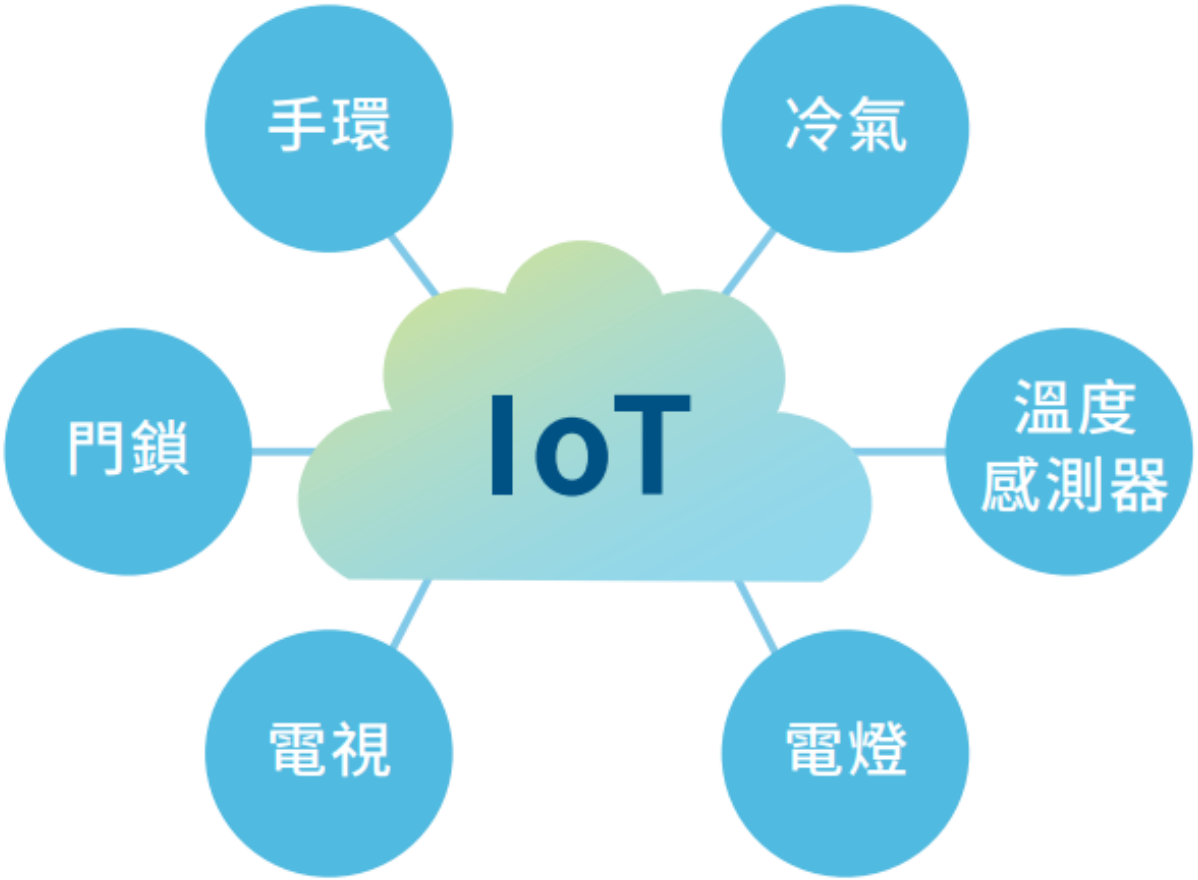
範例程式免費下載網址
https://www.flag.com.tw/download.asp?FM631A

旗標科技創客設計工程師

時間	主題	
08:00 – 08:10	ESP32 簡介	
08:10 – 08:45	安裝 Python 環境 & 點亮 LED 燈	
08:45 – 09:45	防盜監測站	1. 讀取霍爾感測器值
		2. IFTTT 發送 LINE 訊息
		3. 防盜收藏盒
09:45 – 11:00	火車誤點提醒器	1.使用七段顯示器自製時鐘
		2. 使用網路服務查詢火車資訊
		3. 火車誤點提醒器
11:00 – 12:00	智慧門鎖	1. 人臉辨識
		2. 藍牙傳輸
		3. 智慧門鎖

目錄

1-1 物聯網簡介



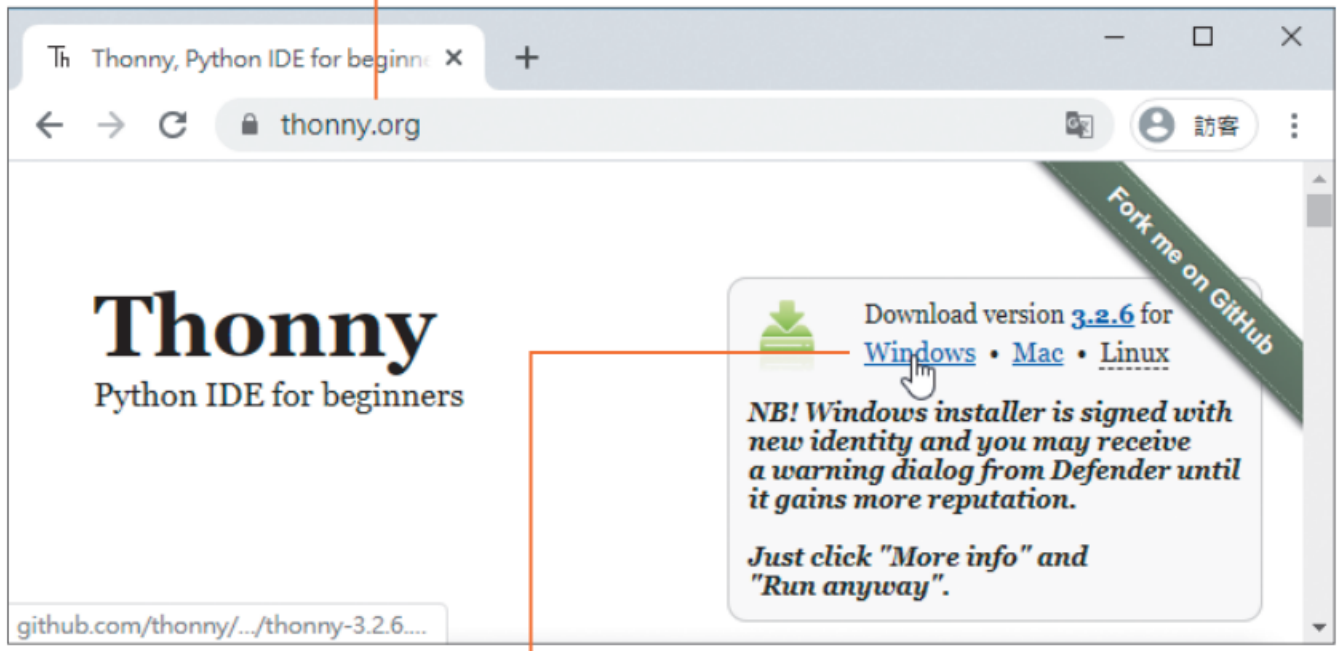
1-2 ESP32 簡介

目錄

2-1 安裝 Python 開發環境

下載與安裝 Thonny

1 連線 https://thonny.org



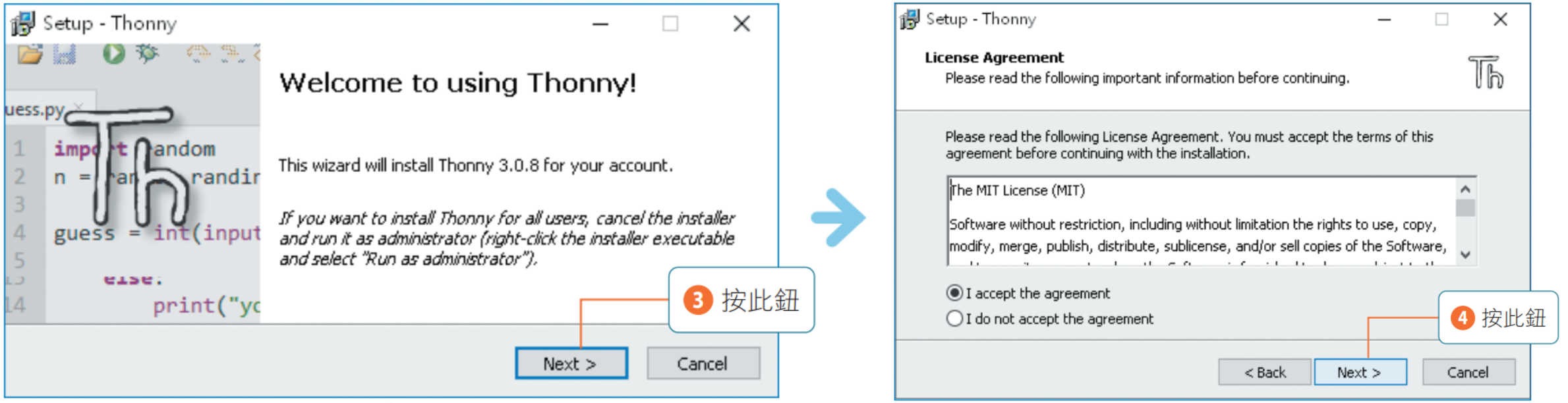
2 按此連結下載

⚠ 使用 Mac/Linux 系統的讀者請點選相對應的下載連結。

跳過安裝教學

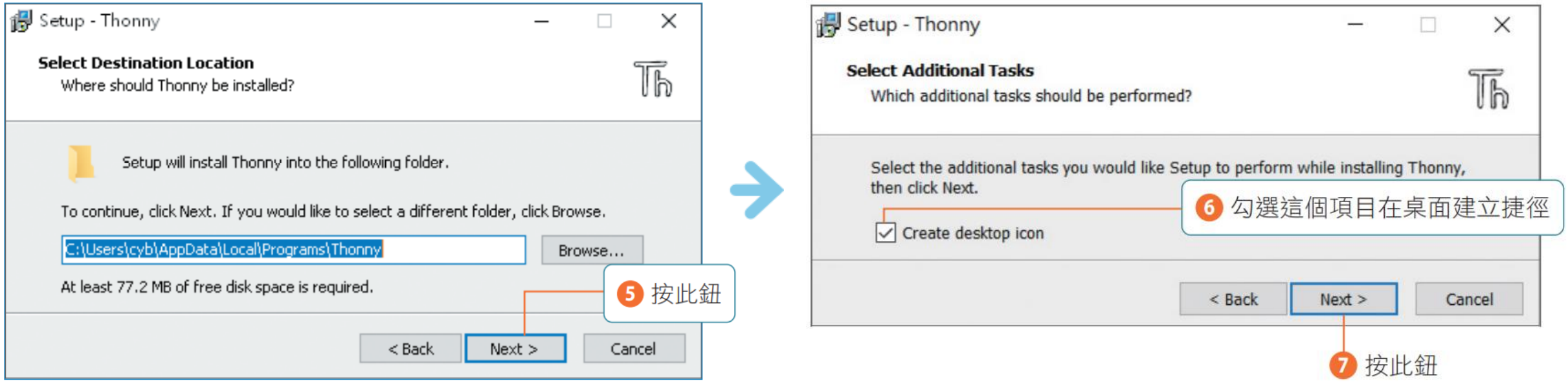
NEXT

2-1 安裝 Python 開發環境



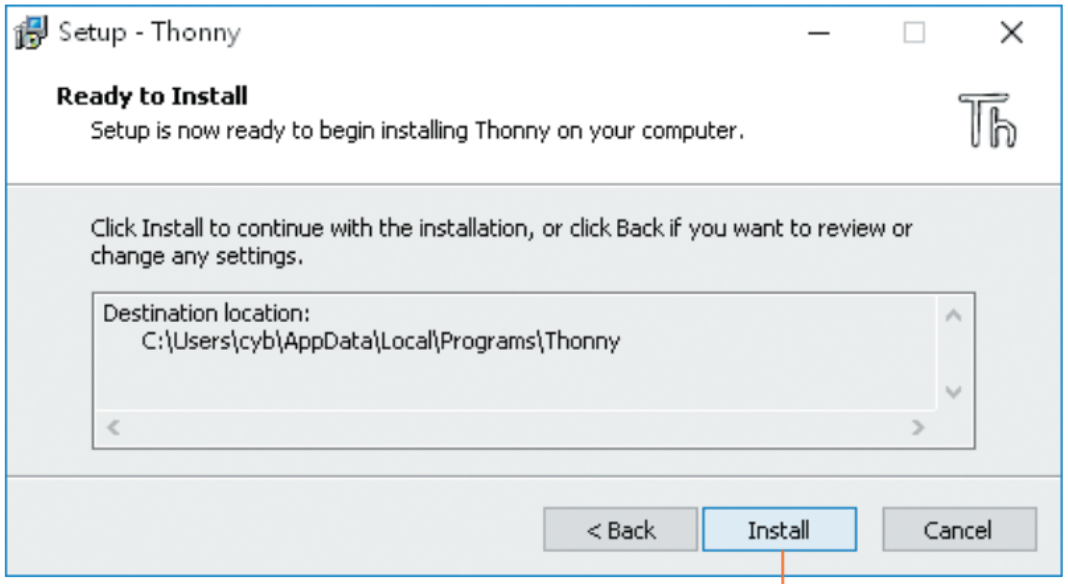
NEXT

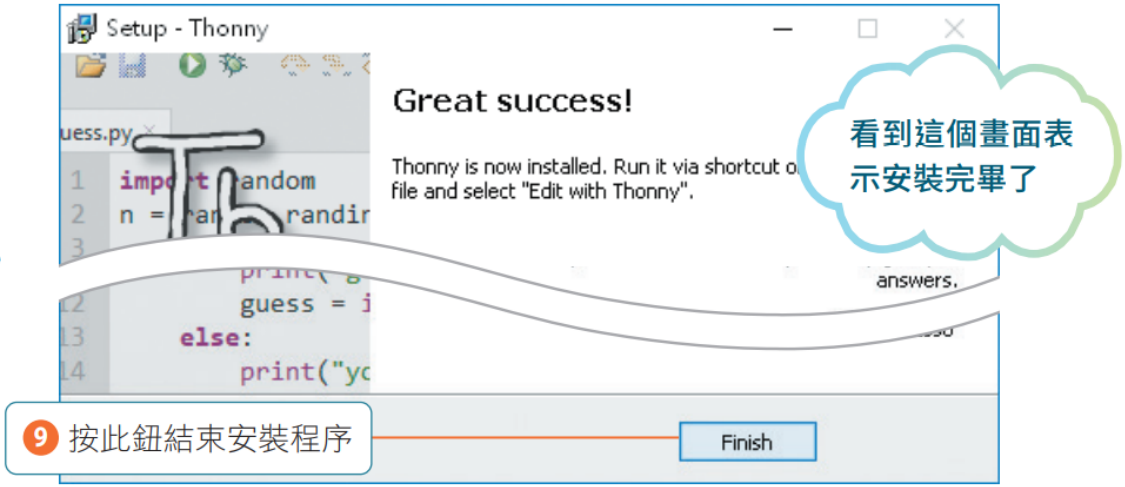
2-1 安裝 Python 開發環境



NEXT

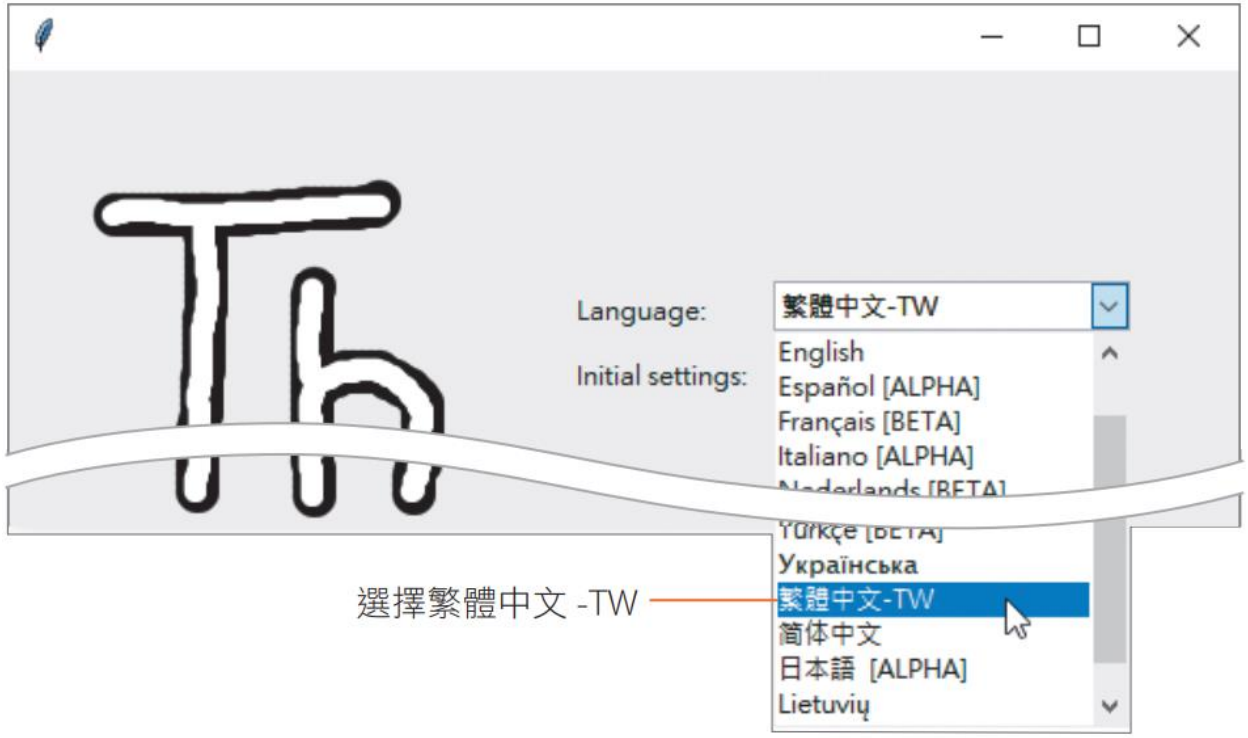
2-1 安裝 Python 開發環境

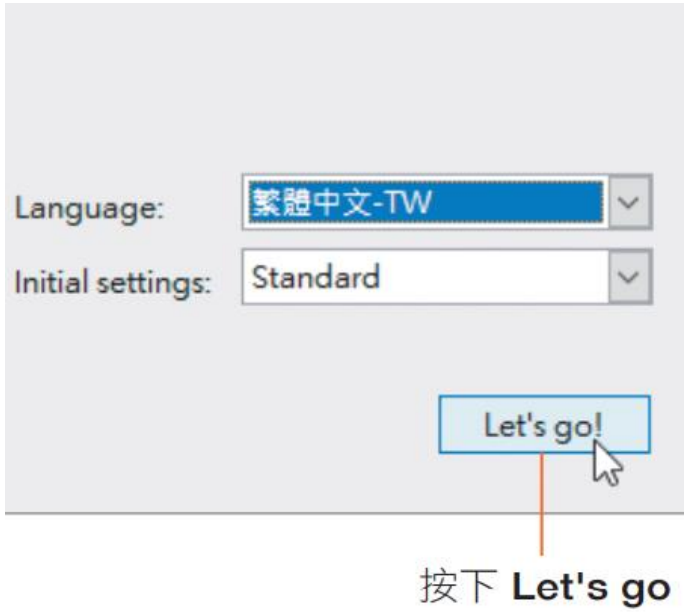




2-1 安裝 Python 開發環境

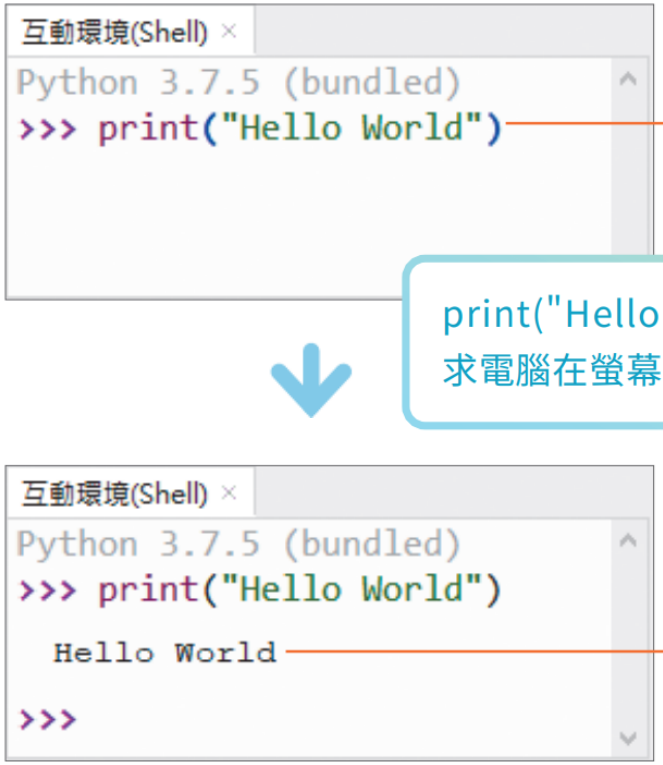
開始寫第一行程式





NEXT

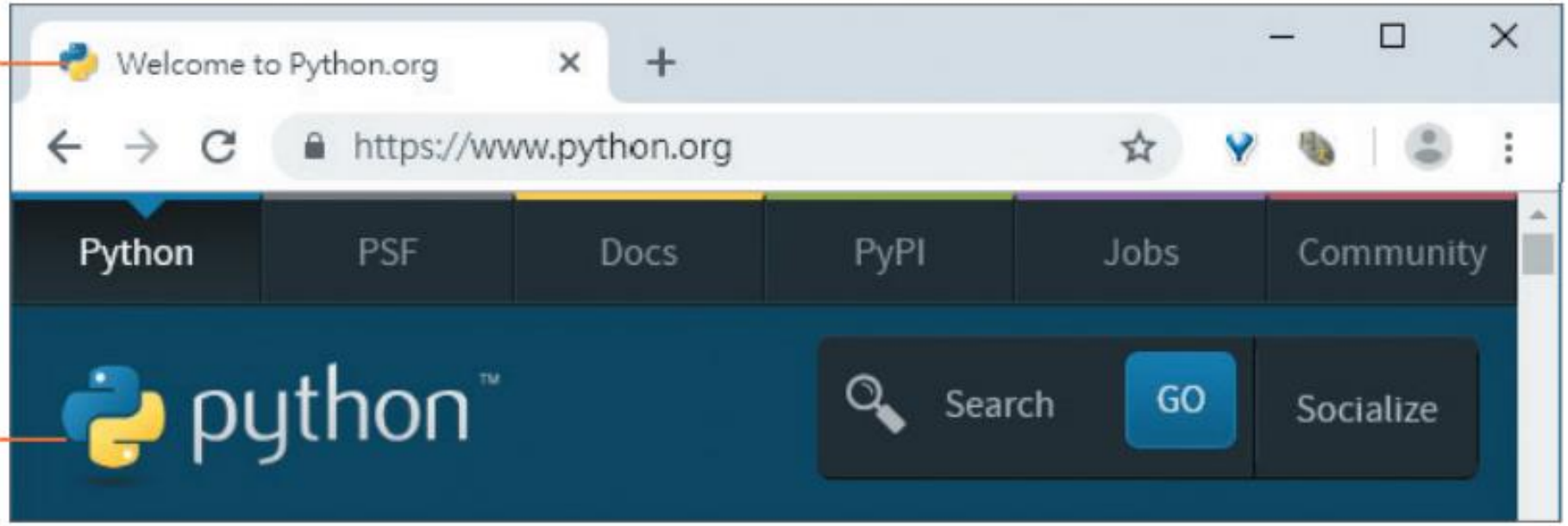
2-1 安裝 Python 開發環境



2-1 安裝 Python 開發環境

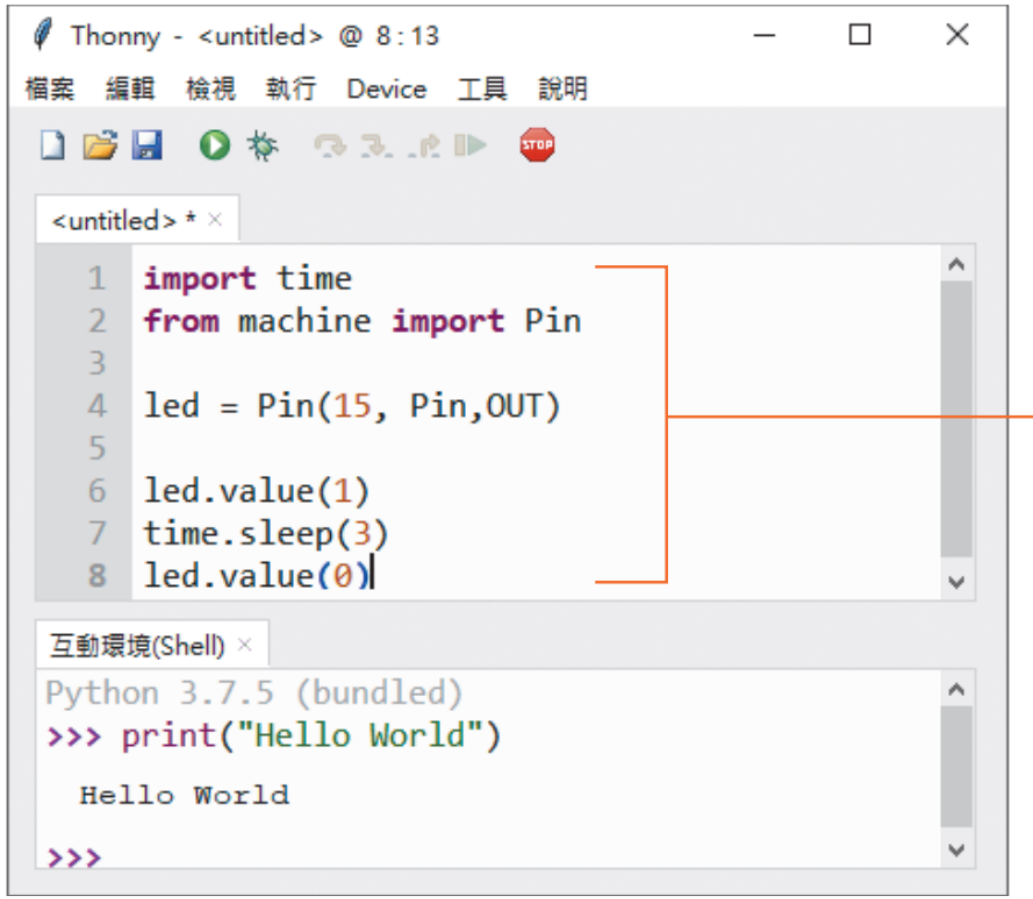
Python 程式語言

Python 的
蟒蛇標誌



2-1 安裝 Python 開發環境

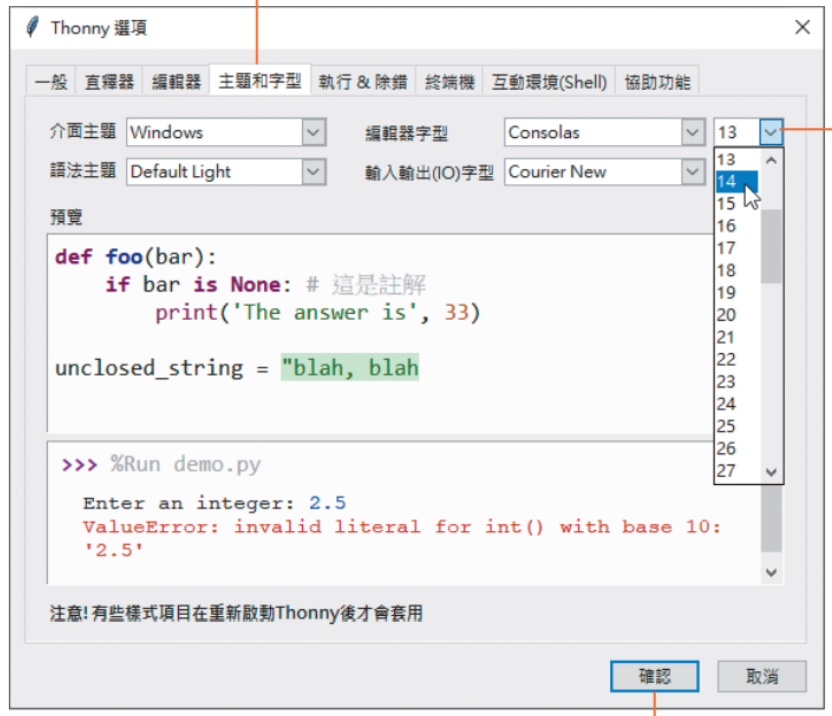
Thonny 開發環境基本操作



⚠ 本書後面章節若看到程式前面有 >>>, 便表示是在 **Shell** 窗格內執行與測試。

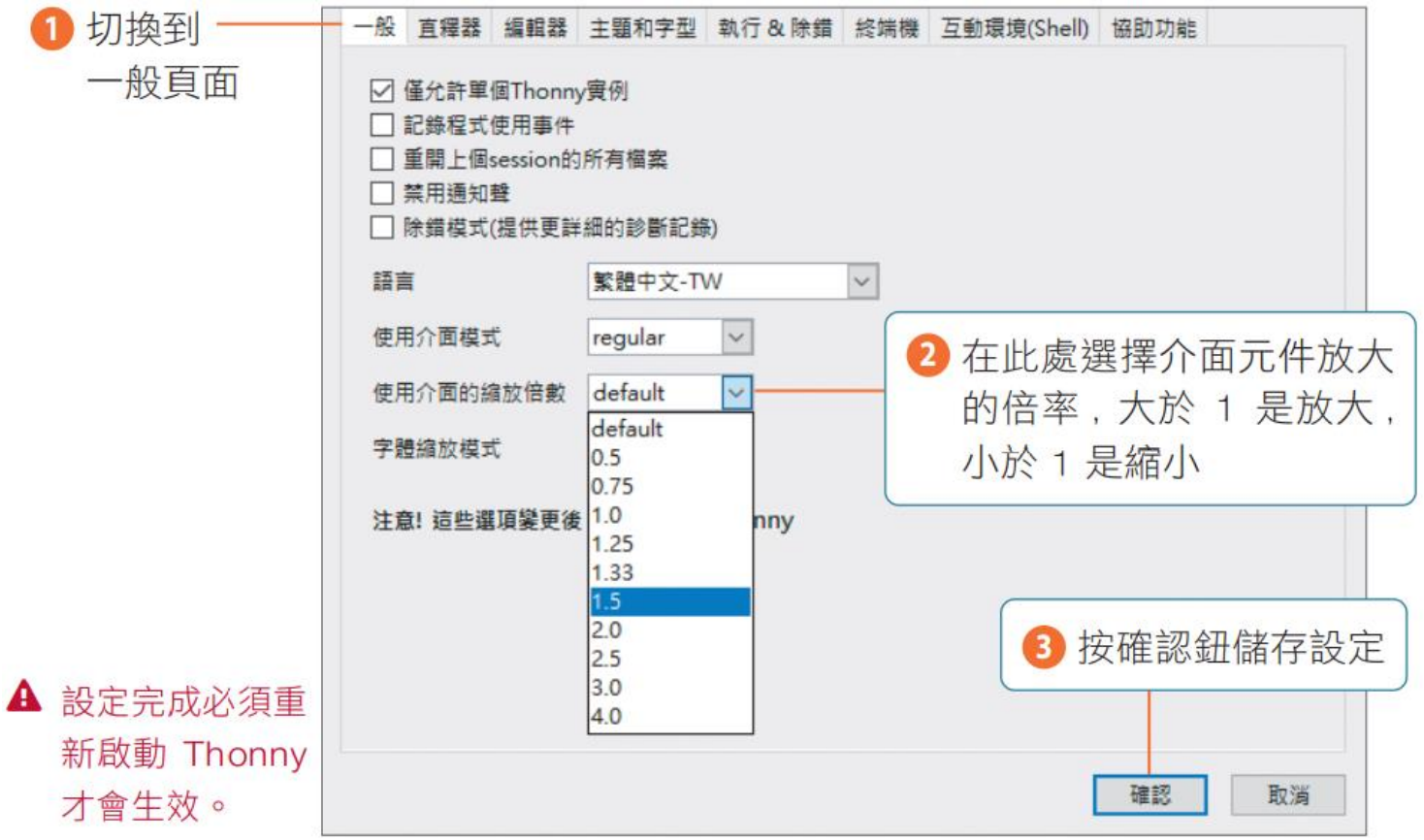
2-1 安裝 Python 開發環境

1 執行選單的『**工具 / 選項…**』命令，開啟設定視窗



NEXT

2-1 安裝 Python 開發環境

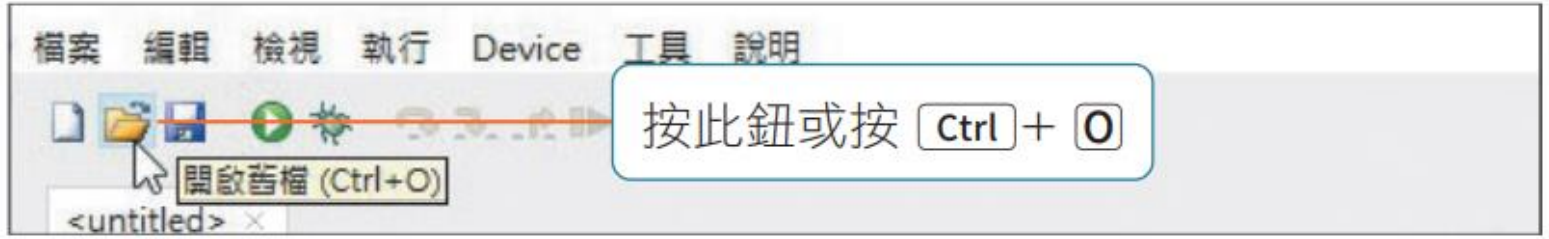


⚠ 設定完成必須重新啟動 Thonny 才會生效。

2-1 安裝 Python 開發環境

儲存檔案

開啟舊檔



執行&停止

2-2 Python 物件、資料型別、變數、匯入模組

物件

寫作文章	寫 Python 程式	說明
車子	car	car 物件
車子向前進	car.go()	car 物件的 go 方法

物件

- 英文一般寫句子時，會以名詞 + 動詞。Python 是以物件.方法來描述。

文章寫作	寫 Python 程式	
車子	car	car 物件
車子向前進	car.start()	car 物件的 start 方法

- 方法後面會加上括號()，有些方法需要加入額外的參數。
  - ✓ 例如：car.go(100)，車子加速到 100。
- 若方法有多個參數，以逗點分隔。
  - ✓ 例如：car.left(50, 30)，以 50 的速度，向左轉 30 度。

練習：字串物件

ex2-4

- 在互動模式中，輸入下列敘述：(>>> 指的是在互動模式中，執行單行敘述)

```
>>> "abc".upper()          ——— 使用字串物件 "abc" 的 upper() 方法，將字串轉成大寫
'abc'
>>> "abc".find('b')        ——— find() 方法尋找 'b' 的位置 (從 0 開始)
1
>>> "abc".replace('b', 'z') ——— replace() 方法將所有的 'b' 取代成 'z'
'azc'
```

- 不同的物件會有不同的方法。例如：字串物件與整數物件。

資料型別

ex2-5

- 除字串物件以雙引號或單引號來表示，寫程式常有整數與浮點數(小數) 物件，例如：111 與 11.1。

```
>>> 111 + 111 ———— 整數物件相加
222
>>> "111" + "111" ———— 字串物件串聯
'111111'
```

- 上述 + 的運算，因物件的資料不同而產生不同的結果。物件的種類，程式語言稱之為『物件型態』或『資料型態』(Data Type)。

練習：要分清楚資料型別

ex2-6

- 兩個資料型別若不同，可能會導致程式錯誤。

```
>>> 111 + "111" —— 不同型別的資料相加發生錯誤
Traceback (most recent call last):

  File "<ipython-input-6-4832c22160be>", line 1, in
<module>
    111 + "111"

TypeError: unsupported operand type(s) for +: 'int'
and 'str'
```

型別轉換

ex2-7

- 兩個資料型別若要運算，可以使用型別轉換。

```
>>> str(111) + "111"   ——— str() 可轉換物件為字串型別
'111111'
>>> 111 + int("111")   ——— int() 可轉換物件為整數型別
222
```

整數與浮點數

ex2-8

- 整數與浮點數的數學運算
  - ✓ + (加)、- (減)、* (乘)、/ (取商)、// (取商，整數)、% (取餘數)、** (指數)。
  - ✓ Python 允許整數與浮點數直接運算，執行下列程式：

加法 (+)

```
>>> 10 + 5.5
15.5
```

減法 (-)

```
>>> 10 - 5.5
4.5
```

ex2-8

乘法 (*)

```
>>> 10 * 5.5
55.0
```

除法取商 (/)

```
>>> 10 / 5.5
1.8181818181818181
```

除法取整數商 (//)

```
>>> 10 // 5.5
1.0
```

ex2-8

取餘數 (%)

```
>>> 10 % 5.5
4.5
```

指數 (**)

```
>>> 4 ** 0.5, 8 ** (1/3)
(2.0, 2.0)
```

TIP

1. 整數與浮點數做運算，結果一定為浮點數。
2. 只有整數與整數做除法，結果為浮點數。

常用的變數運算

ex2-9

- 把整數加上特定的值：

```
>>> x = 1
>>> x = x + 1
>>> x
2
```

- 常用的簡式：

簡式	意義	簡式	意義
x += 2	x = x + 2	x /= 2	x = x / 2
x -= 2	x = x – 2	x //= 2	x = x // 2
x *= 2	x = x * 2	x %= 2	x = x % 2

變數

ex2-10

- 『變數』(variable) 就像是掛在物件的名牌，幫物件取名之後，讓我們方便識別物件與操作，其語法為：

```
變數名稱 = 物件
```

- 例如：

```
>>> n1 = 123456789 ——— 將整數物件 123456789 指派給變數 n1
>>> n2 = 987654321 ——— 將整數物件 987654321 指派給變數 n2
>>> n1 + n2 ——————— 實際上是 123456789 + 987654321
1111111110
```

寫出可讀性高的程式 (1/2)

ex2-11

- 使用有意義的變數 (variable) 的名稱。

```
a = 3.1
b = 2.2
c = a * b * b
```

```
pi = 3.1
radius = 2.2
# 使用公式計算圓面積
circle_area = pi * radius * radius
```

寫出可讀性高的程式 (2/2)

- 將程式加上註解。
  - ✓ 註解可以幫助其他人了解這個程式。
- 較佳的註解：

```
# 註解 1：使用公式計算圓面積
```

- 不好閱讀的註解：

```
# 註解 2：將圓周率乘半徑乘半徑
```

內建函式

ex2-12

- 『函式』(function) 是一段預先寫好的程式，方便重複使用。而程式先將經常使用到的功能以函式的形式先寫好，稱為『內建函式』。
- 例如：print() 是最常用的顯示函數：

```
>>> print("abc") ——— 顯示字串物件
abc
>>> print("abc".upper()) ——— 顯示字串物件.方法的執行結果
ABC
>>> print(111 + 111) ——— 顯示整數物件運算的結果
222
```

匯入模組

ex2-13

- 內建函式不就越多越好？若內建函式無限制增加，會導致啟動速度越來越慢，執行時佔用的記憶體越來越多。
- 『模組』(module)，就是將同一類的函式打包成模組，預設不會啟用。需要時，再用匯入 (import) 的方式啟用。預先寫好的稱為『內建模組』。

```
>>> import time ——— 匯入時間相關的 time 模組
>>> time.sleep(3)  ——— 執行 time 模組的 sleep() 函式，暫停 3 秒


>>> from time import sleep ——— 從 time 模組裡匯入 sleep() 函式
>>> sleep(5) ——— 執行 sleep() 函式，暫停 5 秒
```

練習：匯入模組

ex2-14

程式 暫停 3 秒後，印出 Hello World! 字串物件

```
#  暫停 3 秒後，印出 Hello World!
from time import sleep
sleep(3)
print("Hello World!")
```

指的是在程式編輯窗格中編輯與執行。

觀念整理

各種程式語言的語法邏輯都差不多，就像人類語言的文法。

1. 程式的每一個東西都是物件，有些物件有其特定的操作方法。
2. 基本物件有資料型別，型別不同，結果不同。甚至有時會錯誤。
3. 變數是物件的名牌而已，也方便程式設計師操作。
4. 使用有意義的變數名稱與註解。
5. 內建函式是經常用的函式，預先寫好的。
6. 適當的匯入模組，能精簡效能。

2-3 安裝與設定 ESP32 控制板

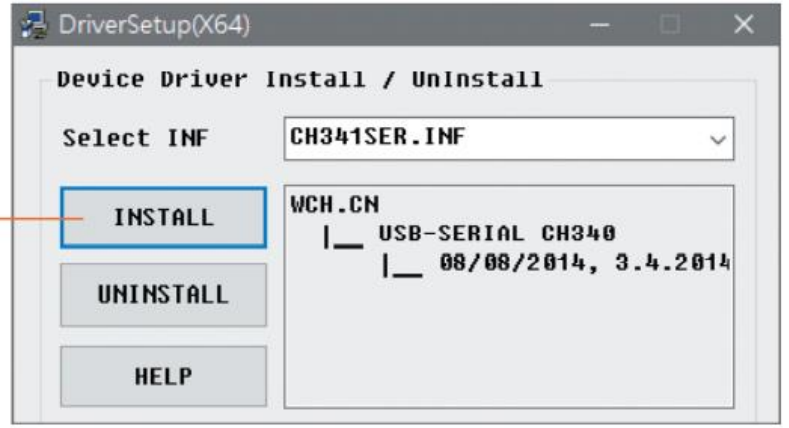
下載與安裝驅動程式

http://www.wch.cn/downloads/CH341SER_EXE.html

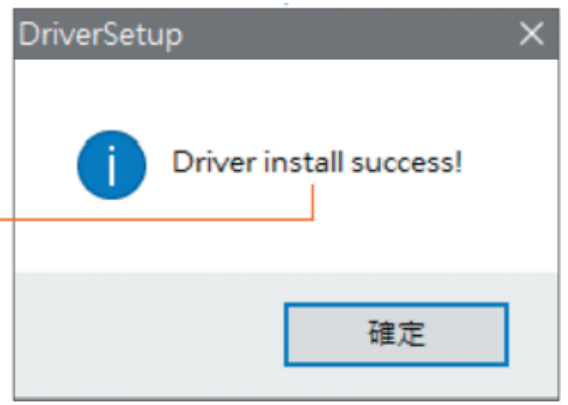
若您使用 Mac，系統已內建驅動程式，不用下載安裝。

2-3 安裝與設定 ESP32 控制板

1 請選是允許安裝



2 按此鈕進行安裝

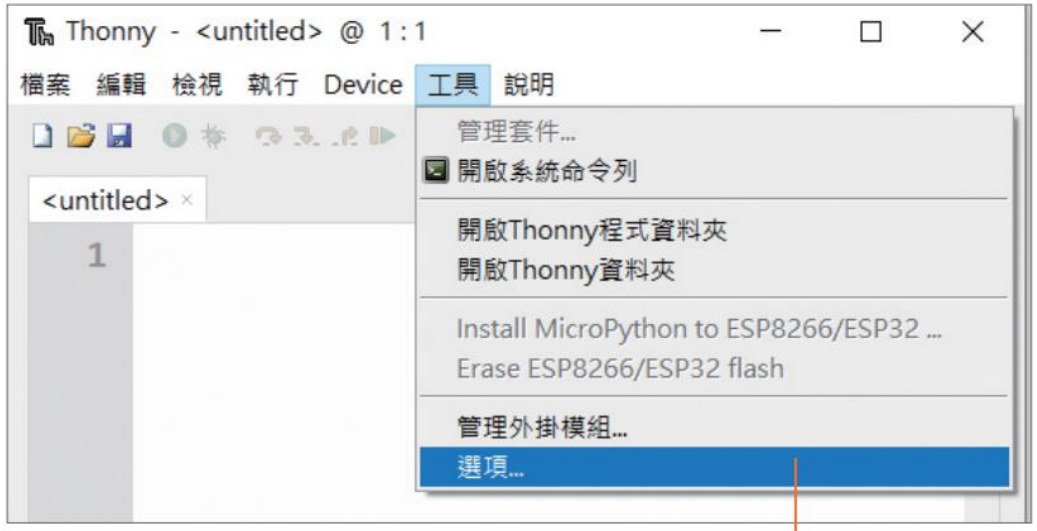


看到 success 便表示安裝成功了！

2-3 安裝與設定 ESP32 控制板

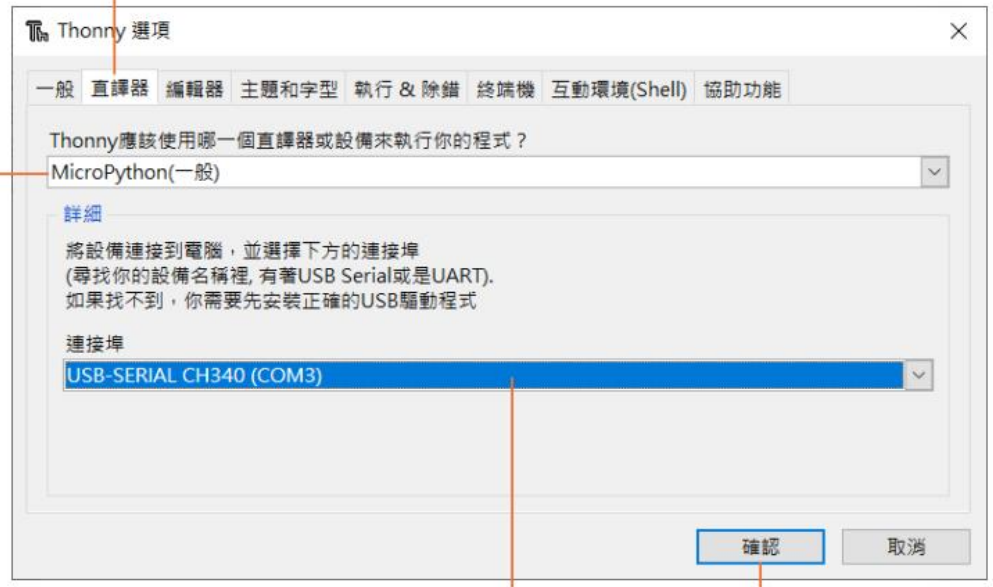
連接 ESP32

2-3 安裝與設定 ESP32 控制板



1 執行選單的『**工具 / 選項…**』命令，開啟設定視窗

2 切換到**直譯器**頁面



3 拉下選單選擇 **MicroPython(一般)**

4 拉下選單選一有 CH340 字樣的序列埠（Mac 上請選有 "/dev/cu.wchusbserial" 字樣的項目）

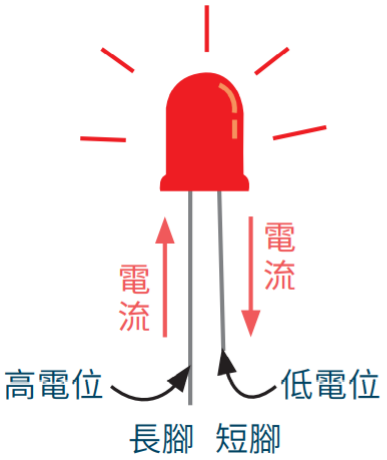
5 按**確認**鈕儲存設定

2-3 安裝與設定 ESP32 控制板

在**互動環境 (Shell)** 窗格看到 MicroPython 字樣便表示連線成功

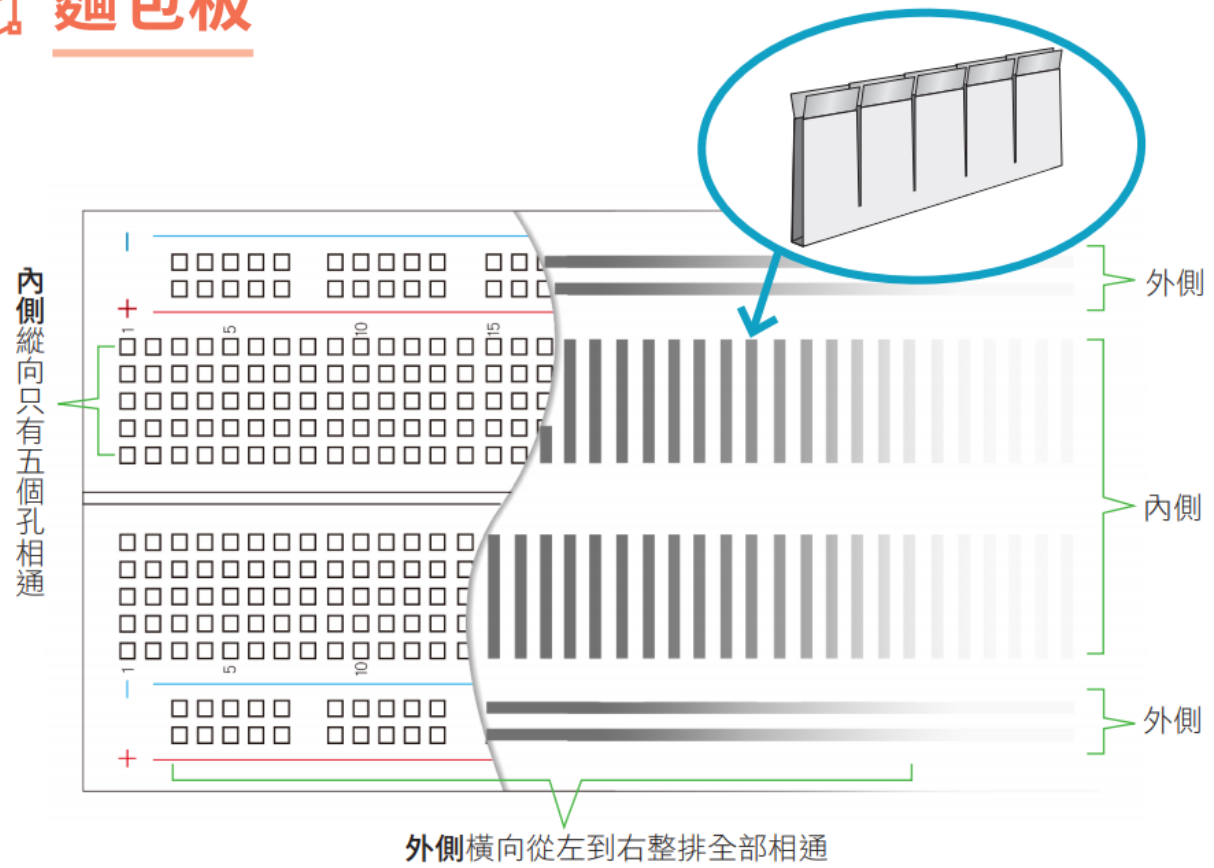
2-4 認識硬體

LED



本套件中的 LED 已內建在 ESP32 上。

麵包板

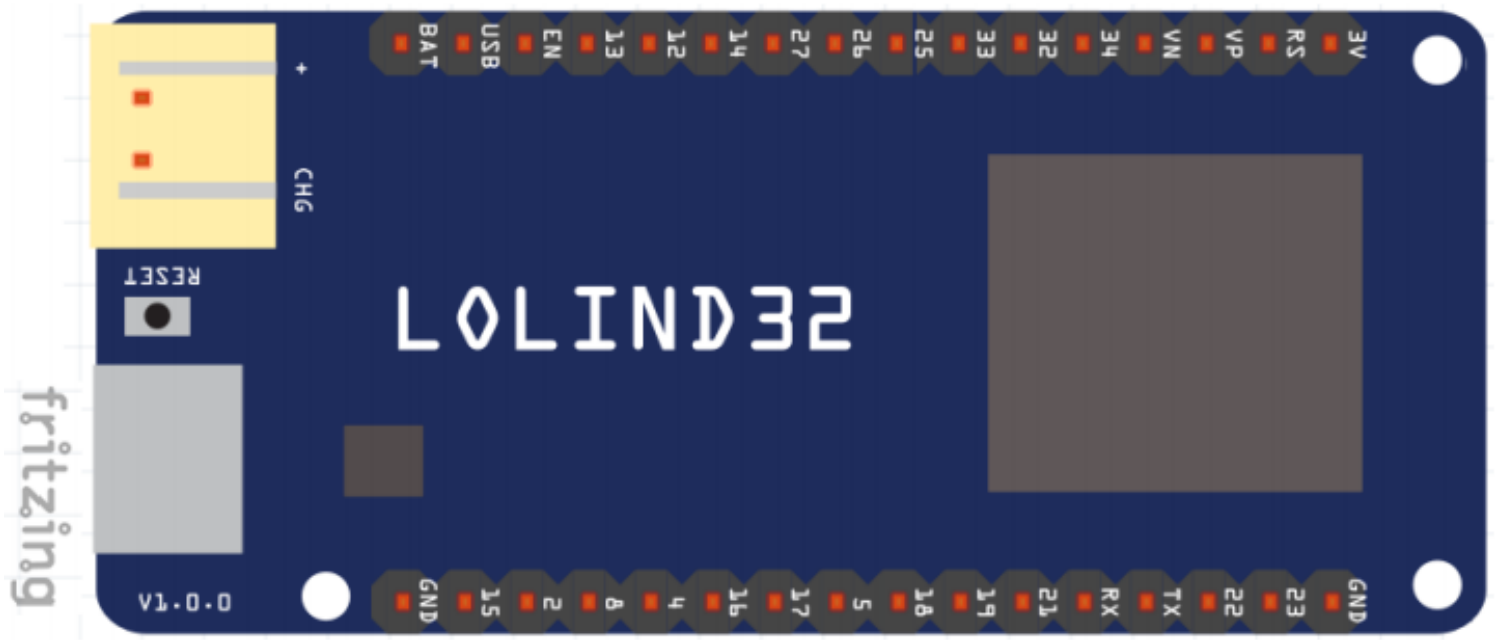


2-4 認識硬體

杜邦線與排針

不同顏色的杜邦線功能都相同，顏色只是方便區分。

2-5 ESP32 的 IO 腳位以及數位訊號輸出



LAB01 閃爍 LED 燈

實驗目的	熟悉 Thonny 開發環境的操作，並點亮 ESP32 上內建的藍色 LED 燈
材料	ESP32 控制版

線路圖

此實驗無須接線

NEXT

LAB01 閃爍 LED 燈

設計原理

```
>>>  from machine import Pin
```

```
>>>  led = Pin(5,Pin.OUT)
```

```
>>>  led.value(1) ← 高電位，熄滅 LED 燈
>>>  led.value(0) ← 低電位，點亮 LED 燈
```

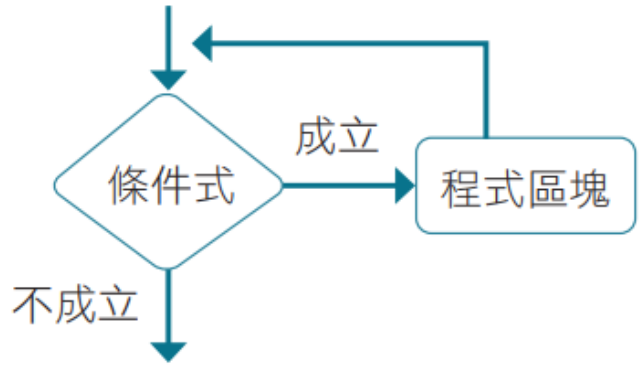
NEXT

LAB01 閃爍 LED 燈

while 迴圈

while 條件式：

　　程式區塊




```
>>>  while True:                # 一直重複執行
         led.value(1)           # 熄滅 LED 燈
         time.sleep(1)          # 暫停 1 秒
         led.value(0)           # 點亮 LED 燈
         time.sleep(1)          # 暫停 1 秒
```

NEXT

LAB01 閃爍 LED 燈

程式設計

```
#從 machine 模組匯入 Pin 物件
from machine import Pin
#匯入時間相關的time模組
import time

#建立 5 號腳位的 Pin 物件, 設定為腳位輸出, 命名為 led
led = Pin(5, Pin.OUT)

while True:
    led.value(1)     #熄滅 LED 燈
    time.sleep(0.5) #暫停 0.5 秒
    led.value(0)     #點亮 LED 燈
    time.sleep(0.5) #暫停 0.5 秒
```

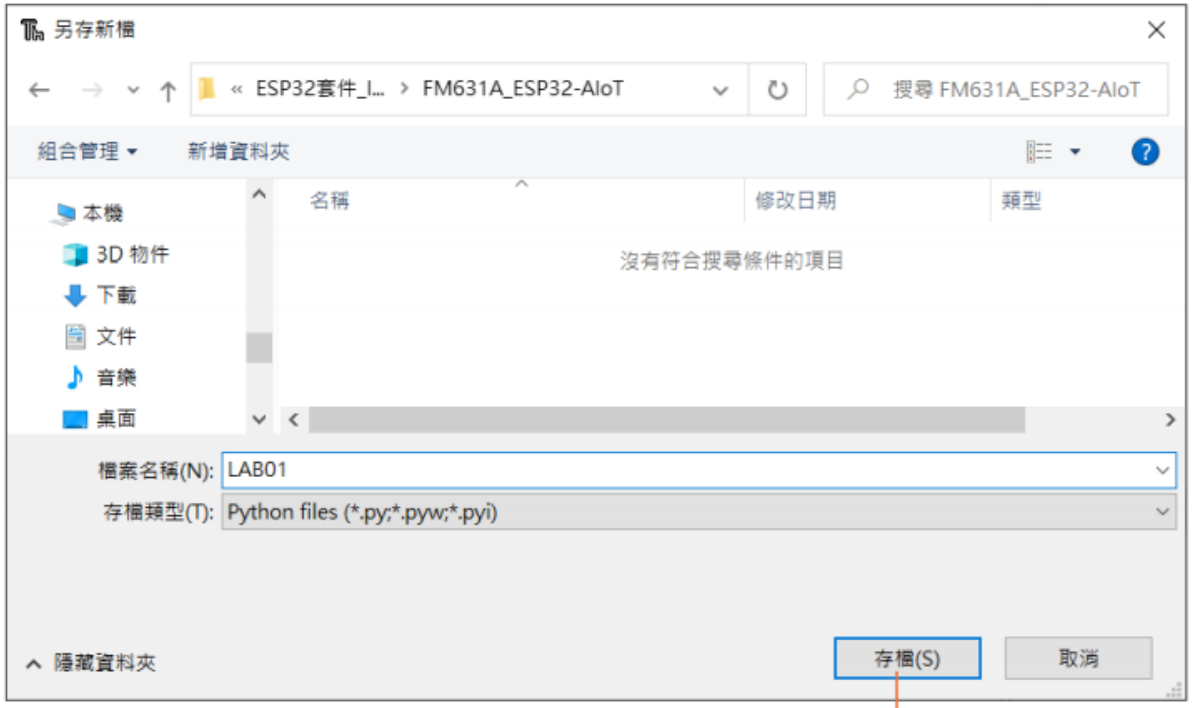
⚠ 程式裡面的 # 符號代表註解

NEXT

LAB01 閃爍 LED 燈

選擇本機

主程式需為： main.py



輸入檔名後按存檔鈕儲存

NEXT

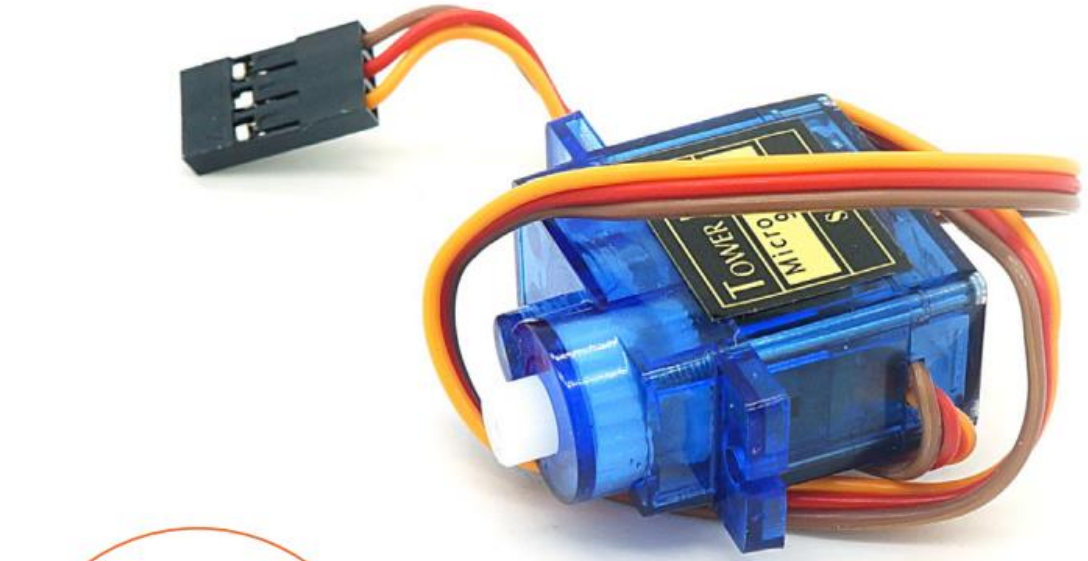
LAB01 閃爍 LED 燈

實測

請按 F5 執行程式, 即可看到 LED 每 0.5 秒閃爍一次。

目錄

3-1 伺服馬達



接地線，將它與 ESP32 的 GND 相連

供電線，將它與 ESP32 的正極相連

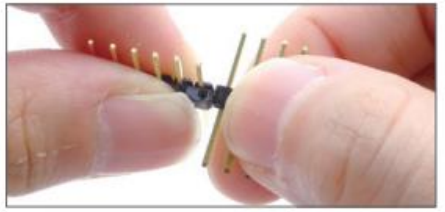
訊號線，將它與 ESP32 的 GPIO 腳位相連

⚠ 伺服馬達通電後**請不要使用外力去轉動轉軸**，否則會導致馬達毀損。

3-1 伺服馬達

伺服馬達的杜邦
母頭請自行加上
右圖的排針

排針



根據需求扭開即可分開排針

⚠ 本套件的伺服馬達規格為 **SG90**,
轉動角度為 0~180 °。

LAB02 門鎖控制

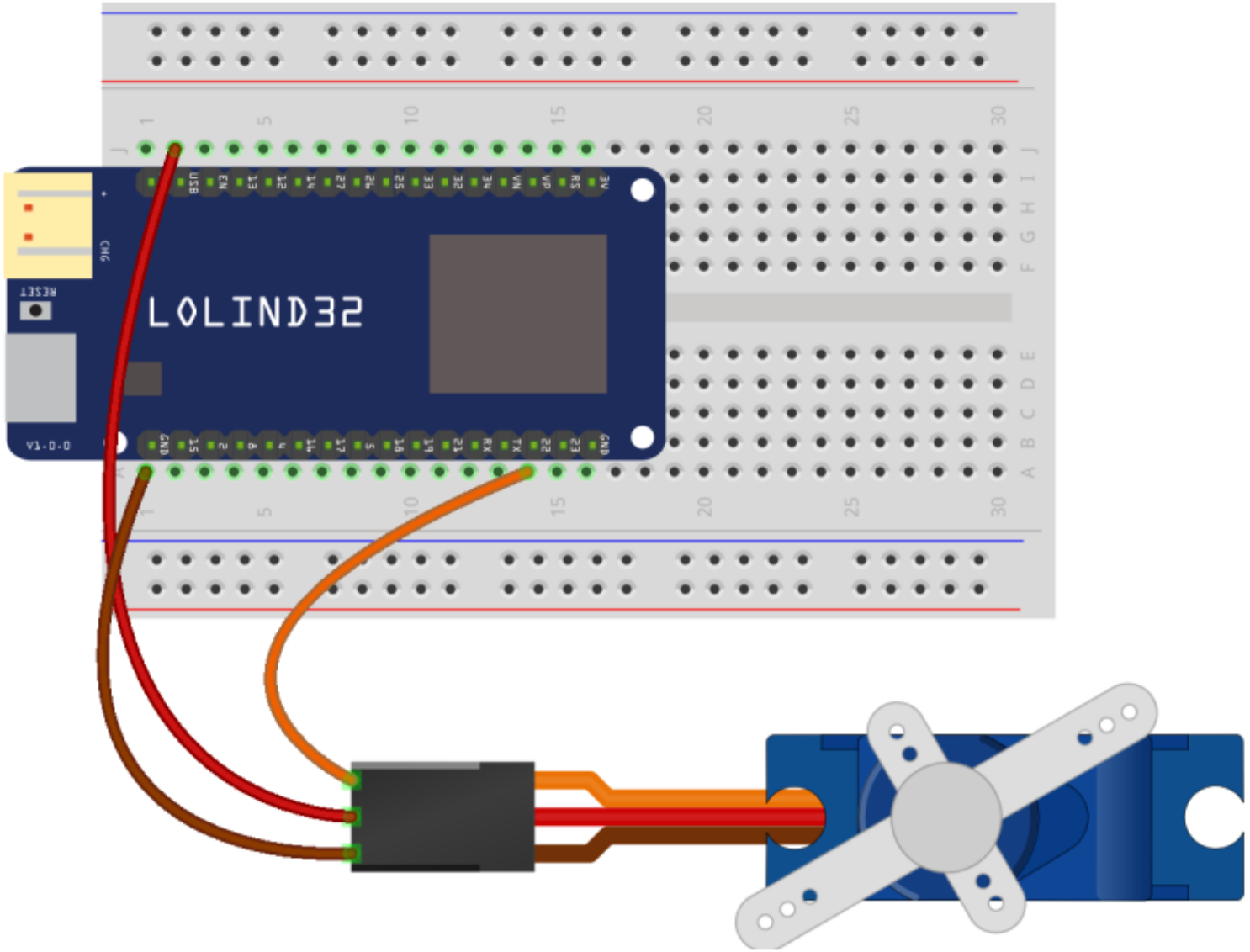
實驗目的	使用 servo 模組控制伺服馬達。
材料	• ESP32 麵包板 • 伺服馬達 • 麵包板 • 杜邦線若干條 • 排針

NEXT

線路圖

ESP32 腳位	伺服馬達
USB	紅線
GND	棕線
22	橘線

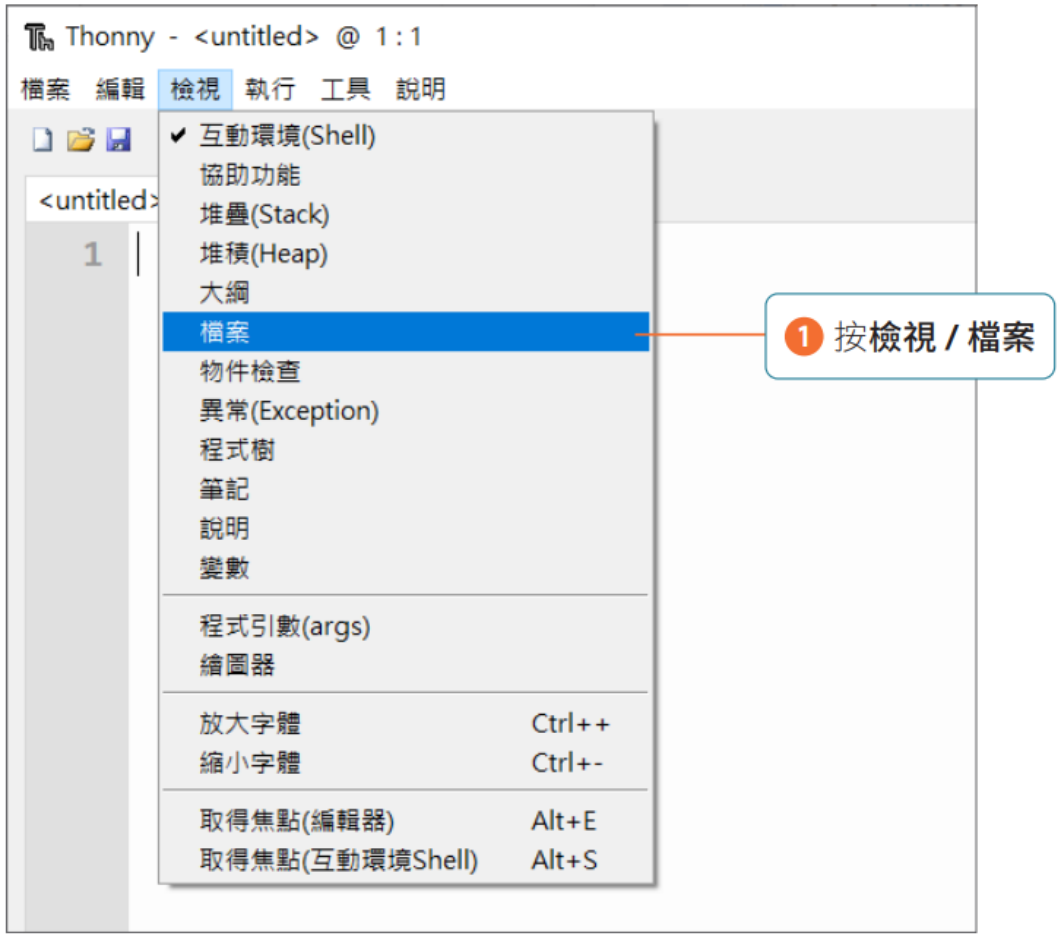
⚠ USB 腳位會輸出 5V 電壓。



NEXT

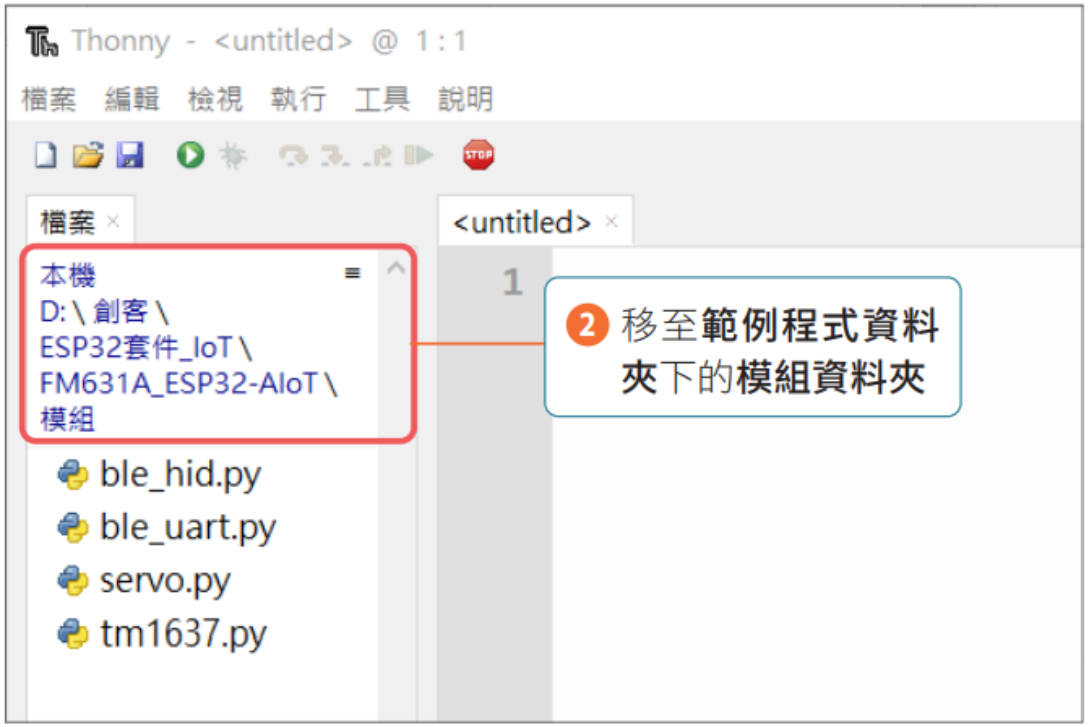
LAB02 門鎖控制

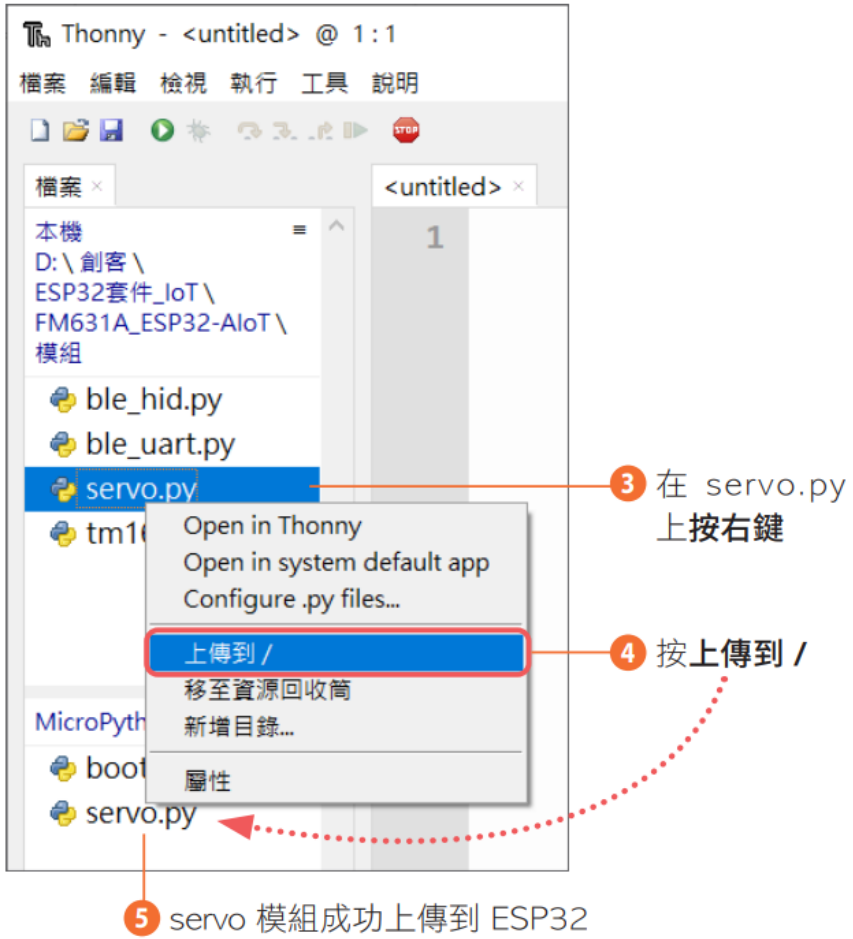
上傳 servo 模組



NEXT

LAB02 門鎖控制





NEXT

LAB02 門鎖控制

servo 模組

```
>>>  from servo import Servo      # 從 servo 模組匯入 Servo 類別
>>>  from machine import Pin
>>>  my_servo = Servo(Pin(22))   # 建立 Servo 物件
```

⚠ 請注意大小寫。

```
>>>  my_servo.write_angle(90)    # 指定馬達轉動到 90 度
>>>  my_servo.write_angle(0)     # 指定馬達轉動到 0 度
```

NEXT

LAB02 門鎖控制

程式設計

```
01  from servo import Servo
02  from machine import Pin
03  import time
04
05  # 建立伺服馬達物件
06  my_servo = Servo(Pin(22))
07
08  # 轉至 0 度
09  my_servo.write_angle(0)
10  time.sleep(1)
11  # 轉至 90 度
12  my_servo.write_angle(90)
13  time.sleep(1)
```

NEXT

LAB02 門鎖控制

實測

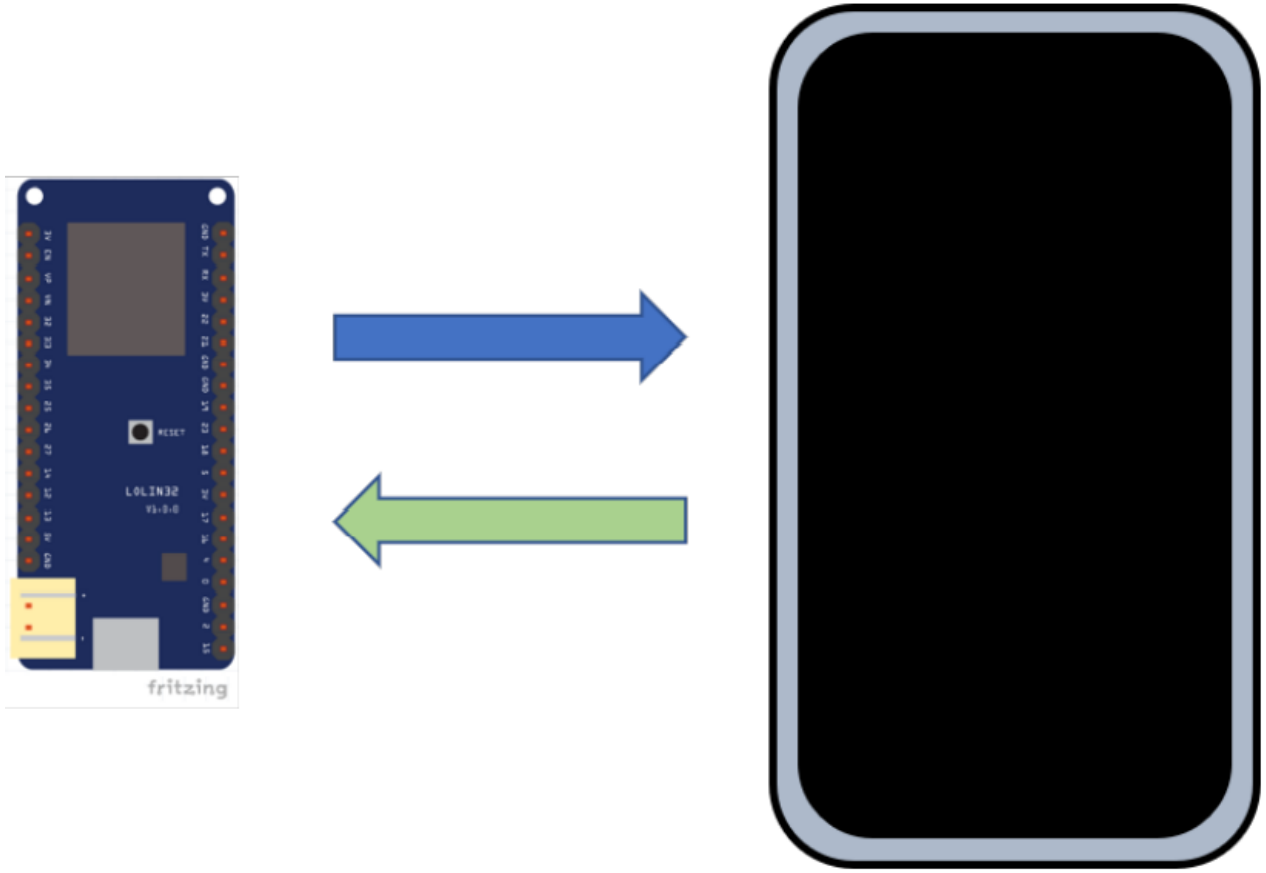
請按 F5 執行程式, 即可看到伺服馬達先轉至 0 度, 等待 1 秒後, 再轉至 90 度。

3-2 藍牙簡介

1. Bluetooth Classic：典型 (Classic) 藍牙, 可以傳送大量資料, 但會快速消耗電量。

2. Bluetooth Low Energy：低功耗藍牙 (BLE), 用於傳送少量資料, 耗電量較小, 適合用於低電量的裝置。

3-3 藍牙序列埠通訊



LAB03 門鎖遙控器

實驗目的	使用現成的手機 App 來控制伺服馬達轉動。
材料	**同 LAB02** ● 手機 (Android 和 iPhone 皆可)

接線圖

同 LAB02

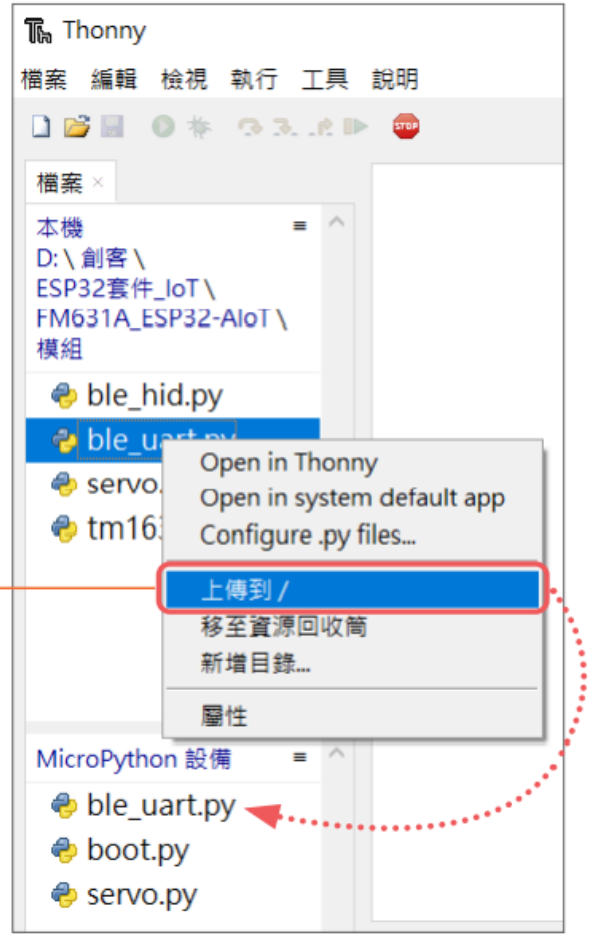
NEXT

LAB03 門鎖遙控器

設計原理

ESP32 藍牙裝置

在 ble_uart.py 上按右鍵，
並點選**上傳到 /**

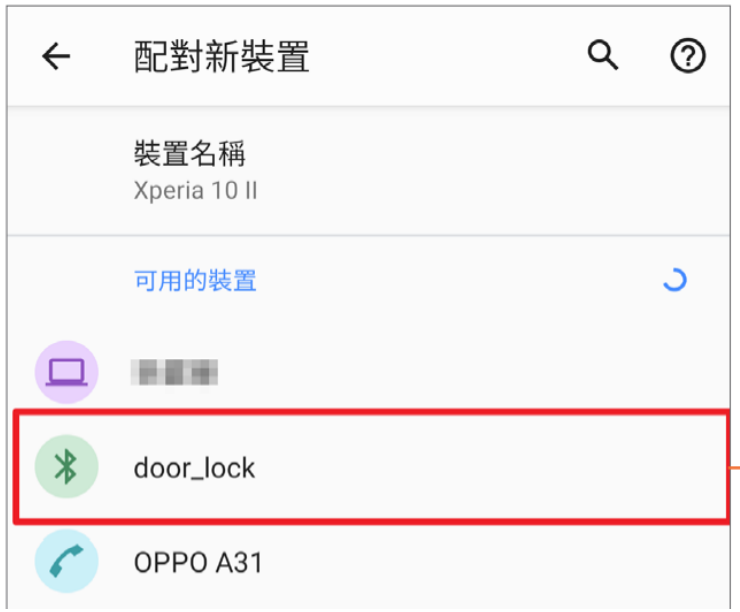


NEXT

LAB03 門鎖遙控器

```
>>>  from ble_uart import BLE_UART   # 匯入 ble_uart 模組
>>>  ble = BLE_UART("door_lock") # 建立藍牙物件, 並取名為 door_lock
```

更改名稱, 避免與其他人重複



顯示名稱為 **door_lock**

⚠ 由 ble_uart 模組所建立的藍牙裝置**必須使用藍牙 App 連線**。

NEXT

LAB03 門鎖遙控器

藍牙 App

1 在 Play 商店搜尋 nrf connect

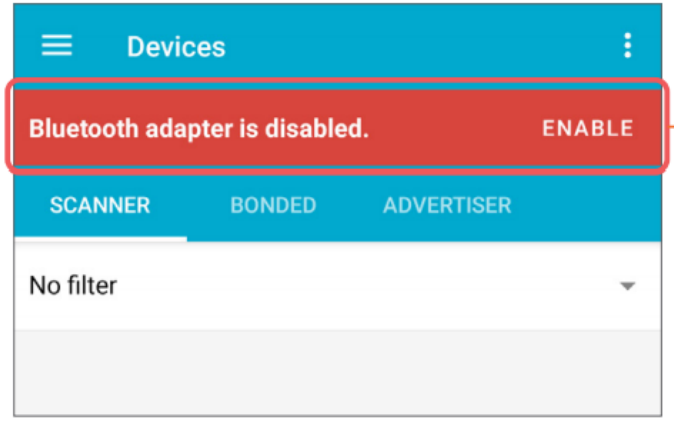
⚠ 下面以 Android 手機為例，iPhone 則是到 App Store 搜尋相同 App 即可。

iPhone 使用者請參考：
https://hackmd.io/@flagmaker/ry6ST-Get

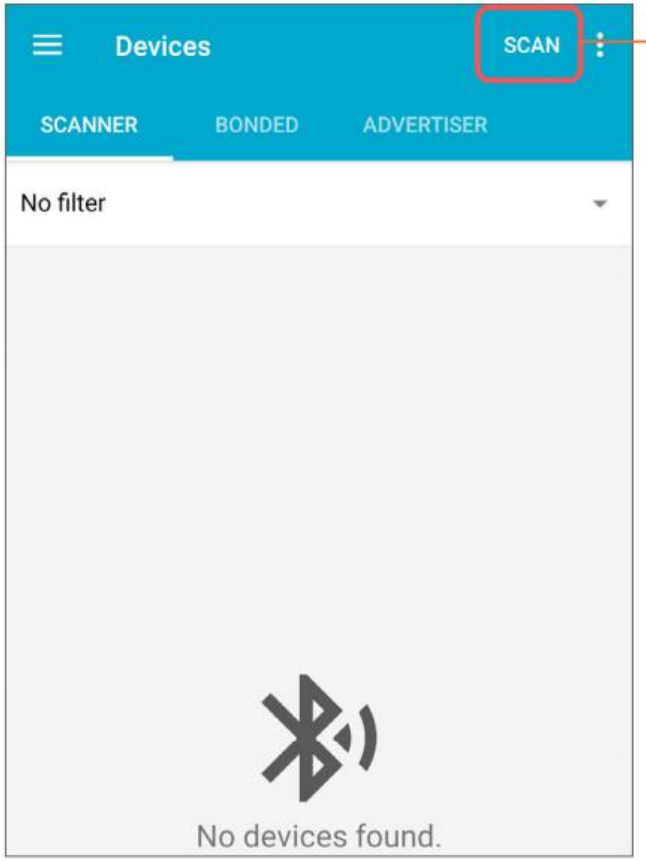
NEXT

LAB03 門鎖遙控器

2 設定 App



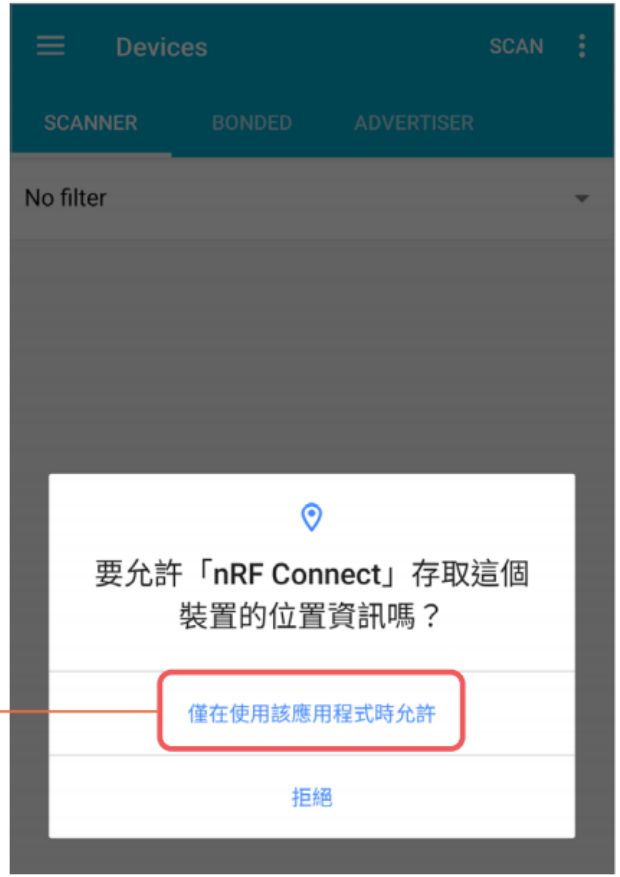
1 如果手機目前沒開啟藍牙，點擊 **ENABLE** 即可開啟



2 按 **SCAN** 來搜尋藍牙裝置

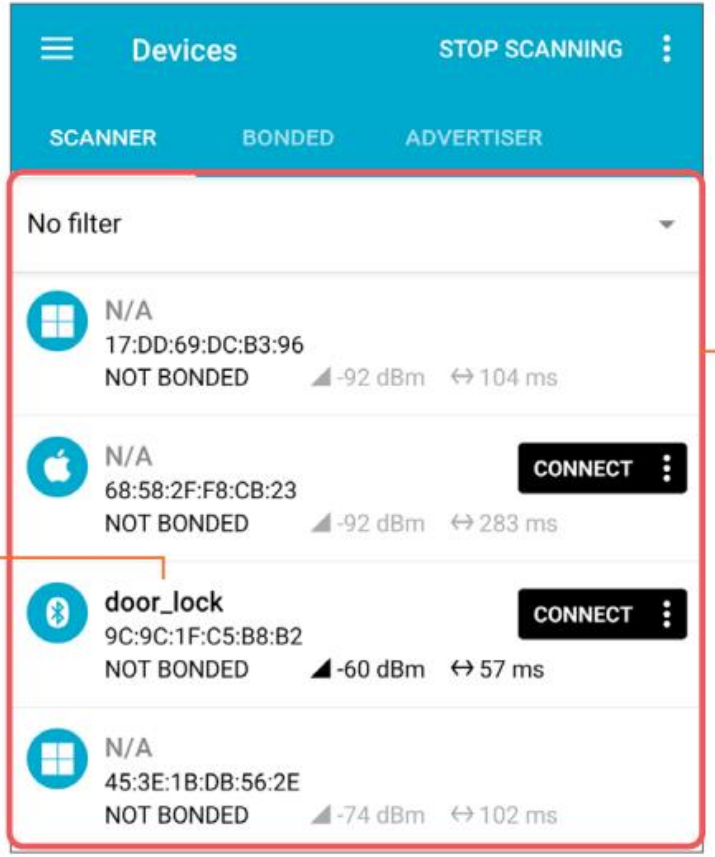
NEXT

LAB03 門鎖遙控器



3 如有提醒需要**定位權限**，選擇**允許**

請尋找自己更改的名稱

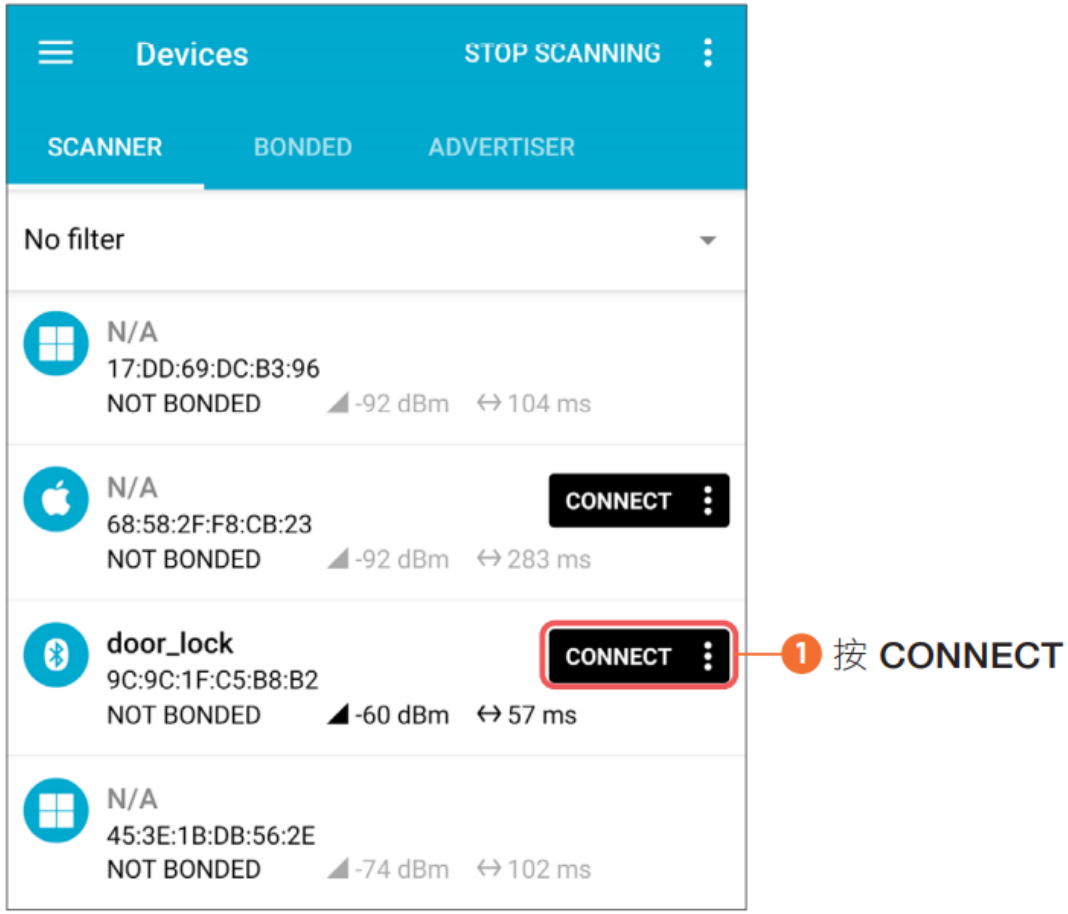


稍等一下即可顯示搜尋到的藍牙裝置

NEXT

LAB03 門鎖遙控器

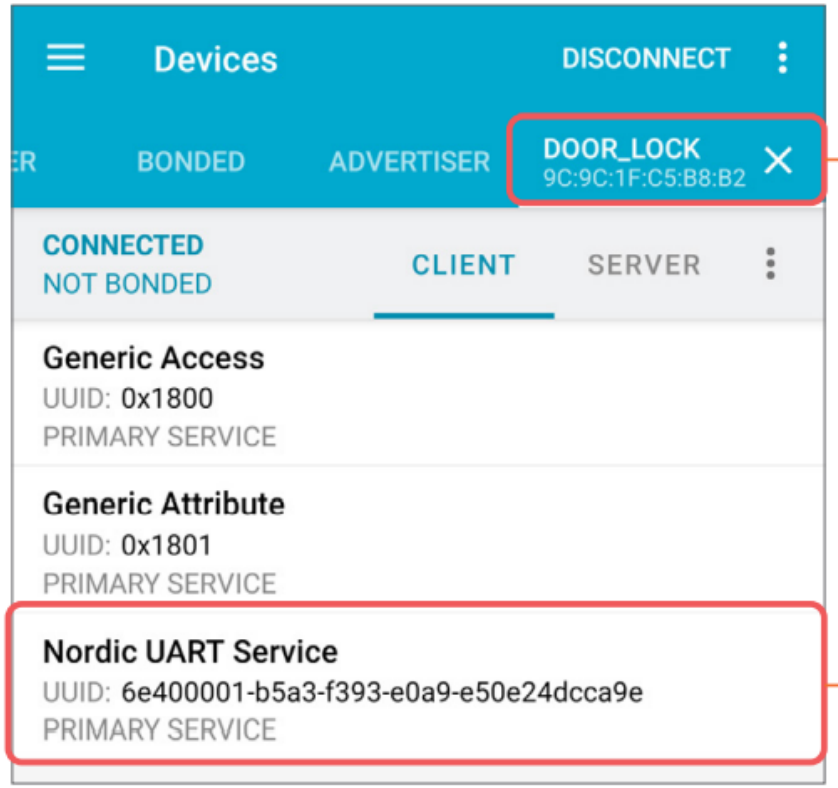
App 傳送資料



1 按 CONNECT

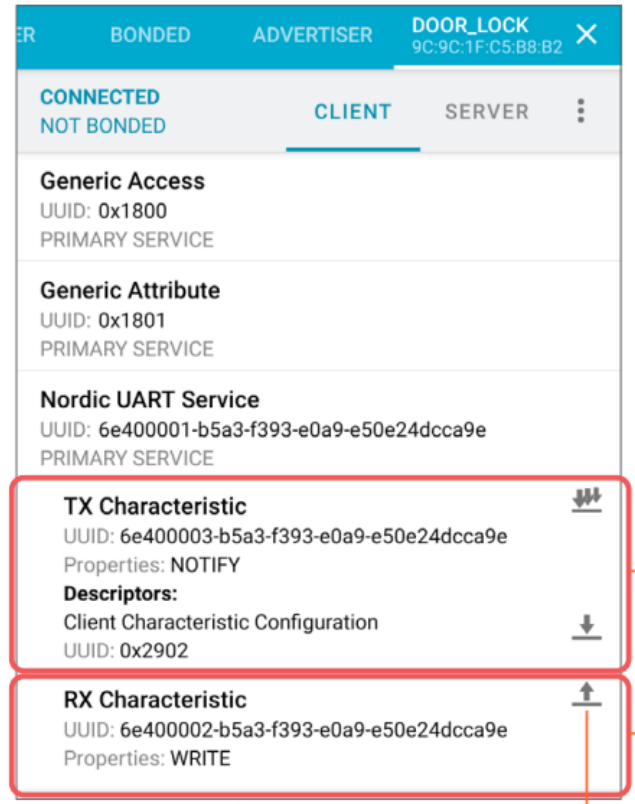
NEXT

LAB03 門鎖遙控器



連接到裝置後，會自動切換到裝置專屬的頁籤

2 點擊 Nordic UART Service



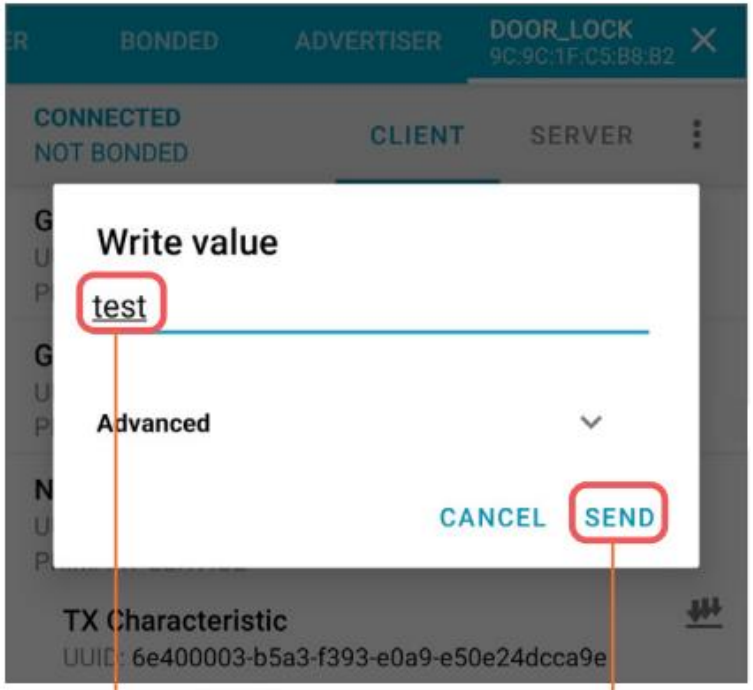
功能：接收資料

功能：傳送資料

3 按箭頭

NEXT

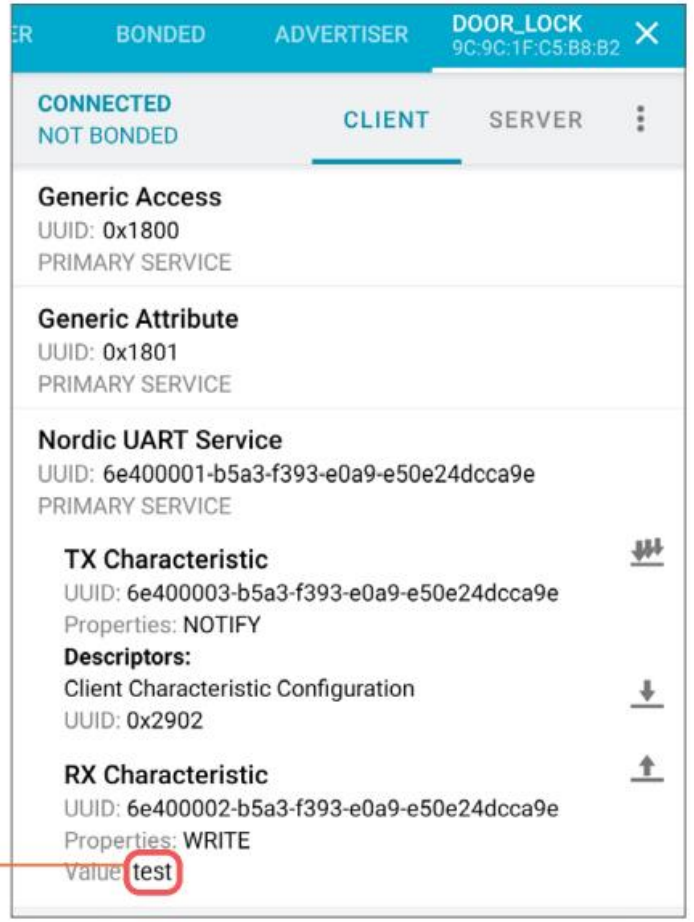
LAB03 門鎖遙控器



4 輸入 test

5 按下 SEND

出現剛剛輸入的文字，
代表傳送成功



NEXT

LAB03 門鎖遙控器

ESP32 接收資料

按一下
ENTER

```
>>> from ble_uart import BLE_UART
>>> ble = BLE_UART("door_lock")
  等待手機連線中...
>>> 連線到手機或電腦

>>> ble.get()
'test'
```

NEXT

LAB03 門鎖遙控器

程式設計

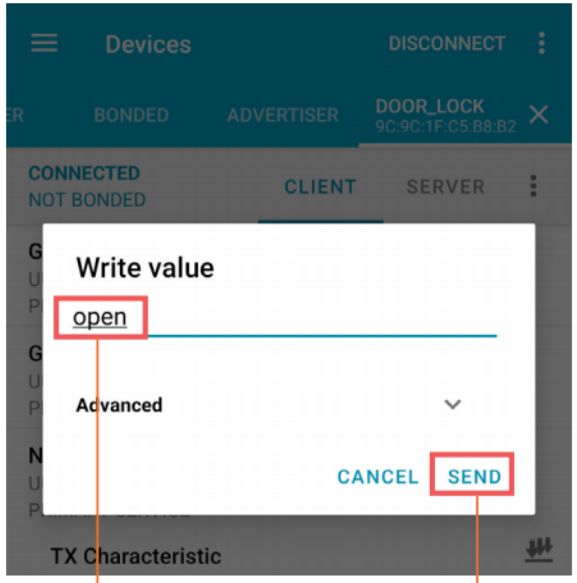
```
01  from ble_uart import BLE_UART
02  from servo import Servo
03  from machine import Pin
04
05  # 建立伺服馬達物件
06  my_servo = Servo(Pin(22))
07  # 建立藍牙物件
08  ble = BLE_UART("door_lock")
09
10  while True:
11      getValue = ble.get()
12      # 將取得的英文字母都更改為小寫
13      getValue = getValue.lower()
14      if(getValue == "open"):
15          # 轉至 0 度
16          my_servo.write_angle(0)
17          print("開啟")
18      if(getValue == "close"):
19          # 轉至 90 度
20          my_servo.write_angle(90)
21          print("關閉")
```

NEXT

LAB03 門鎖遙控器

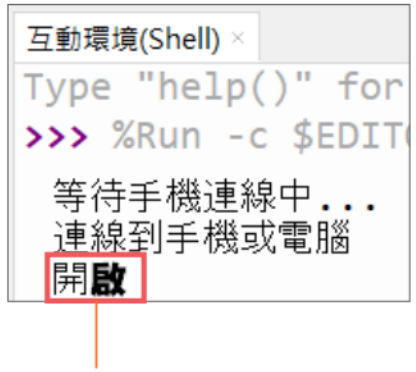
實測

開門



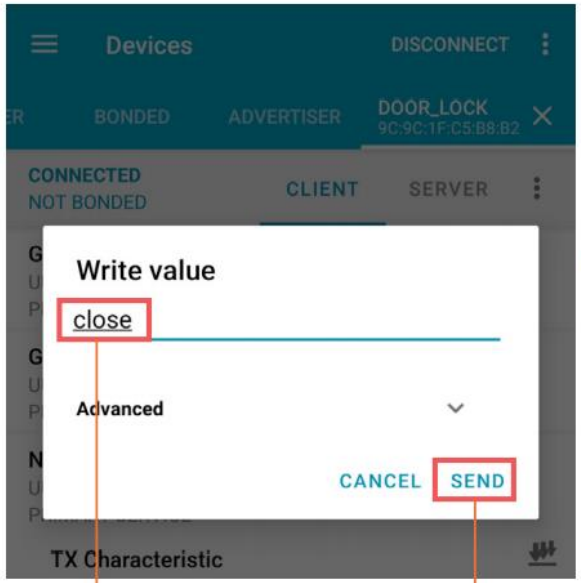
1 輸入 open

2 按 SEND



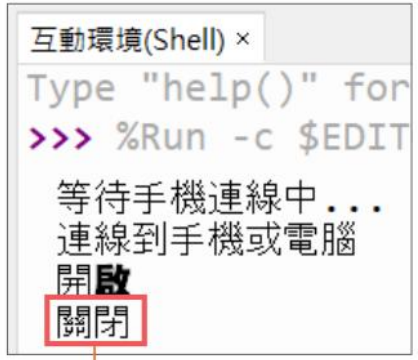
Thonny 會顯示**開啟**。
伺服馬達會轉動至 0 度

關門



1 輸入 close

2 按 SEND

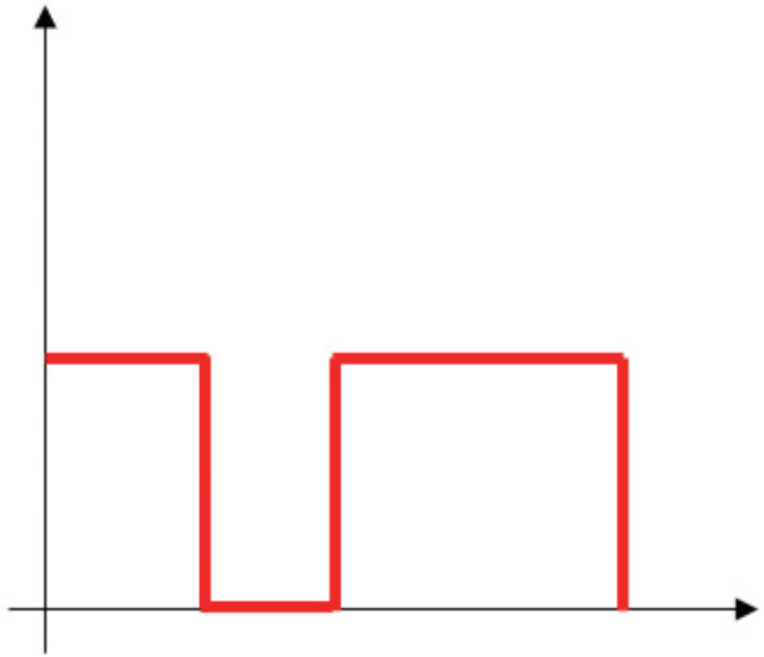


Thonny 會顯示**關閉**。
伺服馬達會轉動至 90 度。

3-4 溫度感測器

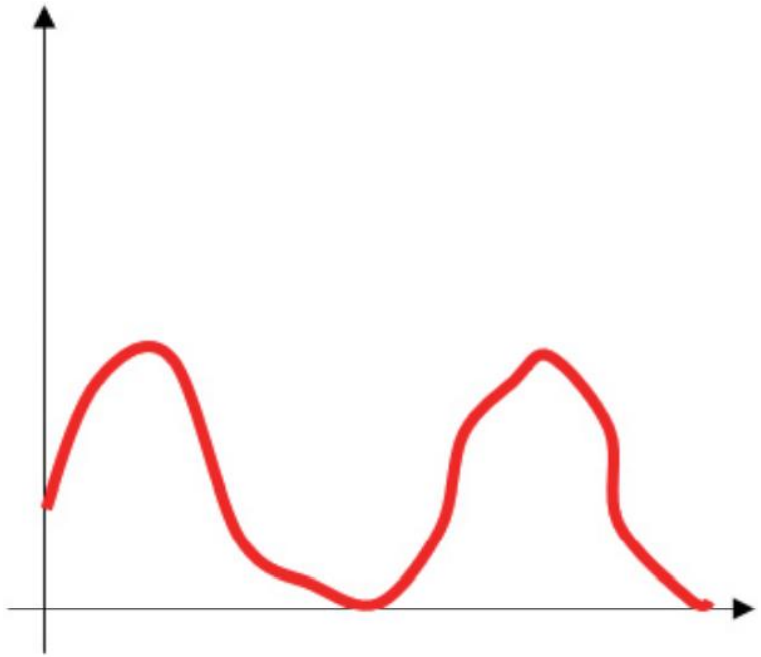
3-5 ESP32 類比輸入

不連續的訊號變化



數位訊號

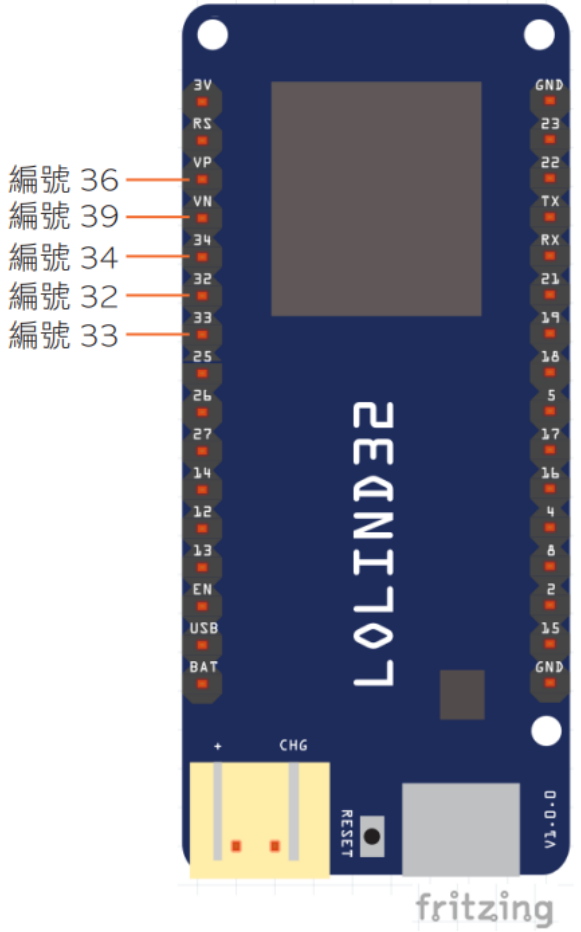
連續的訊號變化



類比訊號

3-5 ESP32 類比輸入

可使用 ADC 功能的腳位

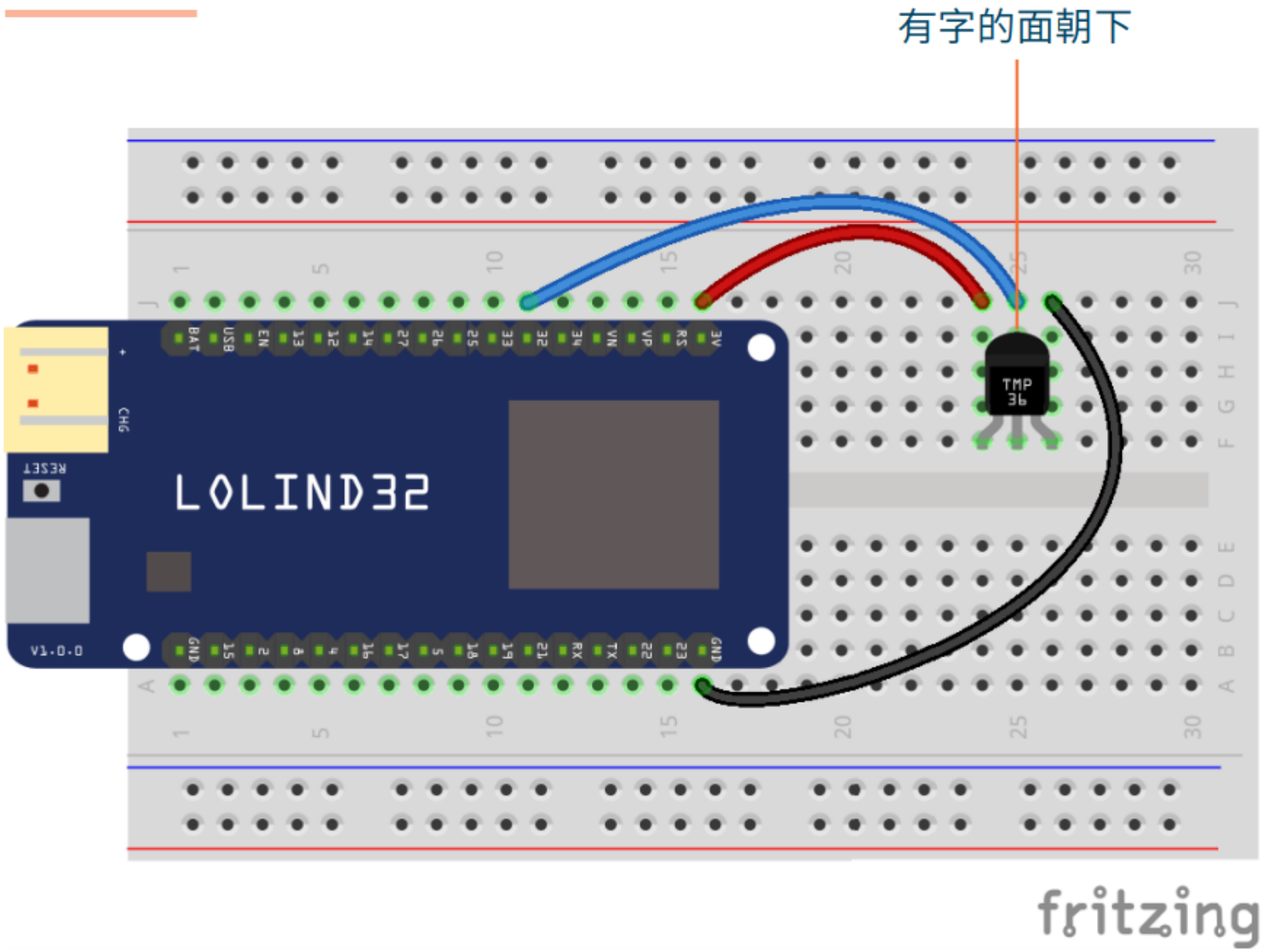


LAB04 量測溫度值

實驗目的	使用 ESP32 的類比輸入腳位讀取溫度感測器的輸出電壓，並轉換為溫度值。
材料	• ESP32 • TMP36 溫度感測器 • 杜邦線若干條 • 排針 • 麵包板

NEXT

線路圖



ESP32	TMP36 溫度感測器
3V	左邊腳
32	中間腳
GND	右邊腳

⚠ 套件內的杜邦線為**公母頭**，需要將**母頭加上排針**才可以接到麵包板上。

NEXT

LAB04 量測溫度值

設計原理

轉換公式：實際溫度 = (輸出電壓 − 0.5) × 100

讀取 ADC 值

```
>>>  from machine import Pin, ADC
>>>  adc_pin = Pin(32)
>>>  adc = ADC(adc_pin)              # 建立 ADC 物件
```

LAB04 量測溫度值

設定解析度：

```
>>>  adc.width(ADC.WIDTH_12BIT)    # 設定 ADC 解析度為 12bit
```

設定最高感測電壓：

```
>>>  adc.atten(ADC.ATTN_11DB)
```

參數	最大感測電壓
ADC.ATTN_0DB	1V
ADC.ATTN_2_5DB	1.34V
ADC.ATTN_6DB	2V
ADC.ATTN_11DB	3.6V

讀取 ADC 值：

```
>>>  adc.read()
```

NEXT

LAB04 量測溫度值

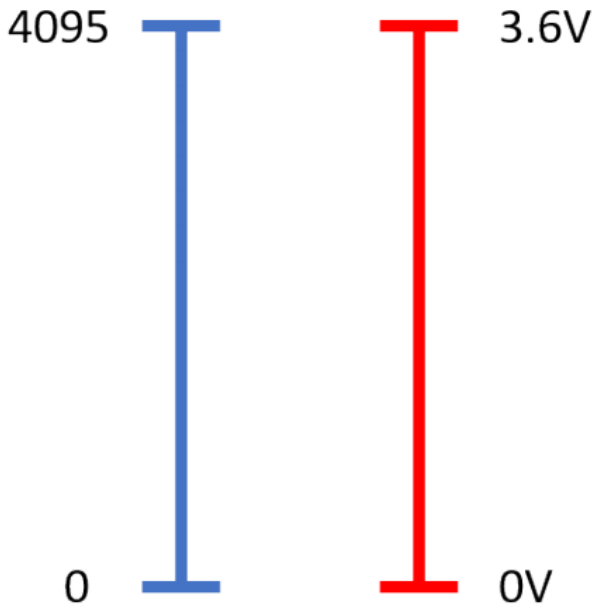
電壓轉換

ADC 值

與解析度的設定有關

電壓值

與最高感測電壓的設定有關



轉換公式：輸出電壓 $= \left(\frac{ADC\ 值}{4095}\right) \times 3.6$

NEXT

LAB04 量測溫度值

程式設計

```
01  from machine import Pin, ADC
02  import time
03
04  # 溫度感測器
05  adc_pin=Pin(32)
06  adc = ADC(adc_pin)                  # 建立 ADC 物件
07  adc.width(ADC.WIDTH_12BIT)          # 設定 ADC 解析度
08  adc.atten(ADC.ATTN_11DB)            # 將最大感測電壓設定成 3.6V
09
10  while True:
11      vol = (adc.read()/4095)*3.6
12      tem = (vol-0.5)*100
13      print("目前溫度:", tem)          # 顯示溫度值
14      time.sleep(1)
```

NEXT

LAB04 量測溫度值

實測

```
MicroPython v1.14 on
Type "help()" for mor
>>> %Run -c $EDITOR_C

  目前溫度: 23.75824
  目前溫度: 23.75824
  目前溫度: 23.23077
```

⚠ 如果溫度值太高 (大於 40 度) 或太低 (小於 0 度), 可以回到**線路圖**檢查接線是否正確。

3-6 手機溫度監控站

LAB05 手機溫度監控站

實驗目的	透過 ble_uart 模組將溫度資料傳送到手機 App
材料	**同 LAB04** • 手機 (Android 和 iPhone 皆可)

線路圖

同 LAB04

NEXT

LAB05 手機溫度監控站

設計原理

傳送資料：

```
ble.send(資料)
```

資料需要是**字串**格式, 可以使用 str() 轉換

```
ble.send(str(123))
```

NEXT

LAB05 手機溫度監控站

程式設計

```
01  from machine import Pin, ADC
02  import time
03  from ble_uart import BLE_UART
04
05  adc_pin=Pin(32)
06  adc = ADC(adc_pin)              # 建立 ADC 物件
07  adc.width(ADC.WIDTH_12BIT)      # 設定 ADC 範圍為 12BIT
08  adc.atten(ADC.ATTN_11DB)        # 將最大感測電壓設定成 3.6V
09
10  ble = BLE_UART("temperature")
11
12  while True:
13      vol = (adc.read()/4095)*3.6
14      tem = (vol-0.5)*100
15      print("目前溫度:", tem)
16      # 傳送溫度值
17      ble.send('temperature:'+ str(tem))
18      time.sleep(1)
```

NEXT

LAB05 手機溫度監控站

實測

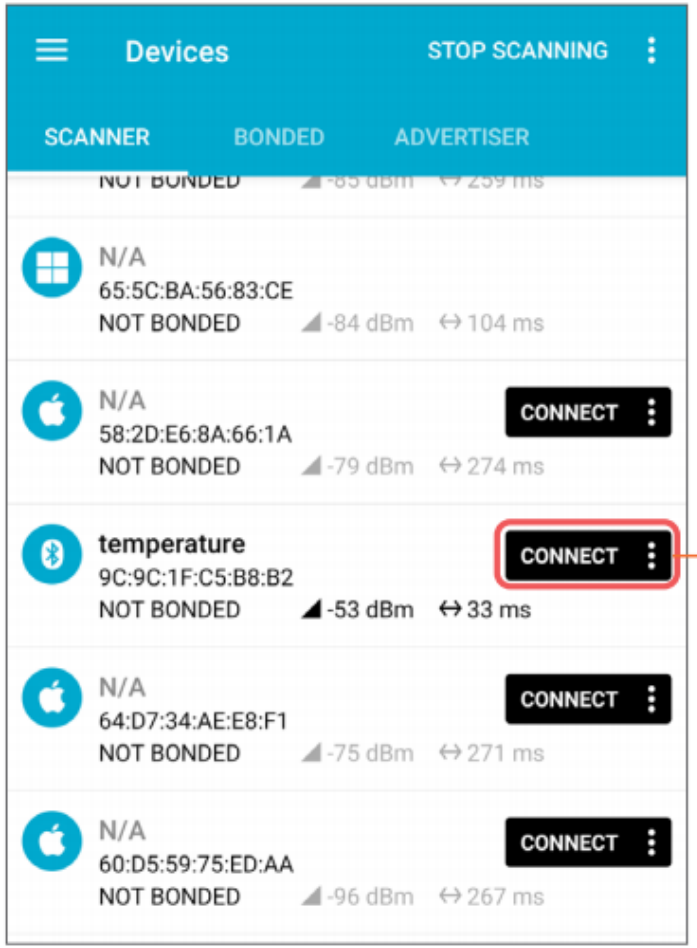
出現**等待手機連線中…**
和**目前溫度**

互動環境(Shell)

```
>>> %Run -c $EDITOR_CONTENT

  等待手機連線中...
  目前溫度: 31.05494
  目前溫度: 31.31868
  目前溫度: 30.96703
  目前溫度: 31.58242
  目前溫度: 31.67033
  目前溫度: 30.7912
  目前溫度: 31.49451
  目前溫度: 31.49451
  目前溫度: 31.67033
  目前溫度: 31.49451
```

NEXT

LAB05 手機溫度監控站

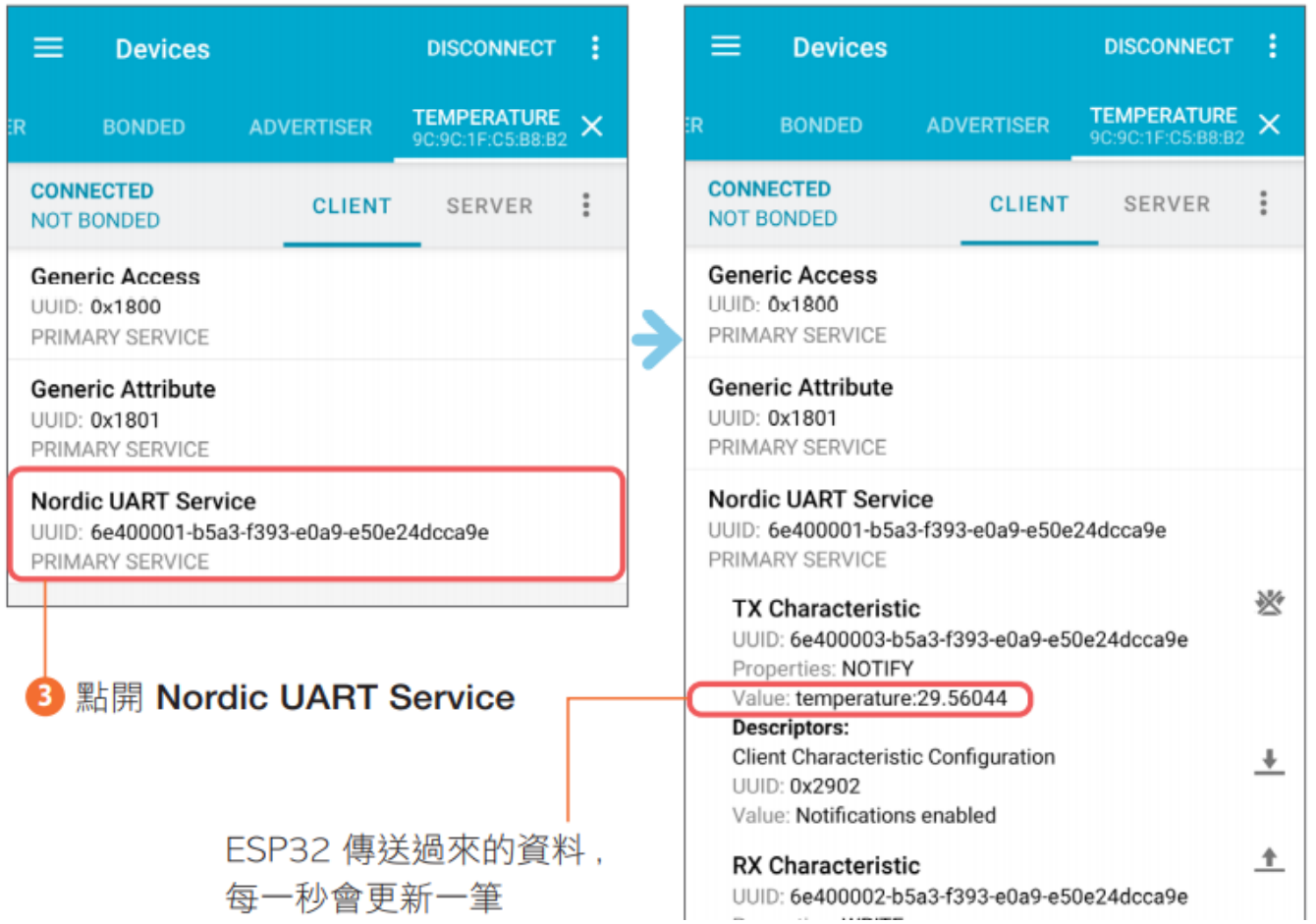


1. 按 temperature 裝置的 **CONNECT**

2. 出現**連接到手機或電腦**，且 **ESP32 的藍燈會恆亮**。

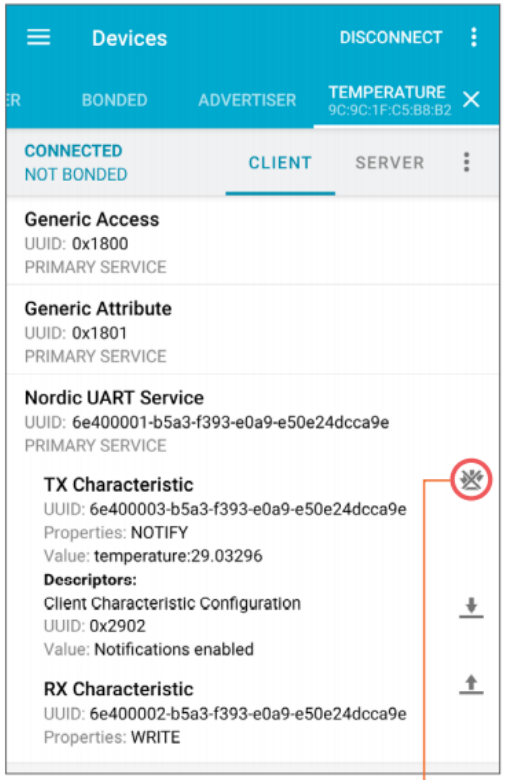
NEXT

LAB05 手機溫度監控站

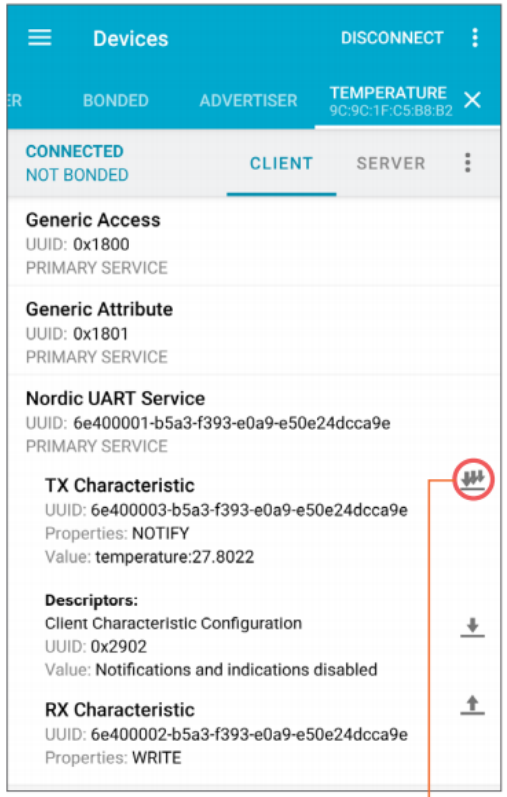


NEXT

LAB05 手機溫度監控站



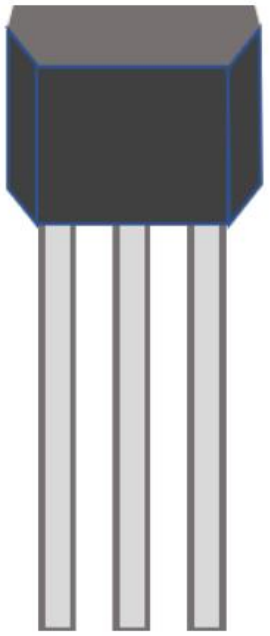
箭頭上有**叉叉**，才會自動更新資料



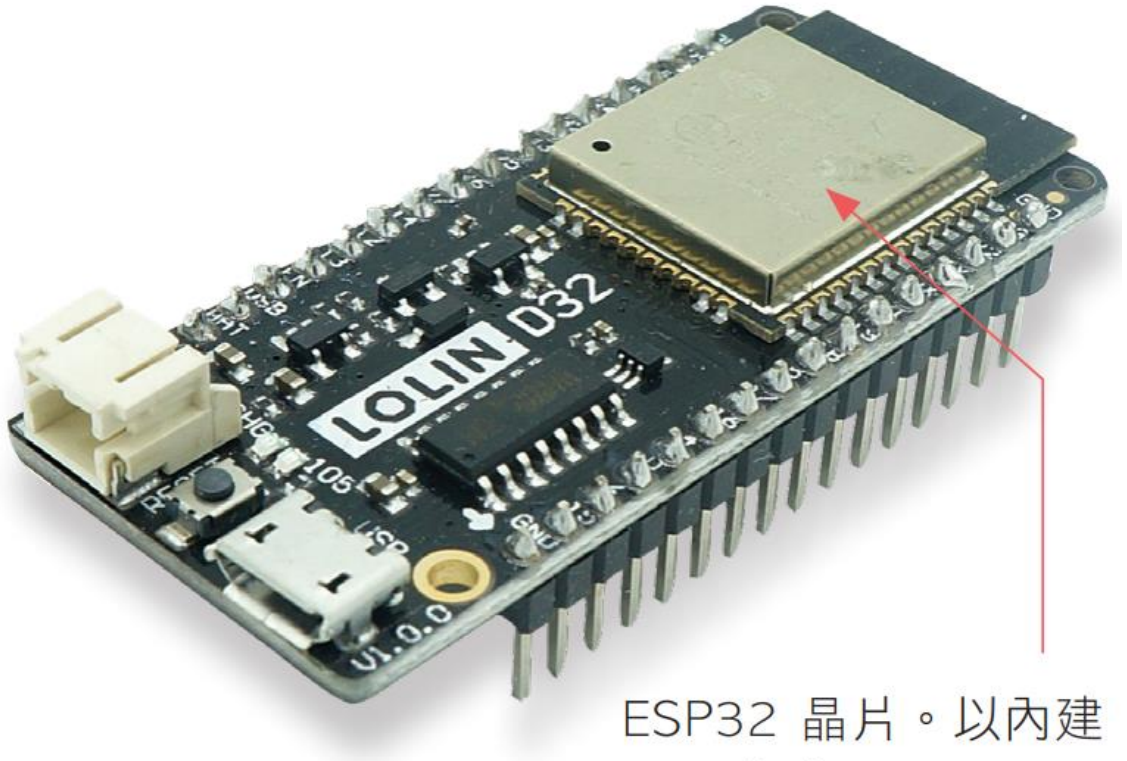
若箭頭上沒有**叉叉**，就不會自動更新資料。只要按下圖示即可自動更新資料。

目錄

4-1 霍爾感測器



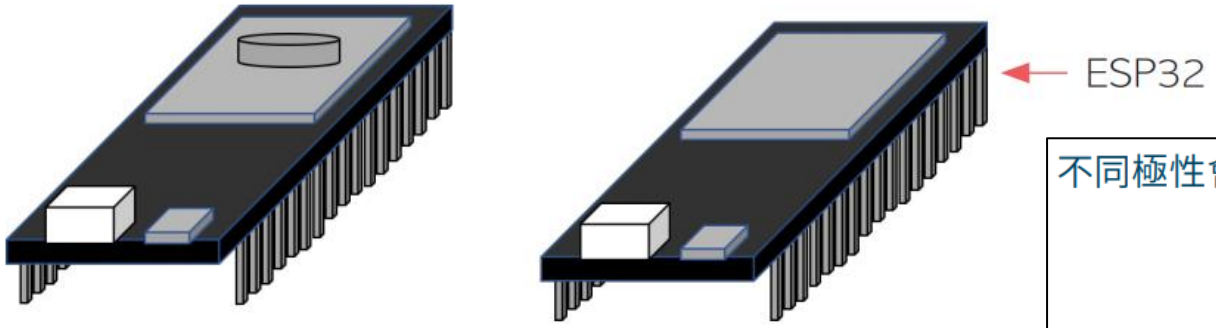
▲ 電子材料行可看到的**霍爾感測器外觀**



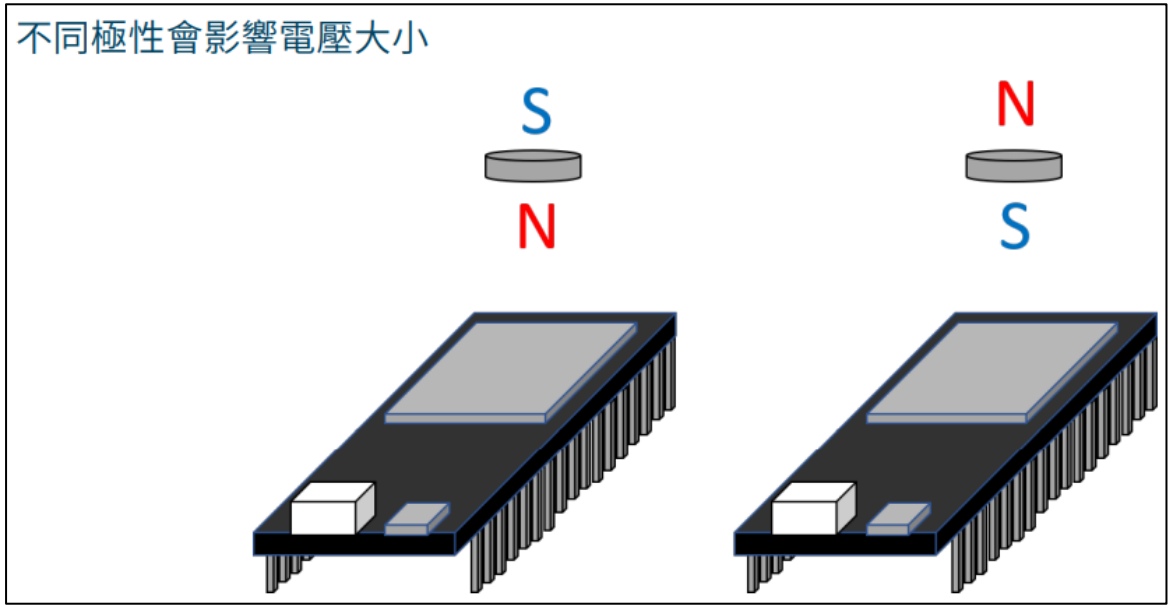
ESP32 晶片。以內建霍爾感測器

4-1 霍爾感測器

磁鐵距離會影響電壓大小



不同極性會影響電壓大小



LAB06 防盜收藏盒

實驗目的	使用 ESP32 內建的霍爾感測器確認磁鐵遠近，並在大於一定距離後，顯示收藏盒已被開啟。
材料	• ESP32 • 磁鐵

接線圖

無

NEXT

LAB06 防盜收藏盒

設計原理

霍爾感測值會根據 ESP32 個體的差異產生誤差,有些 ESP32 的霍爾感測值會在沒有磁鐵的狀況下維持在 100 ~ 150 上下。

霍爾感測器值

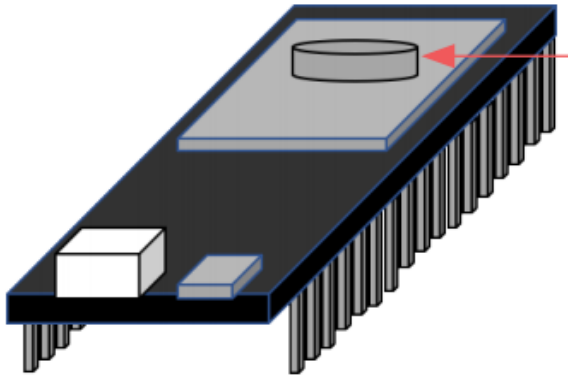
```
>>>  import esp32
```

```
>>>  esp32.hall_sensor()
76 ← 回傳值。當沒有磁鐵靠近時，感測值會在 75 上下
```

NEXT

LAB06 防盜收藏盒

磁鐵靠近 ESP32 晶片：



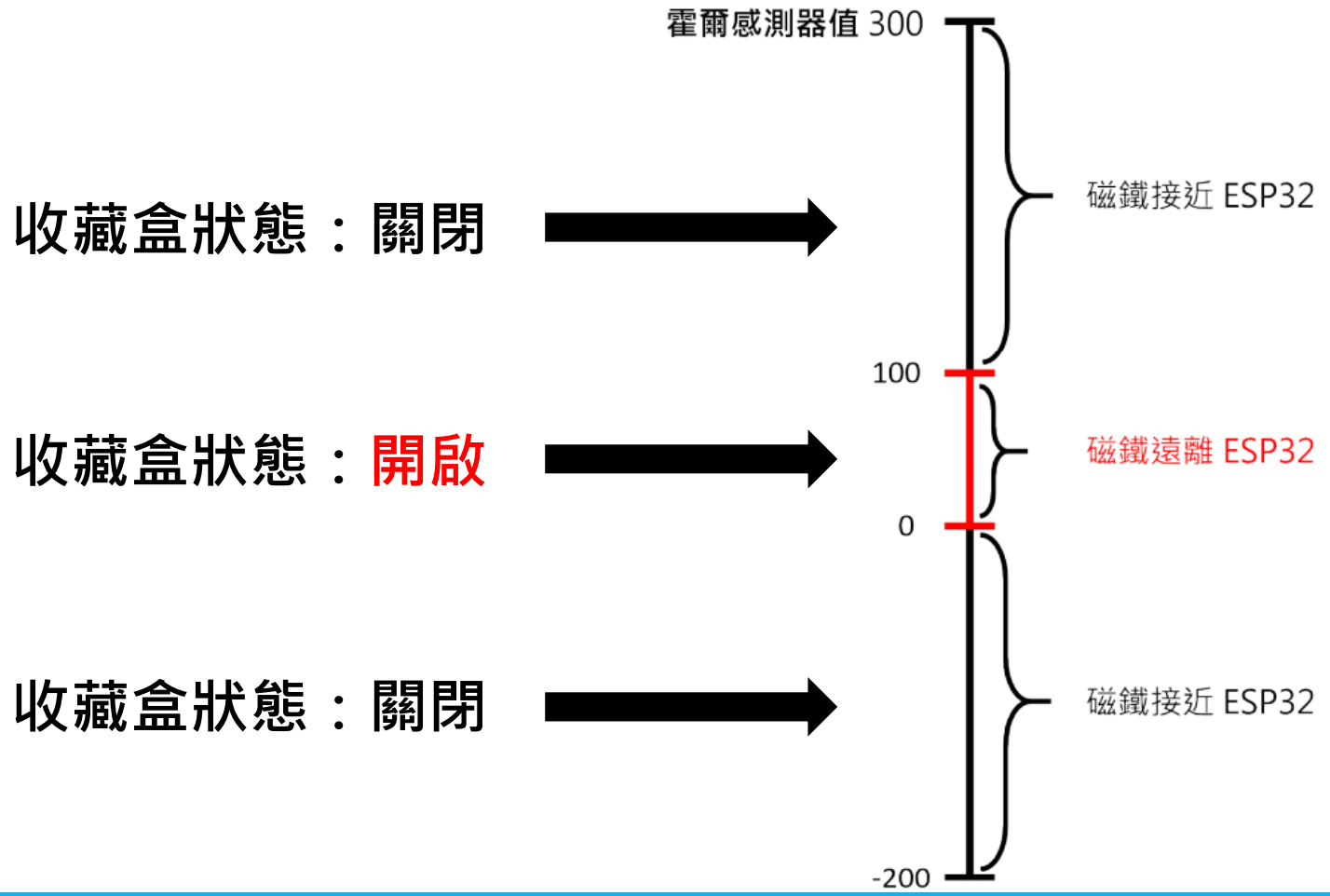

```
>>>  esp32.hall_sensor()
300 ← 回傳值。磁鐵貼到 ESP32 時，感測值會在 300 上下
```

或是

```
>>>  esp32.hall_sensor()
-200 ← 回傳值。
```

NEXT

LAB06 防盜收藏盒



NEXT

LAB06 防盜收藏盒

程式設計

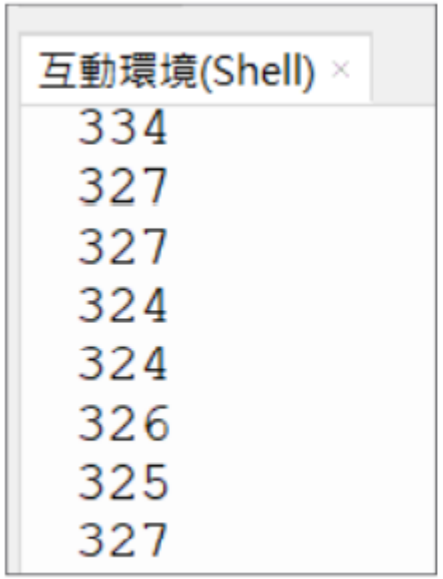
```
01  import esp32
02  import time
03
04  while True:
05      # 霍爾感測值
06      hall = esp32.hall_sensor()
07      print(hall)
08      # 如果磁鐵距離太遠
09      if(hall<100 and hall>0):
10          print("收藏盒已開啟")
11      time.sleep(0.1)
```

NEXT

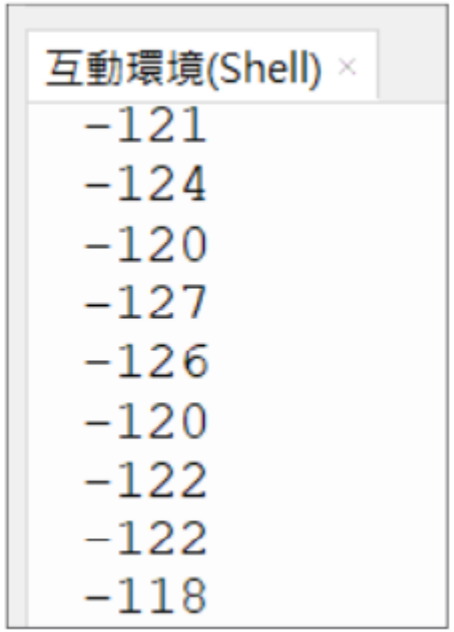
LAB06 防盜收藏盒

實測

先將磁鐵放在 ESP32 晶片上再執行程式：



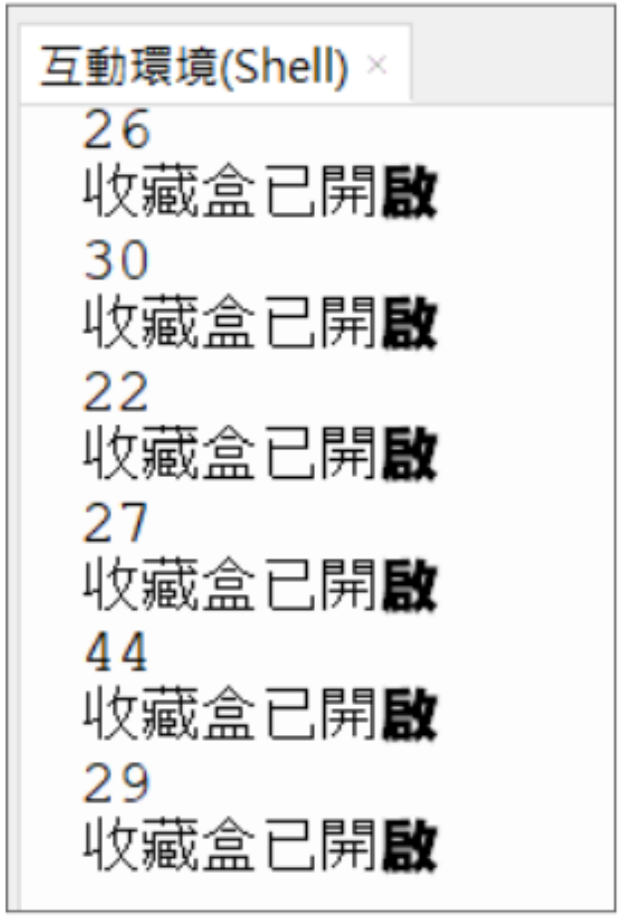
或是



NEXT

LAB06 防盜收藏盒

拿開放在 ESP32 晶片上的磁鐵：



4-2 網際網路連線

建立無線網路物件：

```
>>>  import network
>>>  sta = network.WLAN(network.STA_IF)
```

網路介面	說明
network.STA_IF	工作站 (station) 介面，專供連上現有的 Wi-Fi 無線網路基地台，以便連上網際網路
network.AP_IF	熱點 (access point) 介面，可以讓 ESP32 變成無線基地台，建立區域網路

4-2 網際網路連線

啟用無線網路介面：

```
>>>  sta.active(True)
```

連接指定的無線網路：

```
>>>  sta.connect('無線網路名稱','無線網路密碼')
```

範例：`sta.connect('FLAG','12345678')`

確認連上無線網路：

```
>>>  while not sta.isconnected():
        pass
```

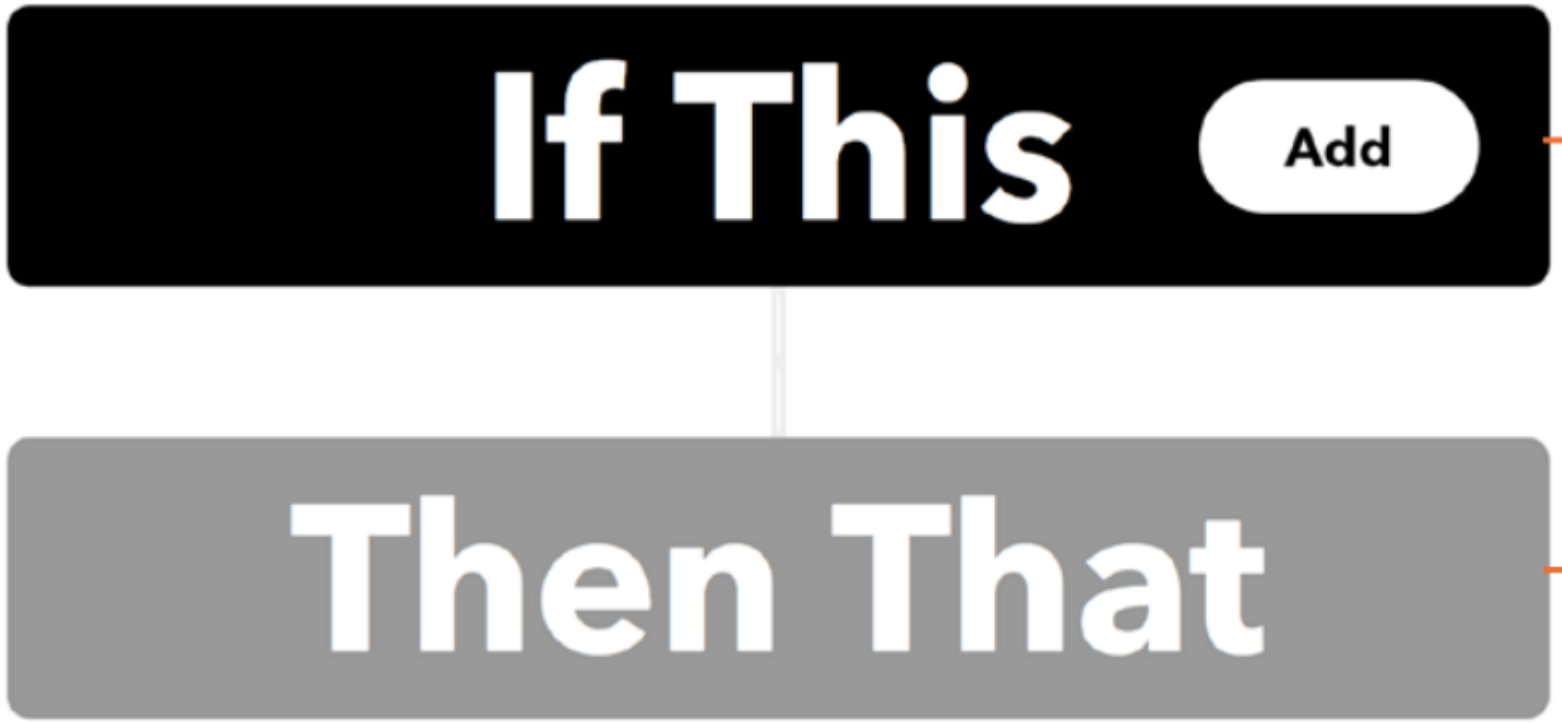
跳至自製時鐘

Python 的資料結構（容器）

Python 的資料結構

- ✓ 字串容器：由字元組成。例如：`string = "52python"`。
- ✓ `tuple` 容器：由資料物件組成。例如：`tuple = (1, '2', 3)`。
- ✓ 串列容器：由資料物件組成。例如：`list = [1, '2', 3]`。
- ✓ 集合容器：由資料物件組成。例如：`set = {1, '2', 3}`。
- ✓ 字典容器：由資料物件組成，以鍵：值表示。例如：`dick = {'A':1, 'B':'2', 'C':3}`。

4-3 使用 IFTTT 傳送 LINE 訊息



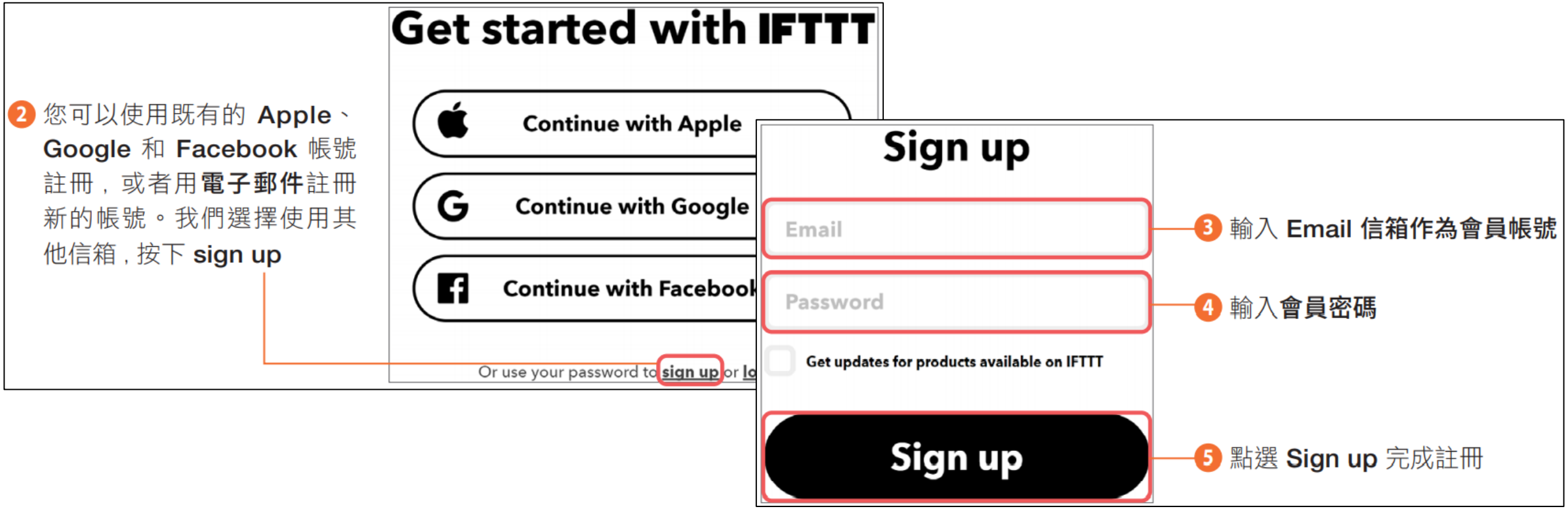
磁鐵遠離晶片
(感測值介於 0 ~100)

發送 LINE 訊息

4-3 使用 IFTTT 傳送 LINE 訊息

網址列輸入：https://ifttt.com/

4-3 使用 IFTTT 傳送 LINE 訊息



跳至火車提醒器

NEXT

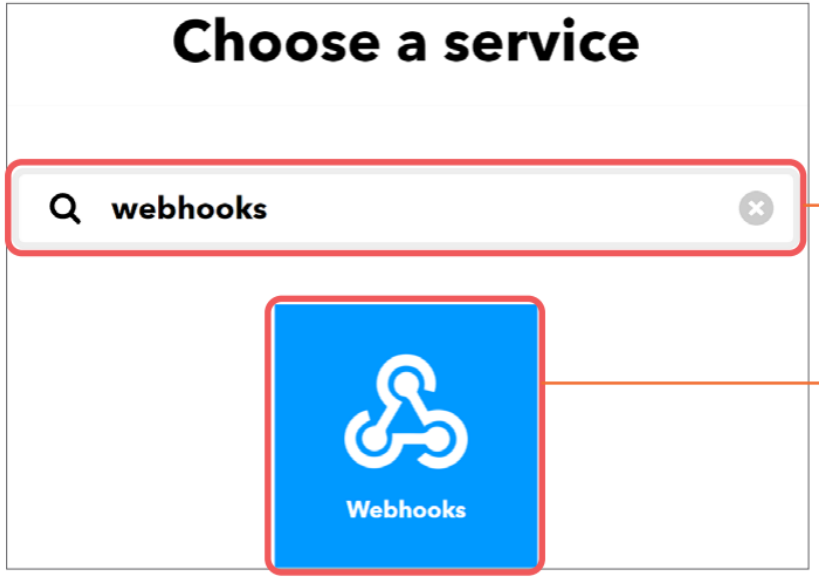
4-3 使用 IFTTT 傳送 LINE 訊息

設定 If

⚠ 免費帳號同一時間只能建立 3 個程序

NEXT

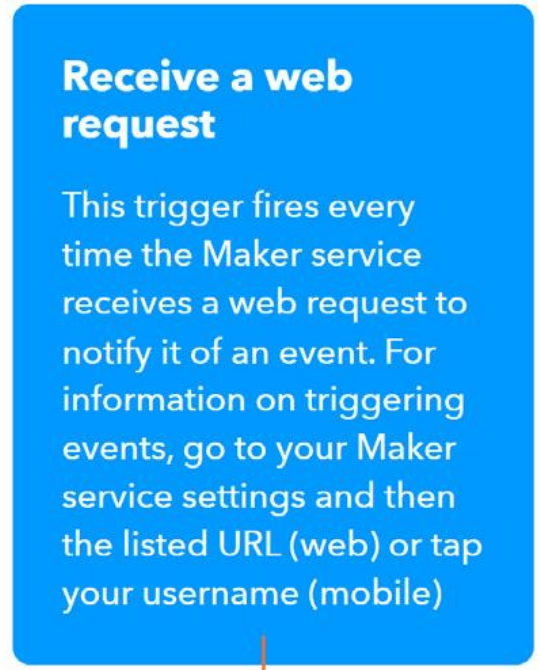
4-3 使用 IFTTT 傳送 LINE 訊息



3 輸入 webhooks

4 點選 Webhooks 圖示

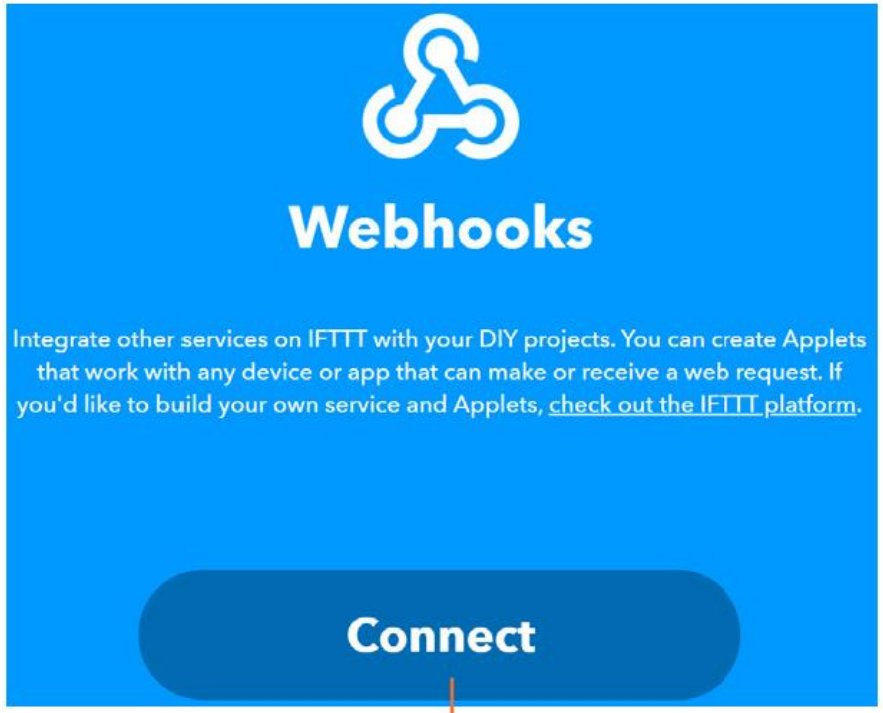
⚠ Webhooks 可以讓外部程式透過**網頁請求**驅動 IFTTT 的 IF 條件。



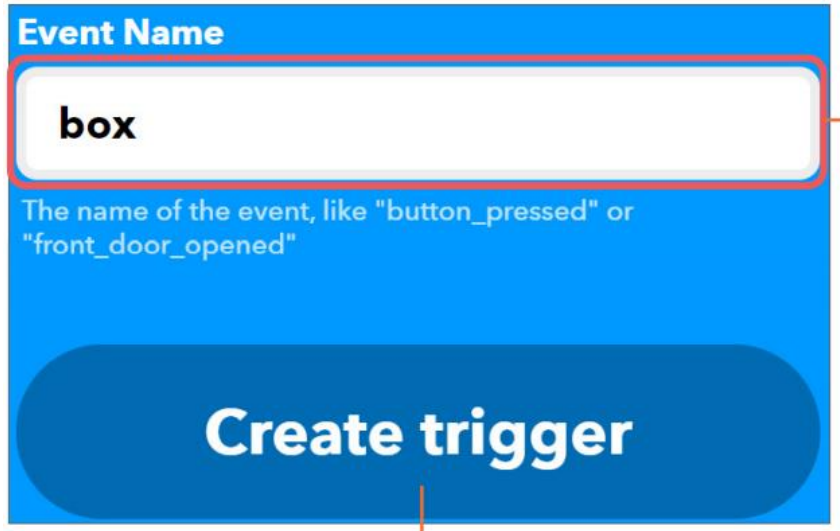
5 選擇 Receive a web request 功能

NEXT

4-3 使用 IFTTT 傳送 LINE 訊息



6 按 Connect



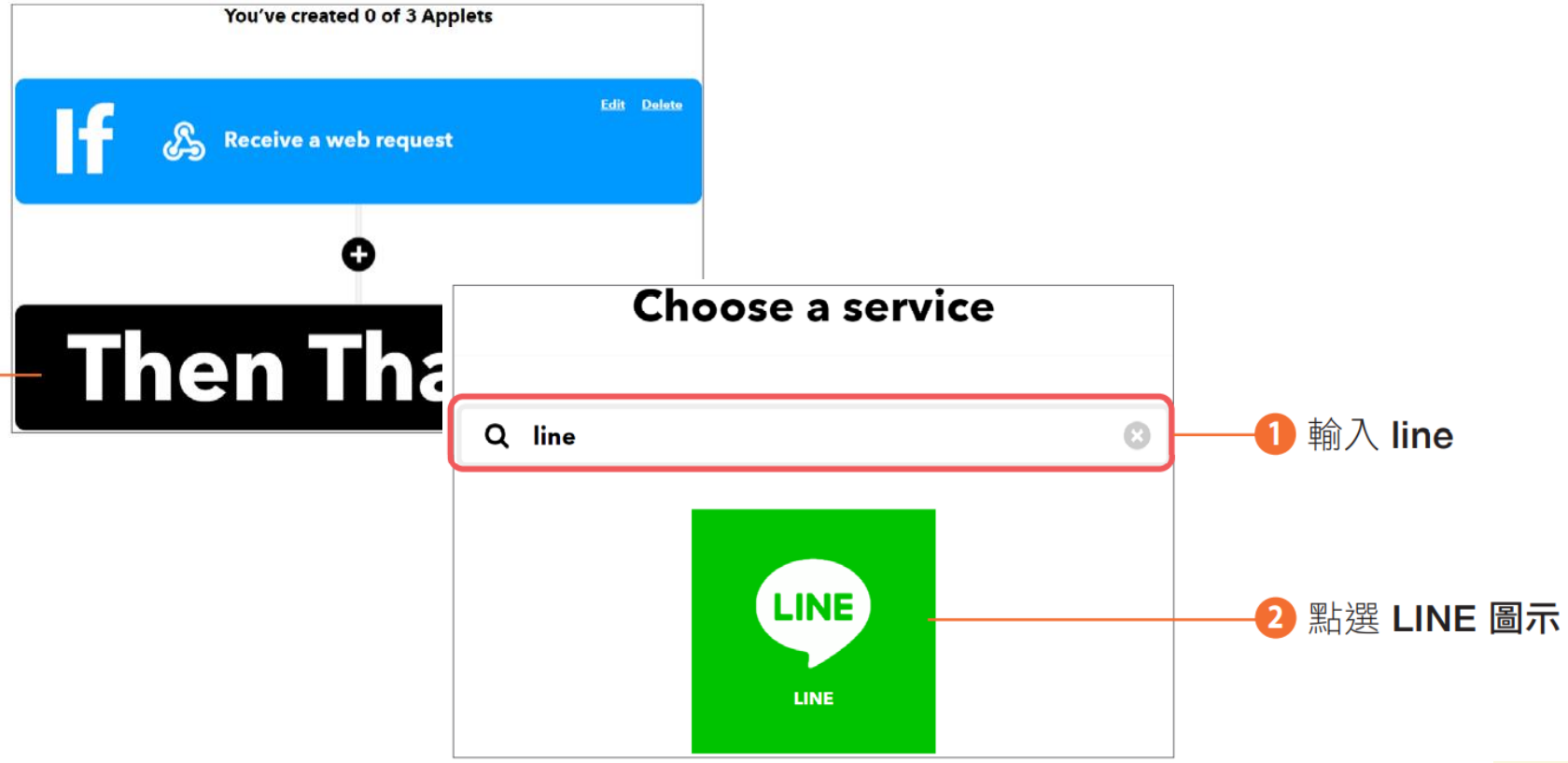
7 輸入 box

8 Create trigger

NEXT

4-3 使用 IFTTT 傳送 LINE 訊息

設定 Then



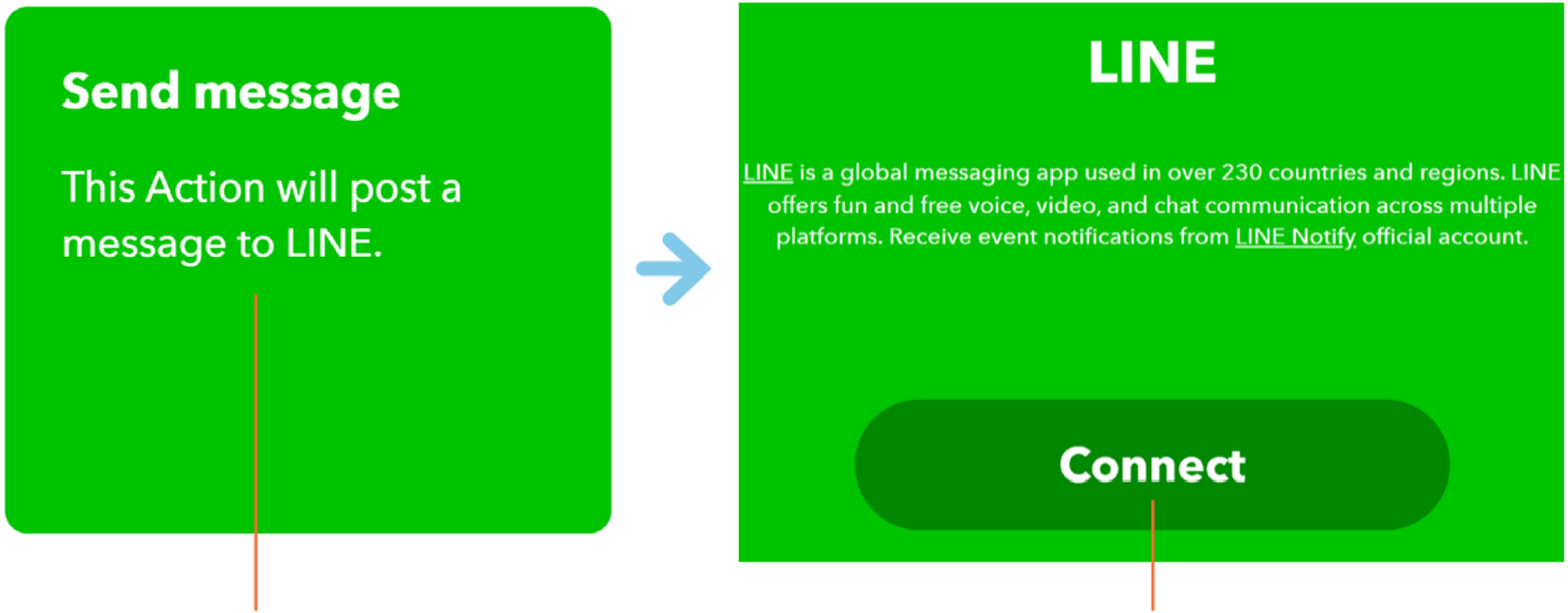
9 設定動作 B, 點選 **Then That**

1 輸入 **line**

2 點選 **LINE 圖示**

NEXT

4-3 使用 IFTTT 傳送 LINE 訊息



3 選擇 Send message 功能

4 按 Connect

NEXT

4-3 使用 IFTTT 傳送 LINE 訊息

NEXT

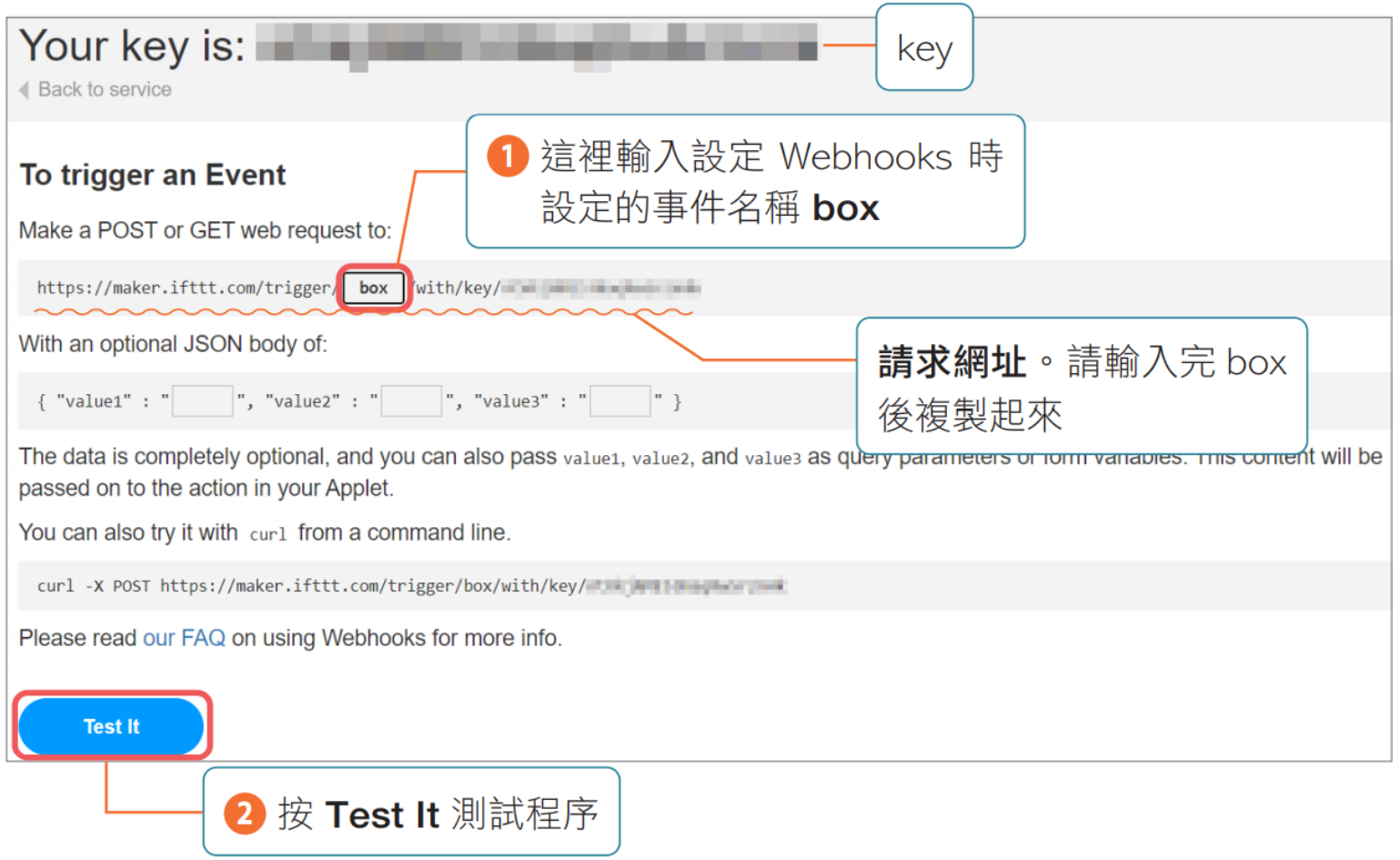
4-3 使用 IFTTT 傳送 LINE 訊息

4-3 使用 IFTTT 傳送 LINE 訊息

測試程序是否可以正常執行：

NEXT

4-3 使用 IFTTT 傳送 LINE 訊息



LAB07 防盜收藏盒 - LINE 通知

實驗目的	使用霍爾感測器確認收藏盒是否被打開，如果是則發出 LINE 通知
材料	同 LAB06

線路圖

同 LAB06

NEXT

LAB07 防盜收藏盒 - LINE 通知

設計原理

匯入內建模組：

```
>>>  import urequests
```

發出請求：

```
>>>  response = urequests.get("請求路徑")
```

關閉連線：

```
>>>  response.close()
```

LAB07 防盜收藏盒 - LINE 通知

程式設計

```
01  import esp32
02  import time
03  import network      # 匯入 network 模組
04  import urequests    # 匯入 urequests 模組
05
06  # IFTTT 網址
07  url = 'IFTTT 請求網址'
08
09  # 連線至無線網路
10  sta=network.WLAN(network.STA_IF)
11  sta.active(True)
12  # 更換無線網路名稱、密碼
13  sta.connect('無線網路名稱','無線網路密碼')
14
15  while not sta.isconnected():
```

NEXT

LAB07 防盜收藏盒 - LINE 通知

實測

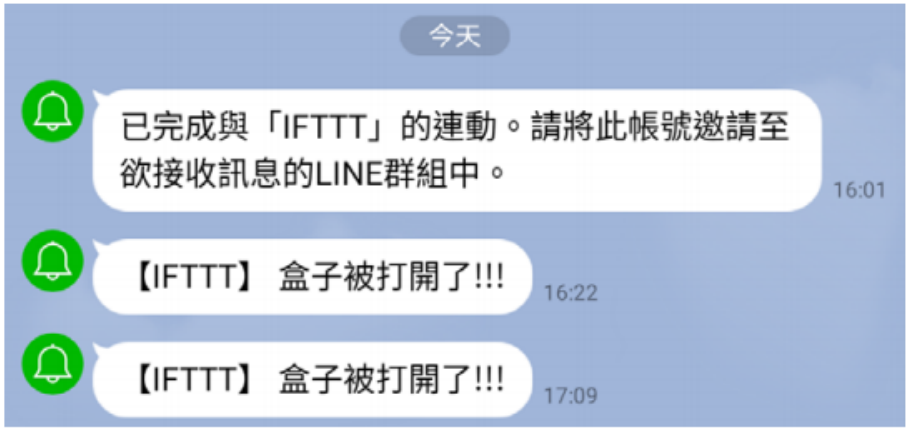
互動環境(Shell)

```
Type "help()" f
>>> %Run -c $ED

  Wi-Fi連線成功
  -95
  -95
  -106
  -100
  -96
```

拿開磁鐵

互動環境(Shell)

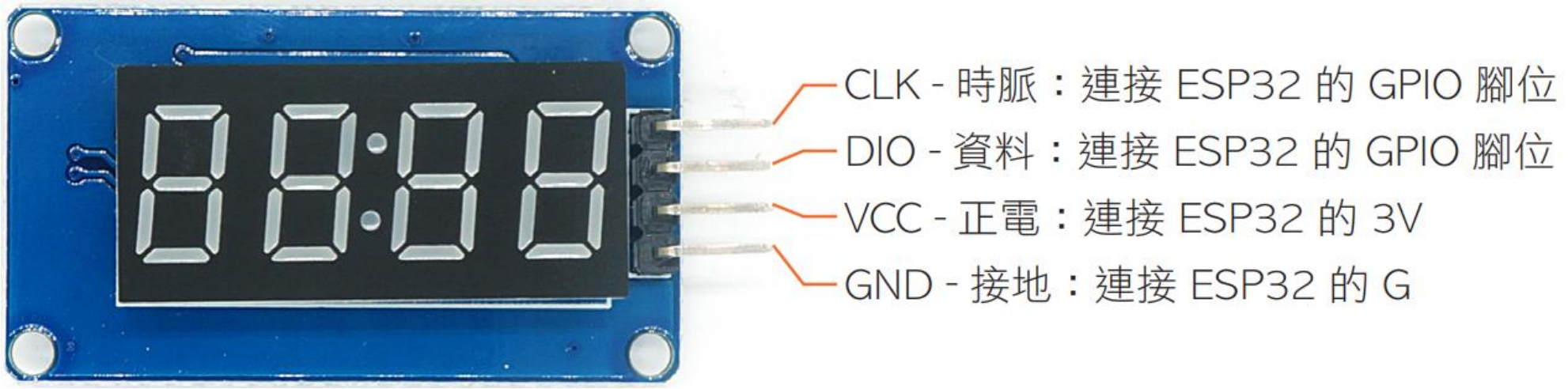
```
-211
-218
-196
-105
-16
6
發送警訊!!!!
傳送成功
```



目錄

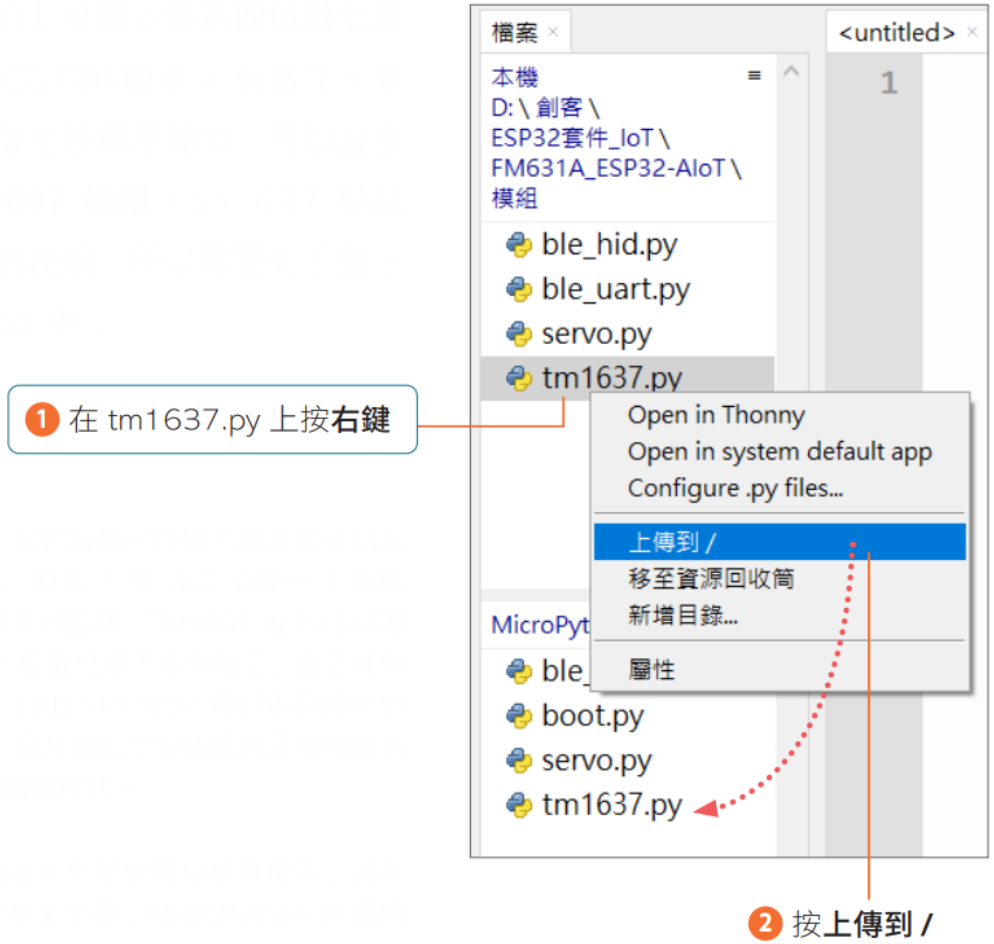
圖片來源：https://www.ettoday.net/news/20140228/329890.htm

5-1 四位數七段顯示器


CLK - 時脈：連接 ESP32 的 GPIO 腳位
DIO - 資料：連接 ESP32 的 GPIO 腳位
VCC - 正電：連接 ESP32 的 3V
GND - 接地：連接 ESP32 的 G

5-1 四位數七段顯示器

上傳模組：



5-1 四位數七段顯示器

匯入模組並建立物件：

```
>>>  import tm1637
>>>  tm = tm1637.TM1637(clk=Pin(16), dio=Pin(17))
```

顯示數字：

```
tm.number(數字)
```

顯示數字(分別指定前後數字)：

```
tm.numbers(數字, 數字)
```

5-2 自製時鐘

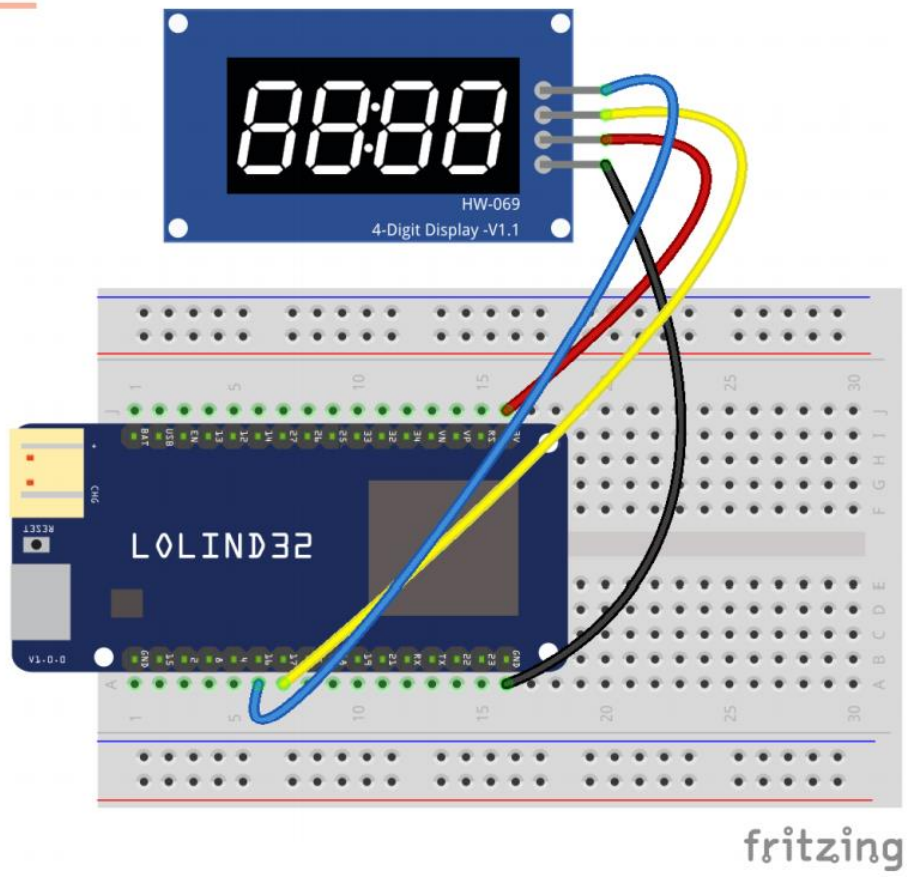
LAB08 自製時鐘

實驗目的	使用 ESP32 取得現在台灣時間，並且顯示在四位數七段顯示器上
材料	• ESP32 • 四位數七段顯示器 • 麵包板 • 杜邦線若干條

NEXT

LAB08 自製時鐘

接線圖



ESP32 腳位	四位數七段顯示器
16	CLK
17	DIO
3V	VCC
GND	GND

NEXT

LAB08 自製時鐘

設計原理

即時時鐘 RTC

```
>>>  import time
>>>  time.localtime()
(2021, 5, 21, 4, 7, 57, 4, 141)
```

年	月	日	時	分	秒	星期幾 (0~6)	今年的第幾天
2021	5	21	4	7	57	4	141

NEXT

LAB08 自製時鐘

網路時間協定 NTP (Network Time Protocal), 與 UTC(世界協調時間)同步

匯入內建模組：

```
>>>  import ntptime
```

將 RTC 更新為 UTC：

```
>>>  ntptime.settime()
```

⚠ ESP32 需要先連接網際網路

連接無線網路教學

NEXT

LAB08 自製時鐘

更改為台灣時間

將時間簡化為秒：

```
>>>  time.localtime()
(2021, 5, 21, 8, 16, 28, 4, 141)
>>>  time.mktime(time.localtime())
674900188
```

增加 8 小時 (28800 秒)：

```
>>>  TW_sec = time.mktime(time.localtime())+28800
```

NEXT

LAB08 自製時鐘

(2021, 5, 21, 8, 16, 28, 4, 141)

藉由 NTP 將 ESP32 的 RTC 更新為 UTC

將 UTC 簡化成秒數

674900188

增加 28800 秒變成台灣時間

將秒數轉換回正常時間

(2021, 5, 21, 16, 16, 28, 4, 141)

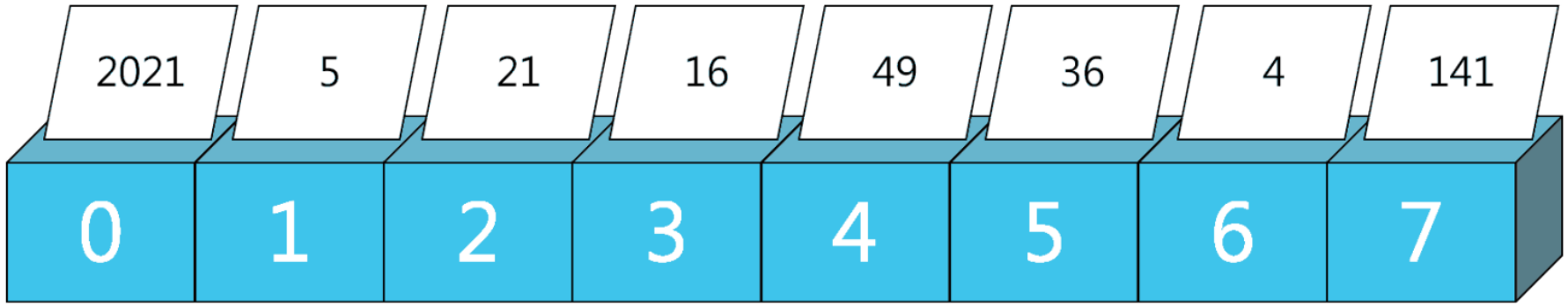
674928988

NEXT

LAB08 自製時鐘

從返回值取出特定資料

```
>>>  TW = (2021, 5, 21, 16, 49, 36, 4, 141)
```




```
>>>  TW[3]
16  ←  時
>>>  TW[4]
49  ←  分
```

NEXT

LAB08 自製時鐘

程式設計

```
01  import tm1637
02  from machine import Pin
03  import ntptime
04  import time
05  import network
06
07  # 四位數顯示器
08  tm = tm1637.TM1637(clk=Pin(16), dio=Pin(17))
09
10  # 連線至無線網路
11  sta=network.WLAN(network.STA_IF)
12  sta.active(True)
13  sta.connect('無線網路名稱', '無線網路密碼')
14  while not sta.isconnected() :
```

NEXT

LAB08 自製時鐘

實測

互動環境(Shell)

```
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Wi-Fi連線成功
  (2021, 5, 25, 9, 43, 41, 1, 145)
  (2021, 5, 25, 9, 43, 42, 1, 145)
  (2021, 5, 25, 9, 43, 43, 1, 145)
  (2021, 5, 25, 9, 43, 44, 1, 145)
  (2021, 5, 25, 9, 43, 45, 1, 145)
  (2021, 5, 25, 9, 43, 46, 1, 145)
  (2021, 5, 25, 9, 43, 47, 1, 145)
```

5-3 無源蜂鳴器

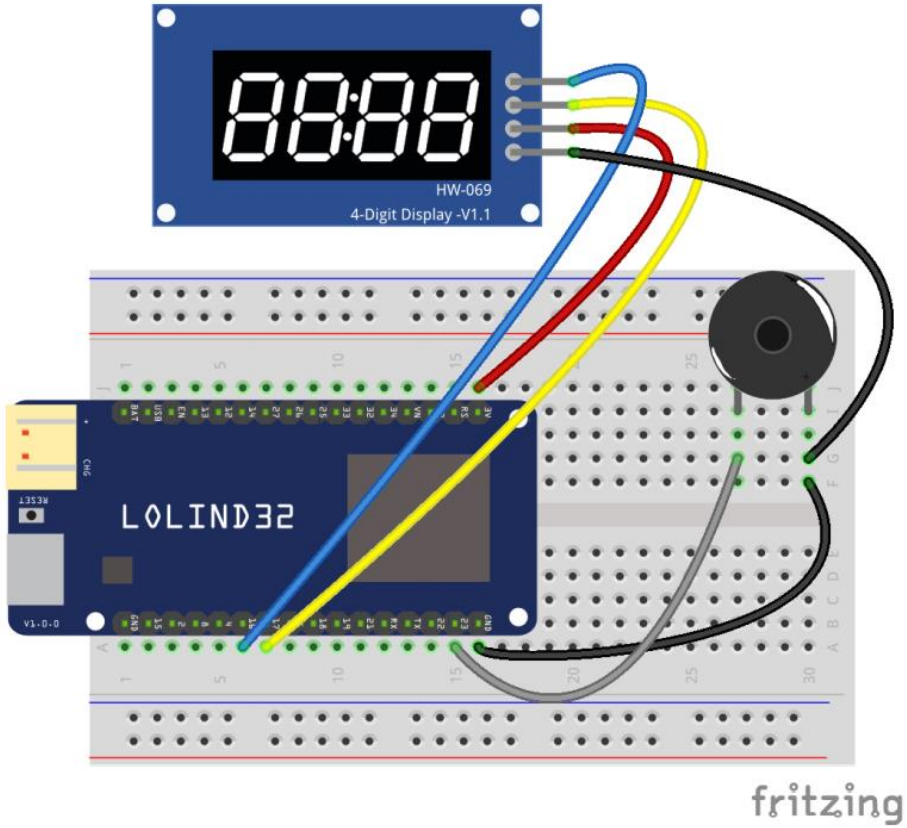
LAB09 自製警報器

實驗目的	透過無源蜂鳴器發出警報聲。
材料	• ESP32 • 麵包板 • 杜邦線若干條 • 排針

NEXT

LAB09 自製警報器

接線圖



ESP32 腳位	無源蜂鳴器
23	左右腳皆可
GND	左右腳皆可

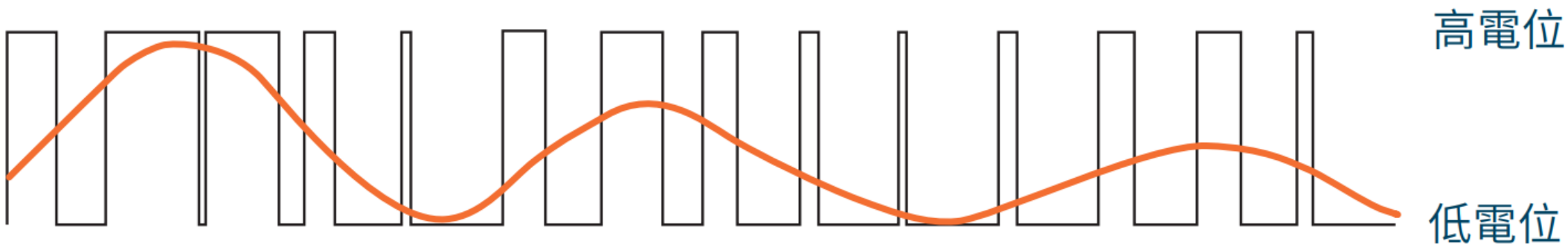
⚠ 前一個實驗的四位數七段顯示器接線不用拔除 (GND 需要調整一下)，下一個實驗 LAB10 會繼續使用到。

NEXT

LAB09 自製警報器

設計原理

PWM



▲ 模擬出來的電壓值

NEXT

LAB09 自製警報器

工作週期

▲ 工作週期 10% (10% 高電位 , 90% 低電位)

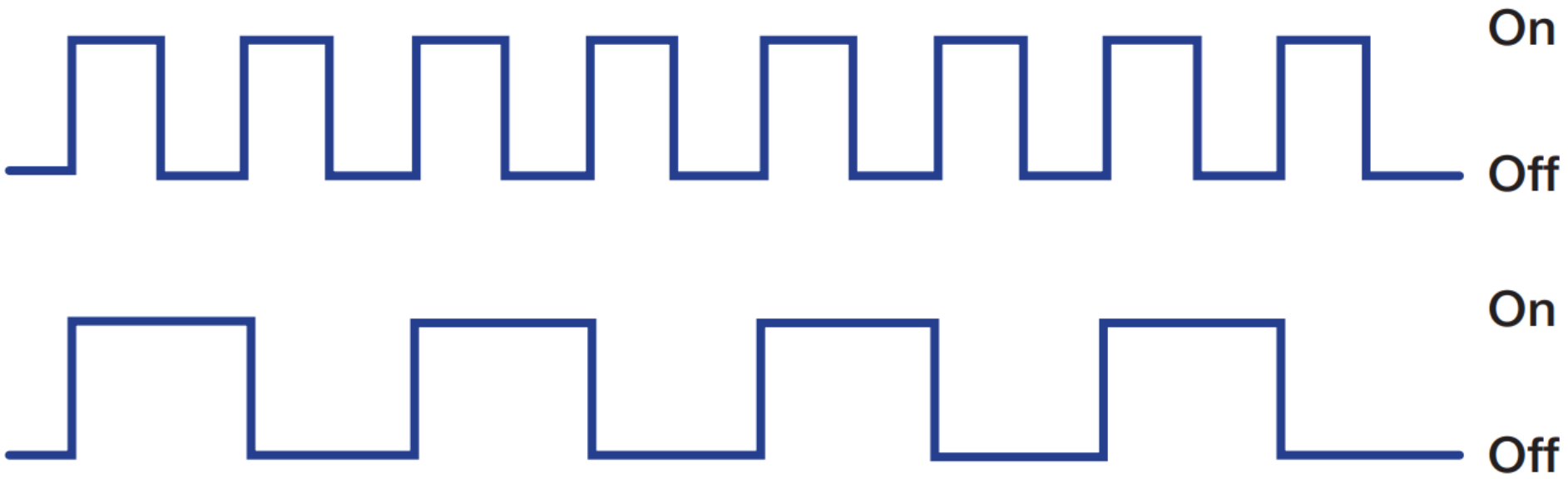
▲ 工作週期 50%

▲ 工作週期 90%

NEXT

LAB09 自製警報器

頻率



▲ 兩者工作週期皆為 50%, 輸出電壓相同 , 但上面的輸出頻率是下面的 2 倍

NEXT

LAB09 自製警報器

匯入相關模組：

```
from machine import Pin, PWM # 匯入 Pin 和 PWM
```

建立物件、設定頻率 (freq) 和工作週期 (duty)：

```
buzzer = PWM(Pin(23, Pin.OUT))
buzzer.freq(0)
buzzer.duty(512)
```

可以合併成一行

```
buzzer = PWM(Pin(23, Pin.OUT), freq=0, duty=512)
```

NEXT

LAB09 自製警報器

音符	中音 C	D	E	F	G	A	B
唱名	Do	Re	Mi	Fa	Sol	La	Ti
頻率 (Hz)	261	294	330	349	392	440	494

NEXT

LAB09 自製警報器

程式設計

```
01  from machine import Pin, PWM
02  import time
03
04  # 建立 PWM 物件
05  buzzer = PWM(Pin(23, Pin.OUT), freq=0, duty=512)
06
07  buzzer.freq(349)    # 發出 Fa 聲
08  time.sleep(1)
09  buzzer.freq(294)    # 發出 Re 聲
10  time.sleep(1)
11
12  buzzer.deinit()
```

NEXT

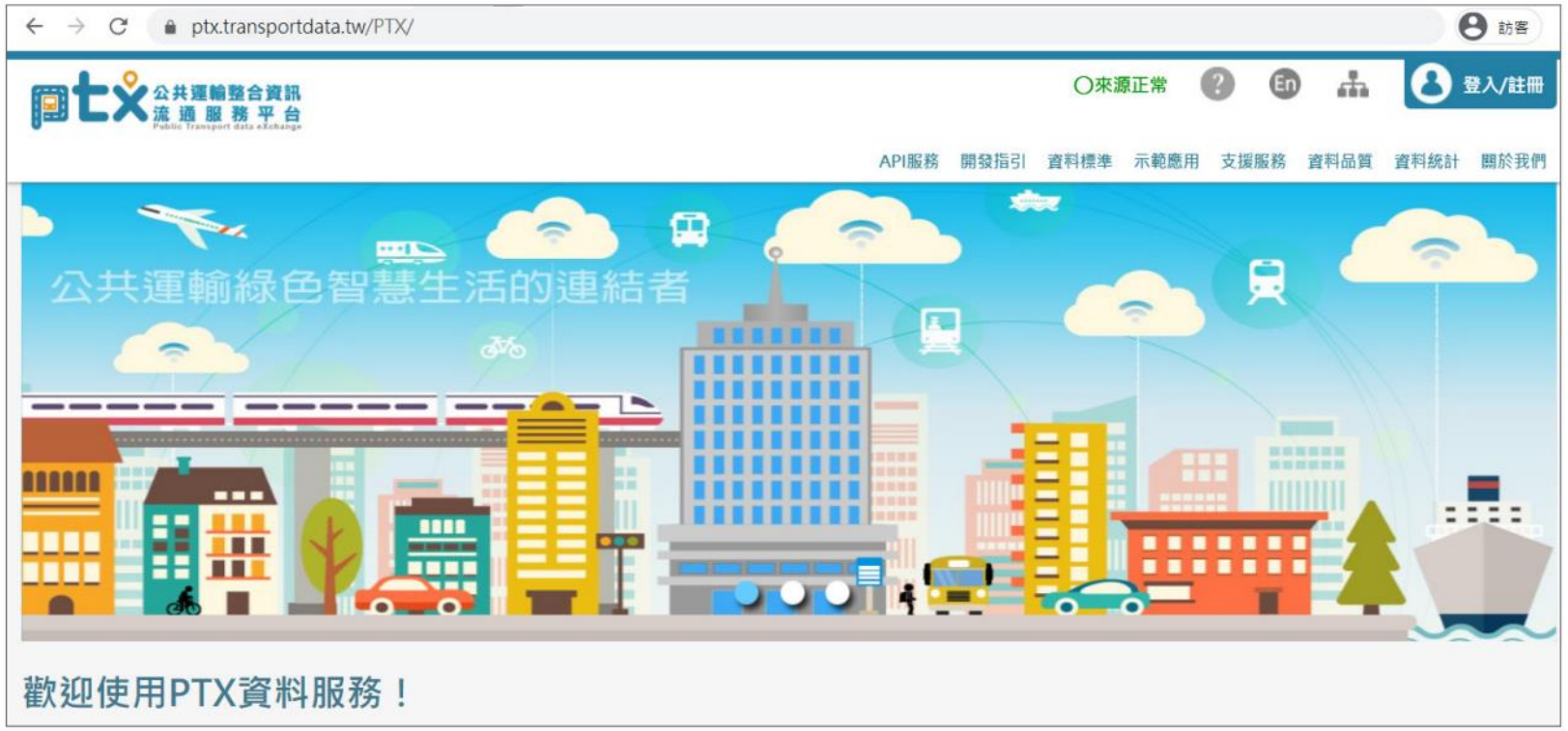
LAB09 自製警報器

實測

請按 F5 執行程式, 蜂鳴器就會發出 Fa 聲 , 1 秒後發出 Re 聲 , 再過 1 秒後結束。

5-4 PTX 服務平台

網址列輸入：**https://ptx.transportdata.tw/PTX/**



5-4 PTX 服務平台

API 服務　應用程式介面(Application Programming Interface)

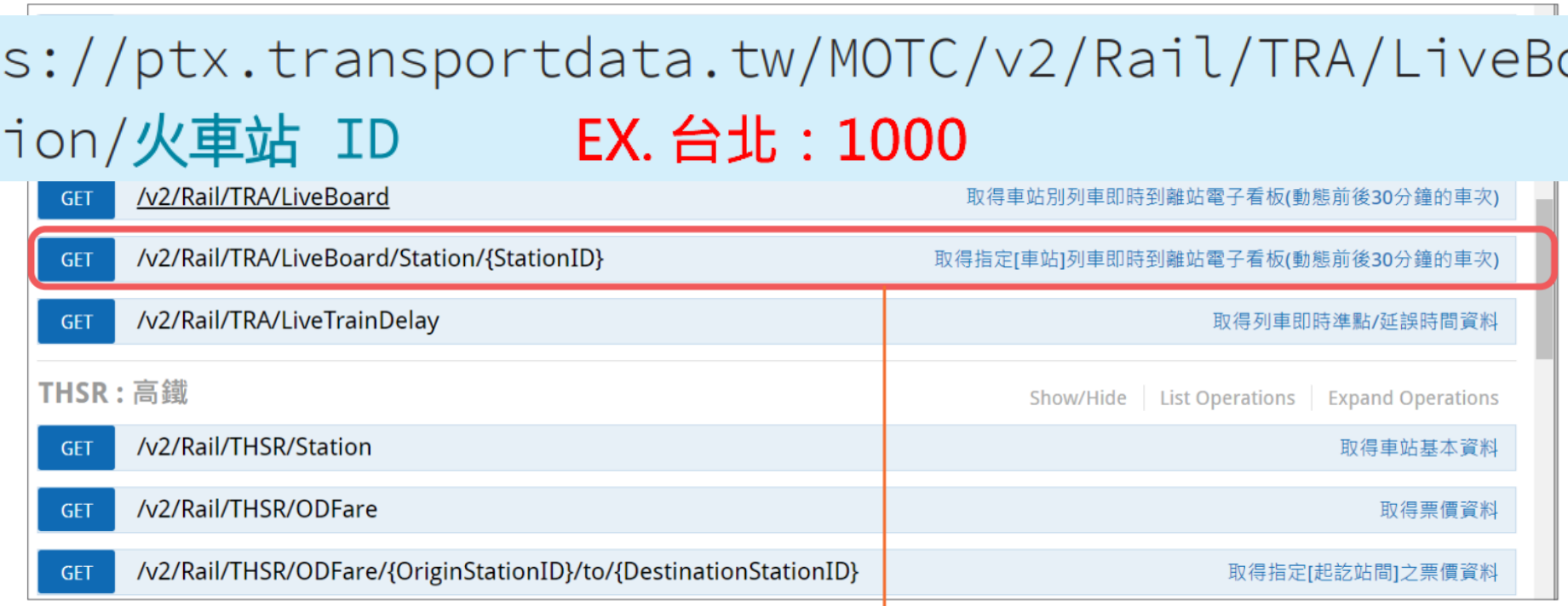
NEXT

5-4 PTX 服務平台

NEXT

5-4 PTX 服務平台

https://ptx.transportdata.tw/MOTC/v2/Rail/TRA/LiveBoard/Station/火車站 ID　　EX. 台北：1000



這個就是可查詢誤點時間的 API，主要功能為**查看指定車站的電子看板**，當中就包含誤點時間。

5-4 PTX 服務平台

參數

Parameters

Parameter	Value	Description	Parameter Type	Data Type
StationID	(required)	車站代碼	path	string
$select		挑選	query	string
$filter		過濾	query	string
$orderby		排序	query	string
$top	30	取前幾筆	query	integer
$skip		跳過前幾筆	query	string
$format	JSON	指定來源格式	query	string

NEXT

5-4 PTX 服務平台

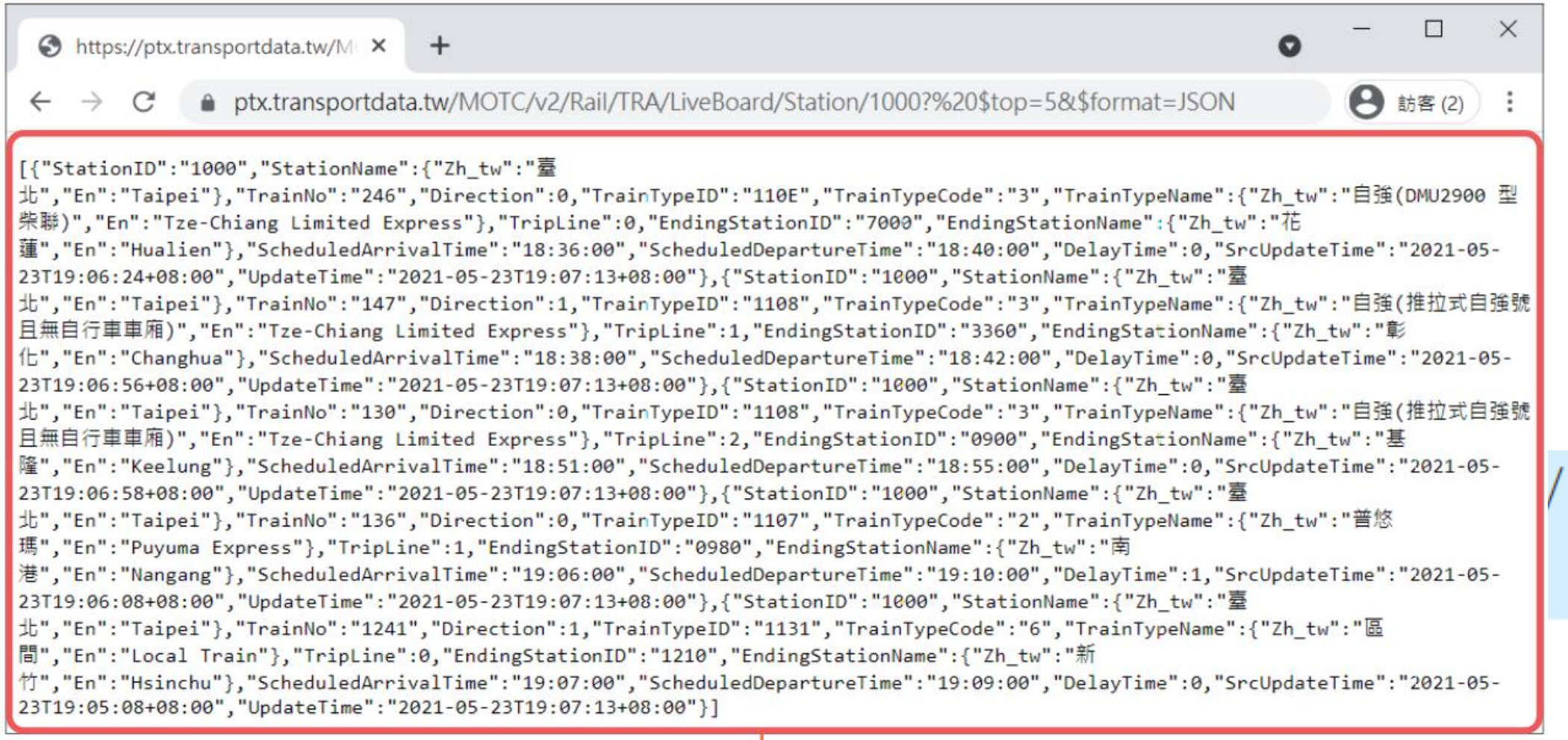
使用 '?' 將參數接在網址後面, 參數與參數名稱用 '=' 分開

```
https://ptx.transportdata.tw/MOTC/v2/Rail/TRA/LiveBoard/Station/火車站 ID?$top=5
```

加入多個參數使用 & 連接

```
前面省略?$top=5&$format=JSON
```

⚠ format 代表傳回資料的格式。

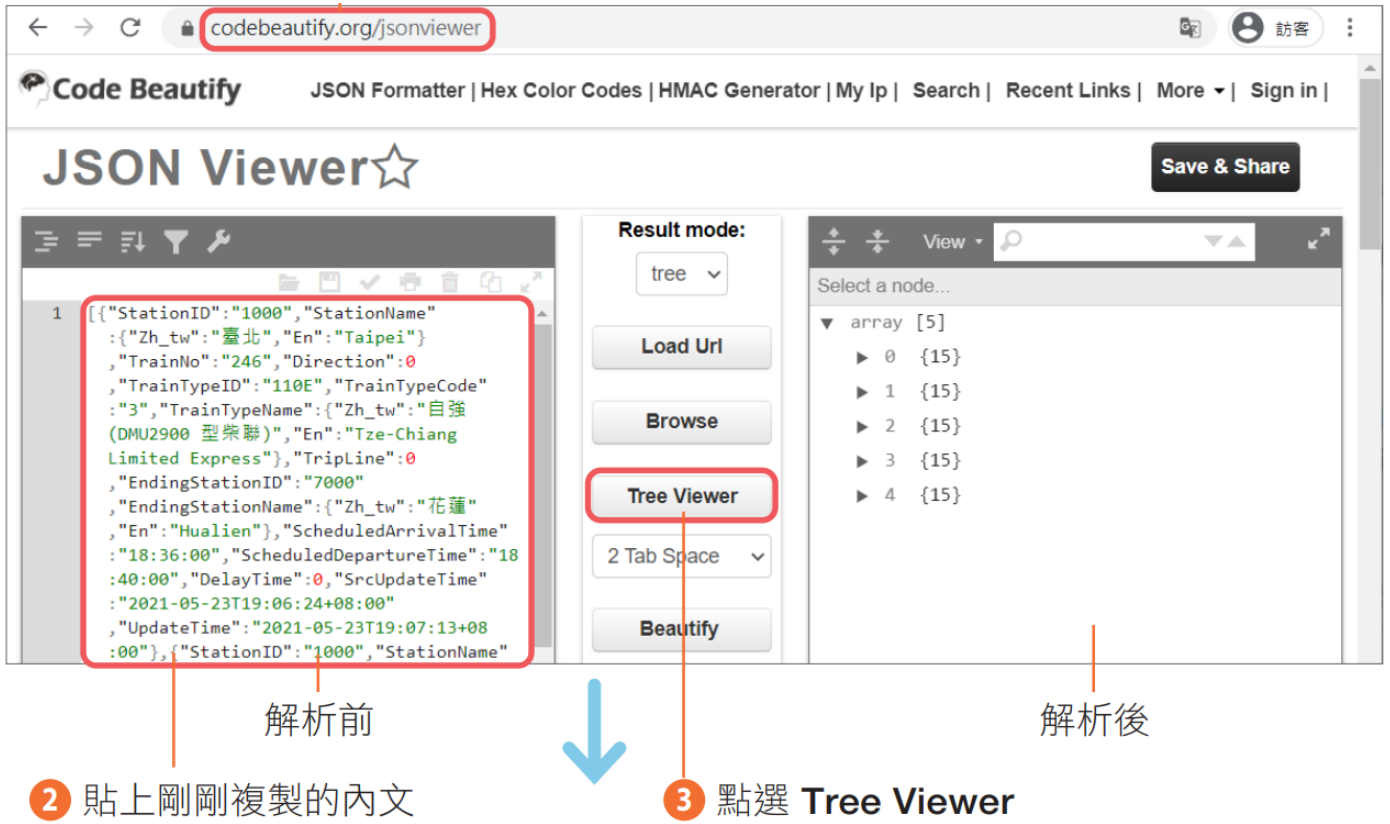


將此段內文複製

NEXT

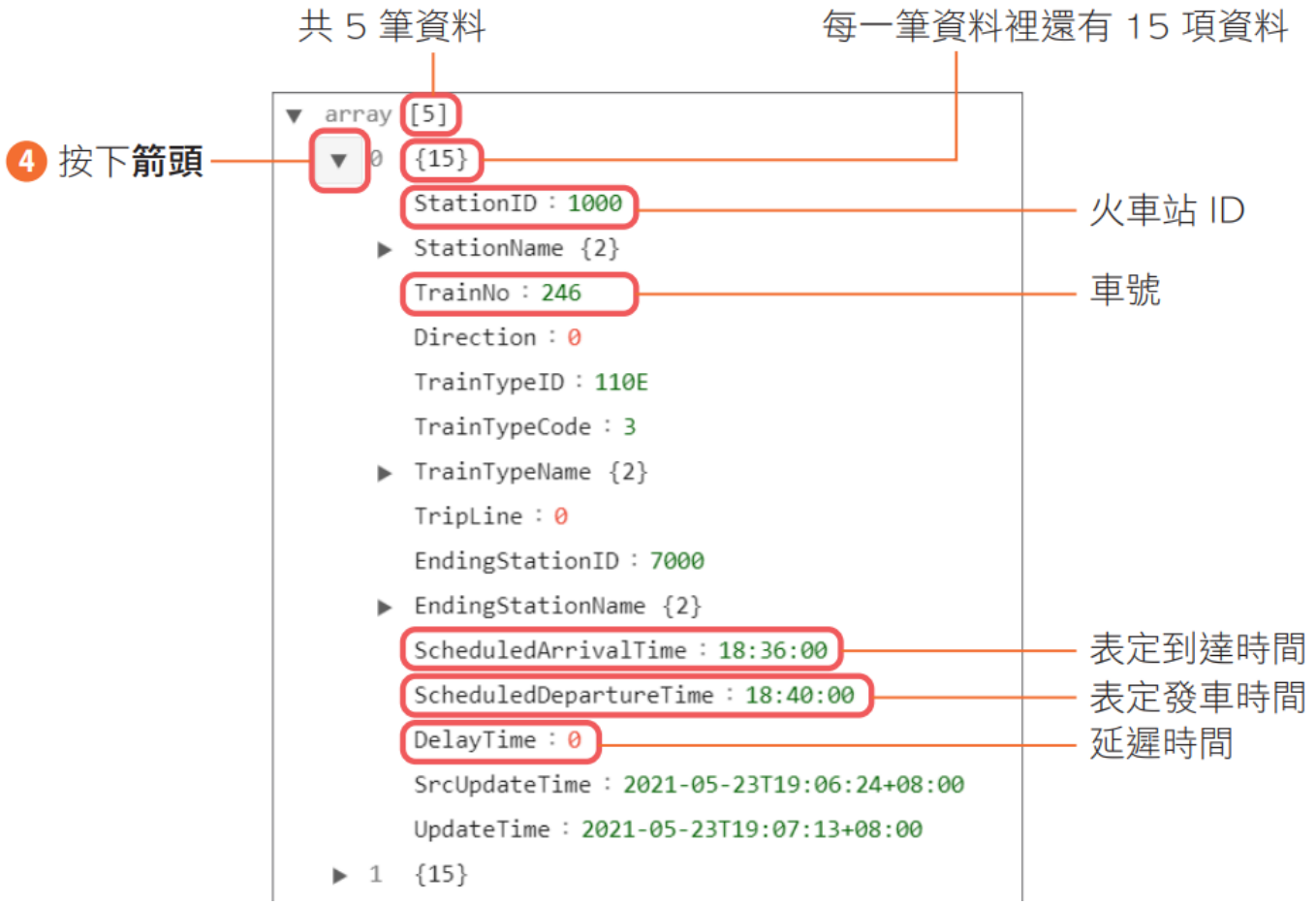
5-4 PTX 服務平台

網址列輸入：https://codebeautify.org/jsonviewer



NEXT

5-4 PTX 服務平台



中括號：陣列

大括號：物件

5-4 PTX 服務平台

使用程式發出請求並解讀 JSON 資料

```
>>>  import urequests
>>>  headers = {'user-agent':'curl/7.76.1'}
>>>  res = urequests.get("https://ptx.transportdata.tw/MOTC/
v2/Rail/...", headers=headers)
```

增加表頭來偽裝成瀏覽器：

```
>>>  headers = {'user-agent':'curl/7.76.1'}
```

5-

使用

>>>

>>>

0

目的資料

串列

```
>>> a = [16, 14, 12, 13, 15, 5, 4] ◄── 以中括號表示串列
>>> a[0]◄── 取得第一個元素(從 0 起算)
16
>>> a[1]
14
```

字典

```
>>> ages = {"Mary":13, "John":14}
>>> ages["Mary"]
13
>>> ages["John"]
14
```

LAB10 火車誤點提醒器

實驗目的	透過 PTX 服務平台查詢火車誤點時間，如果有查詢到對應的火車，會發出警報聲，並將延遲時間顯示在七段顯示器上，再經由 IFTTT 傳送延遲時間到 LINE。
材料	同 LAB09

接線圖

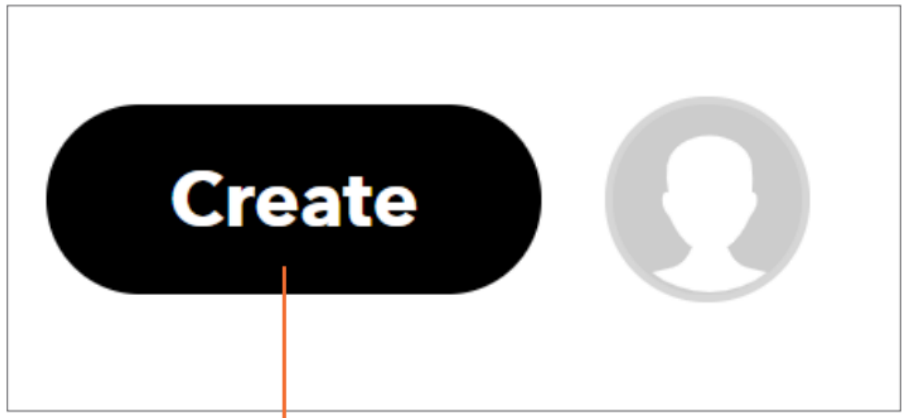
同 LAB09

NEXT

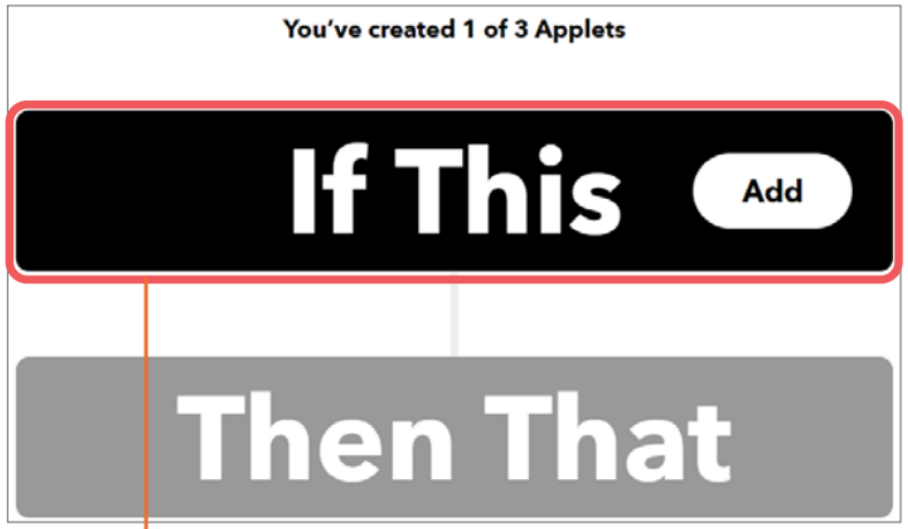
LAB10 火車誤點提醒器

設計原理

使用 IFTTT 傳送 LINE 訊息



1 按頁面右上角的 **Create**

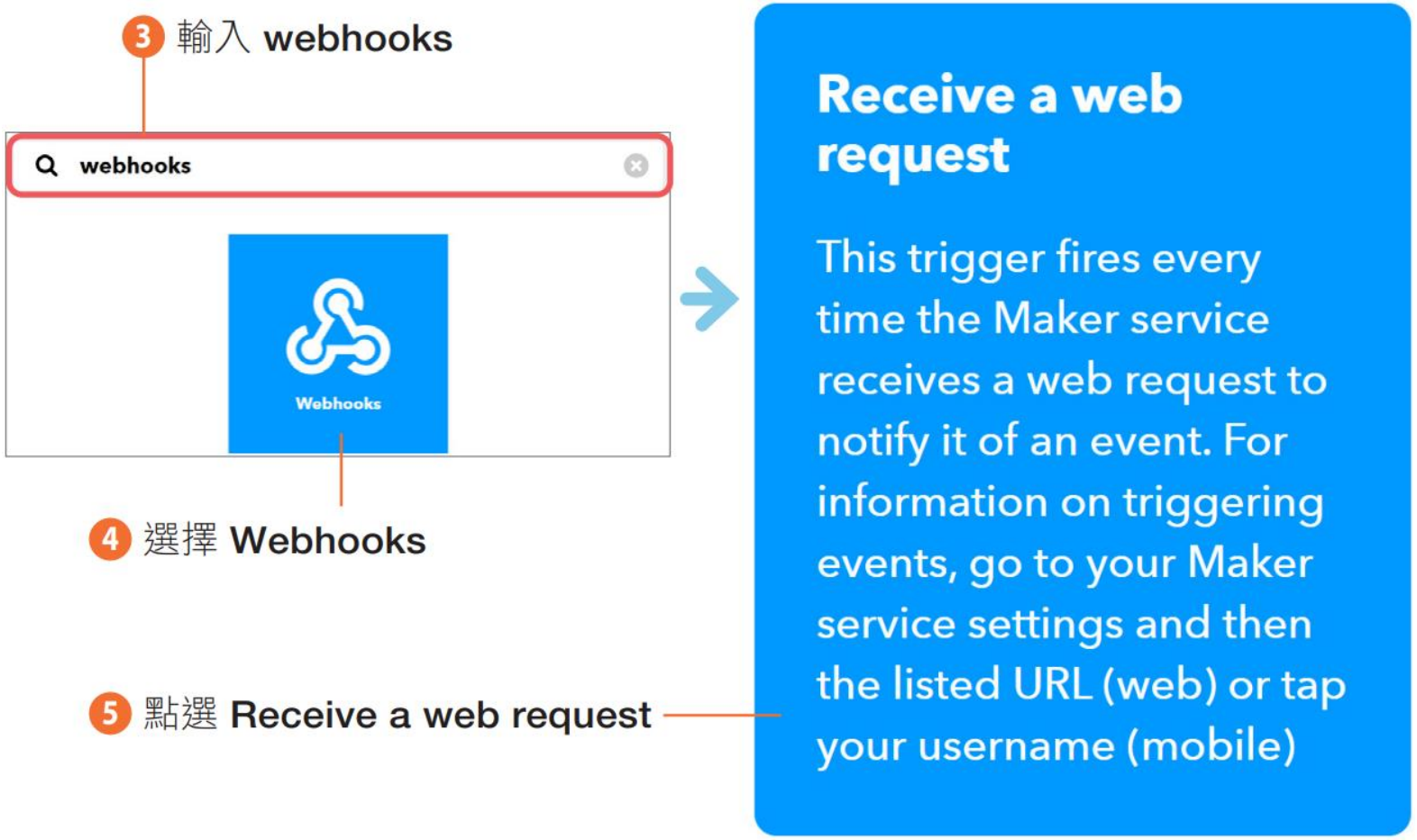


2 按 **If This**

跳至 IFTTT

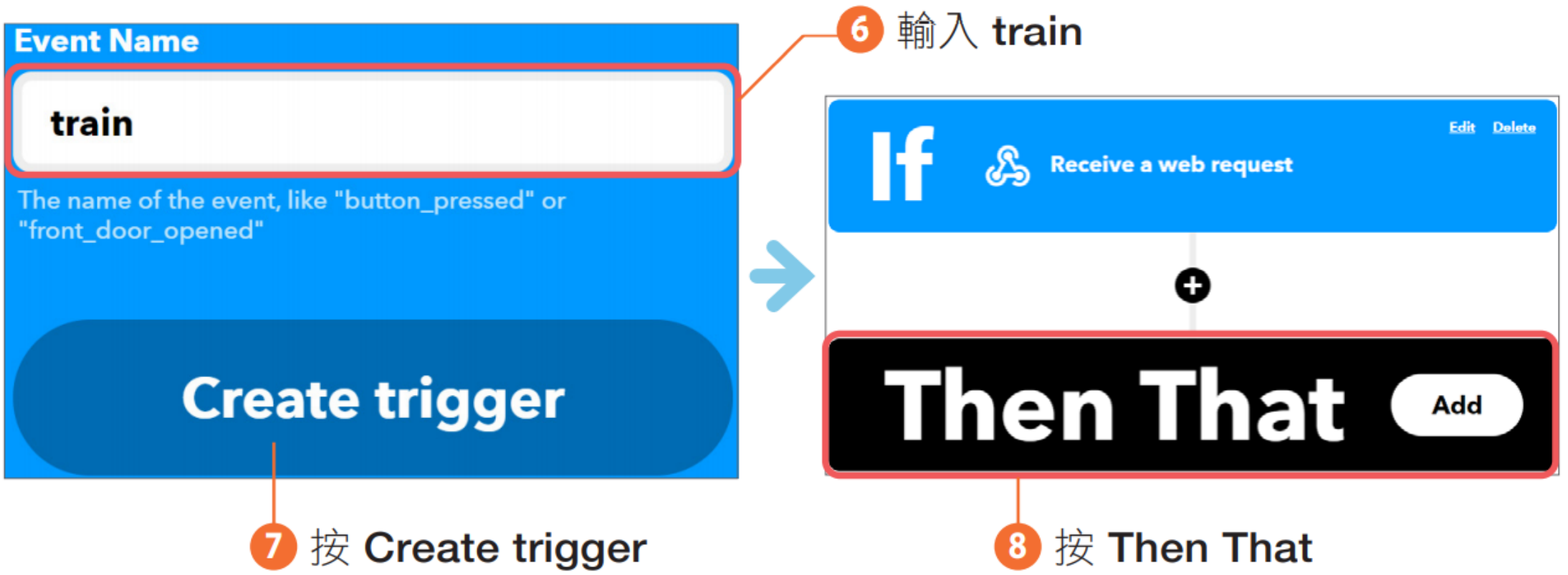
NEXT

LAB10 火車誤點提醒器



NEXT

LAB10 火車誤點提醒器



NEXT

LAB10 火車誤點提醒器

9 輸入 line

10 選擇 LINE

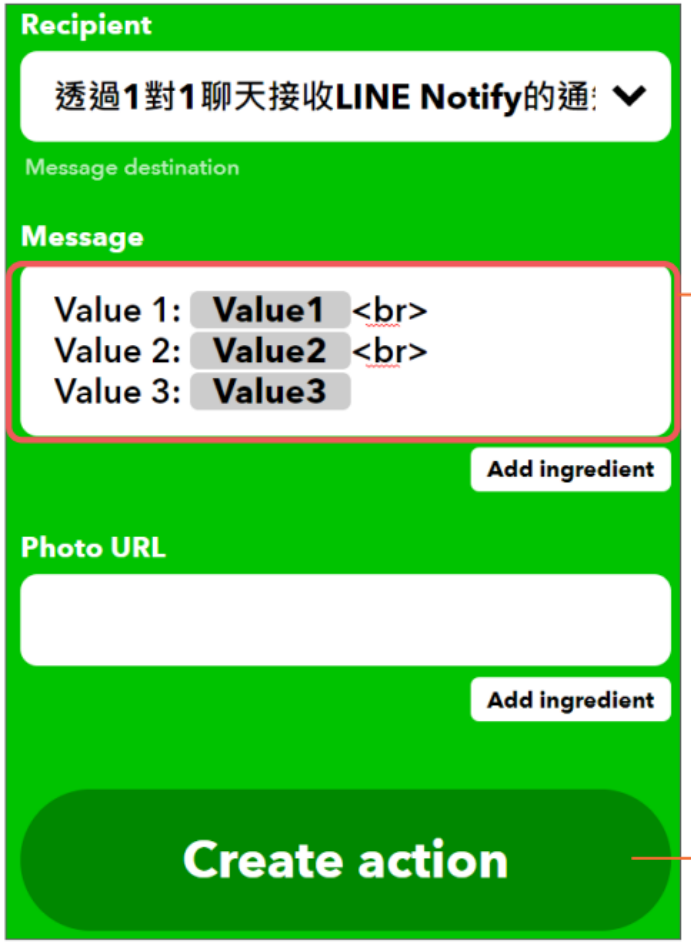
Send message

This Action will post a message to LINE.

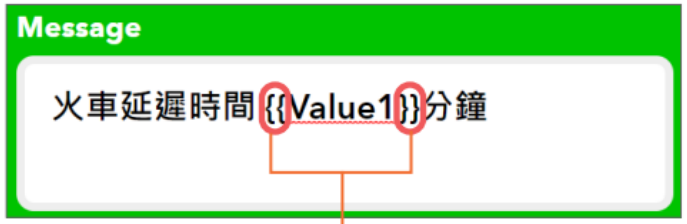
11 點選 Send meaasge

NEXT

LAB10 火車誤點提醒器



12 更改 Message 內容

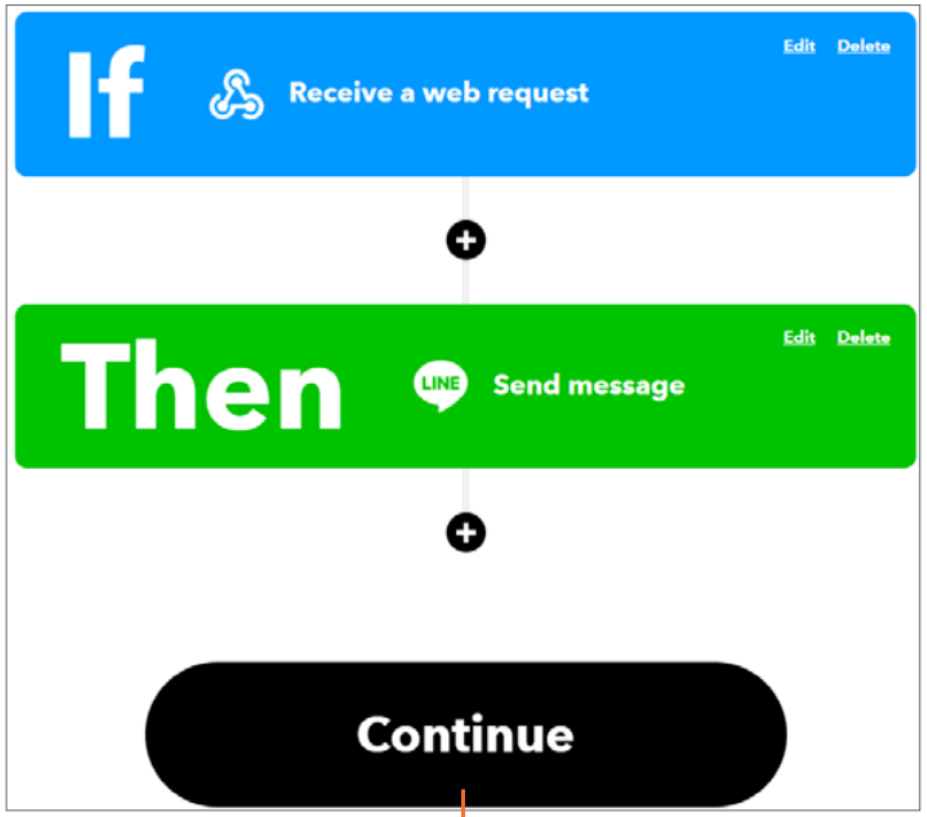


Value1 旁各有兩個大括號

13 按 Create action

NEXT

LAB10 火車誤點提醒器



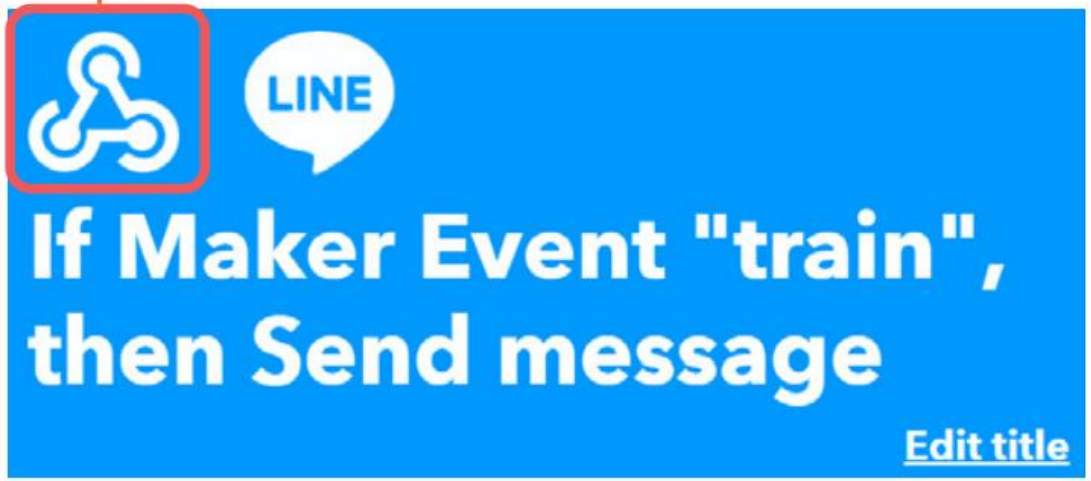
14 按 Continue

15 按 Finish

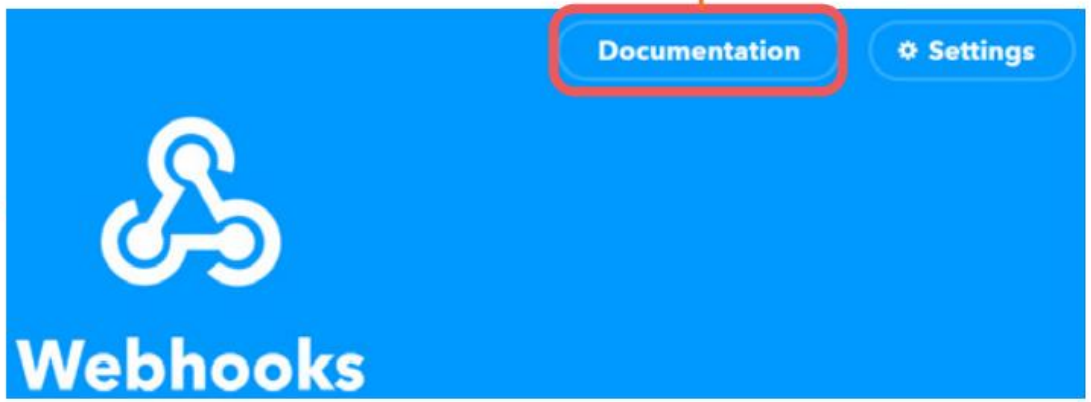
NEXT

LAB10 火車誤點提醒器

16 點選 Webhooks 圖示



17 按 Documentation



NEXT

LAB10 火車誤點提醒器

NEXT

LAB10 火車誤點提醒器

程式設計

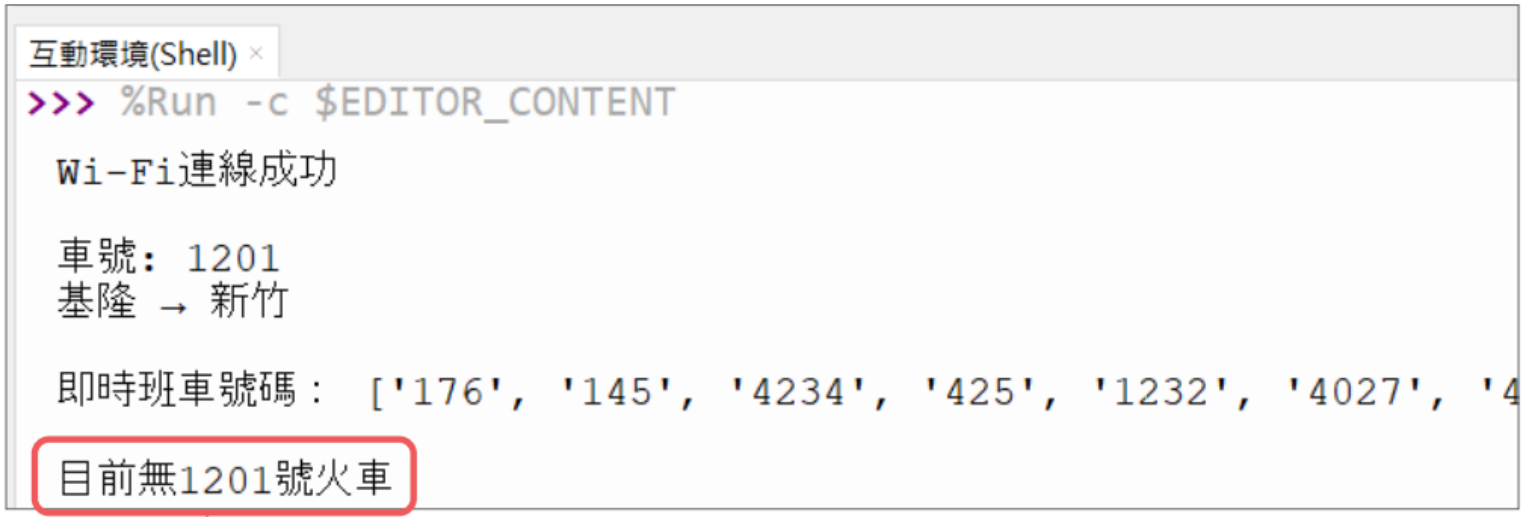
```
01  # 無會員：當天次數 50 次
02  from machine import Pin,PWM
03  import network
04  import urequests
05  import time
06  import tm1637
07  import ntptime
08
09  # 四位數顯示器
10  tm = tm1637.TM1637(clk=Pin(16), dio=Pin(17))
11  # 清空四位數顯示器
12  tm.write([0, 0, 0, 0])
13
14  # 連線至無線網路
15  sta=network.WLAN(network.STA_IF)
16  sta.active(True)
17  sta.connect('無線網路名稱','無線網路密碼')
18  while not sta.isconnected() :
19      pass
```

NEXT

LAB10 火車誤點提醒器

實測

電子看板沒有指定的車號



顯示沒有指定車號的資訊

NEXT

LAB10 火車誤點提醒器

電子看板有指定的車號

互動環境(Shell)

```
>>> %Run -c $EDITOR_CONTENT

  Wi-Fi連線成功

  車號: 4027
  蘇澳新 → 湖口

  即時班車號碼:  ['176', '425', '1232', '4027', '438', '4189', '1

  表定發車時間: 18:25:00
  延遲時間: 0 分鐘
```

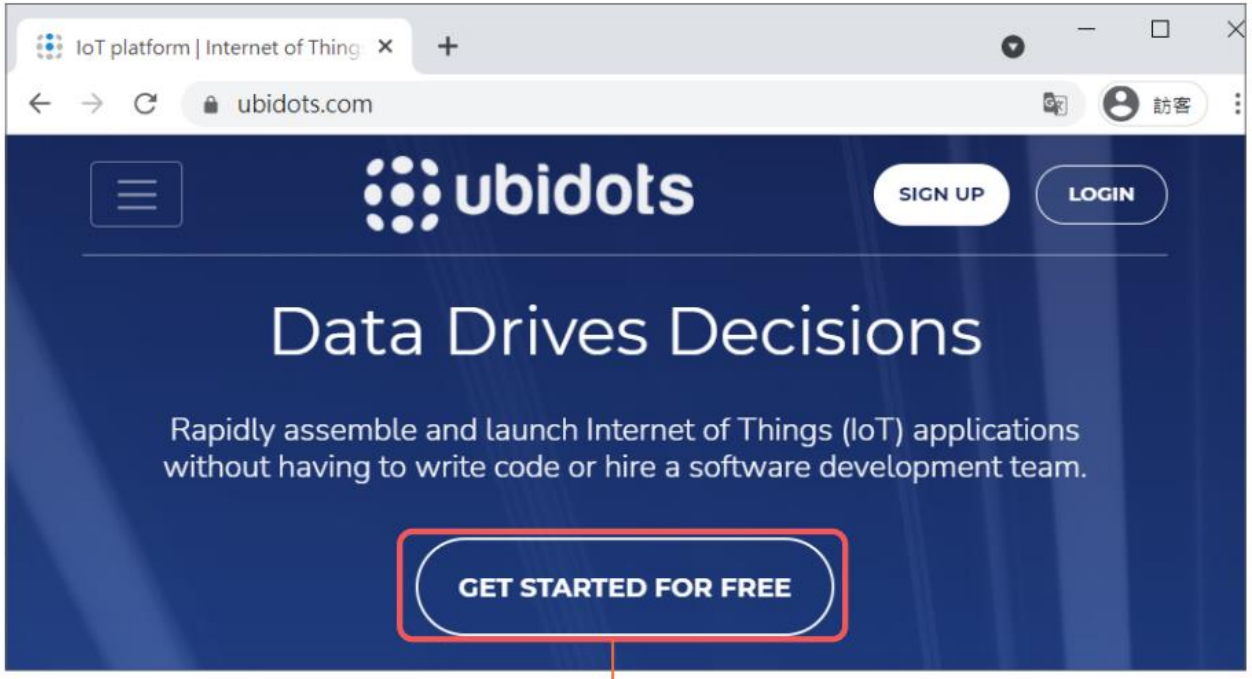
延遲時間

NEXT

目錄

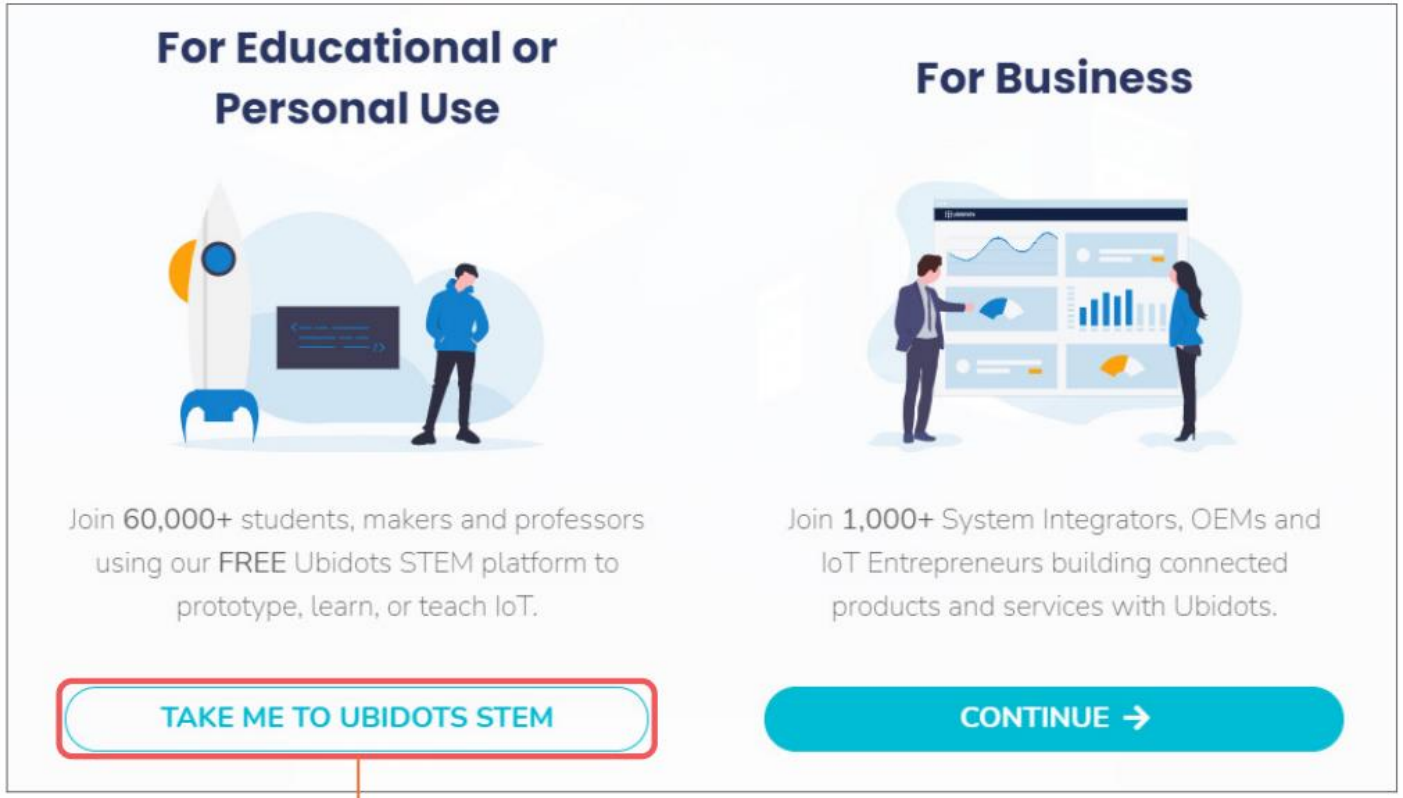
6-1 Ubidots

網址列輸入：https://ubidots.com/



NEXT

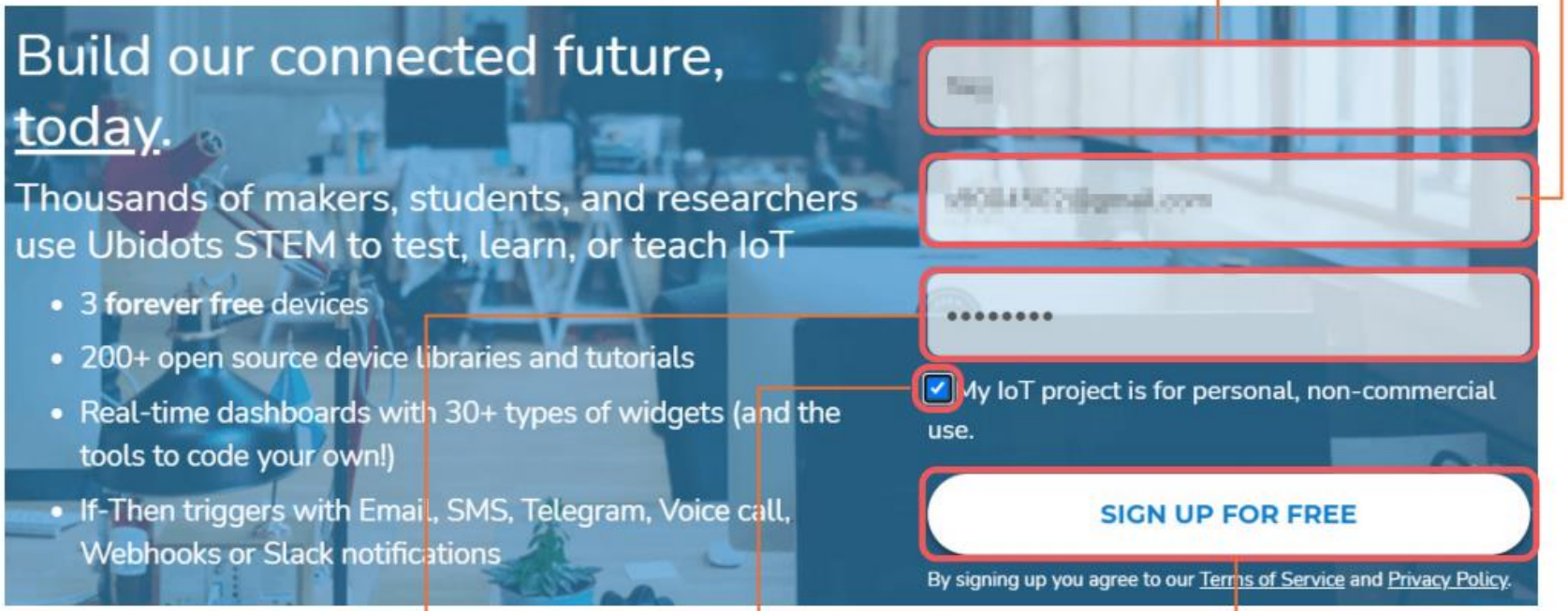
6-1 Ubidots



2 點選 TAKE ME TO UBIDOTS STEM

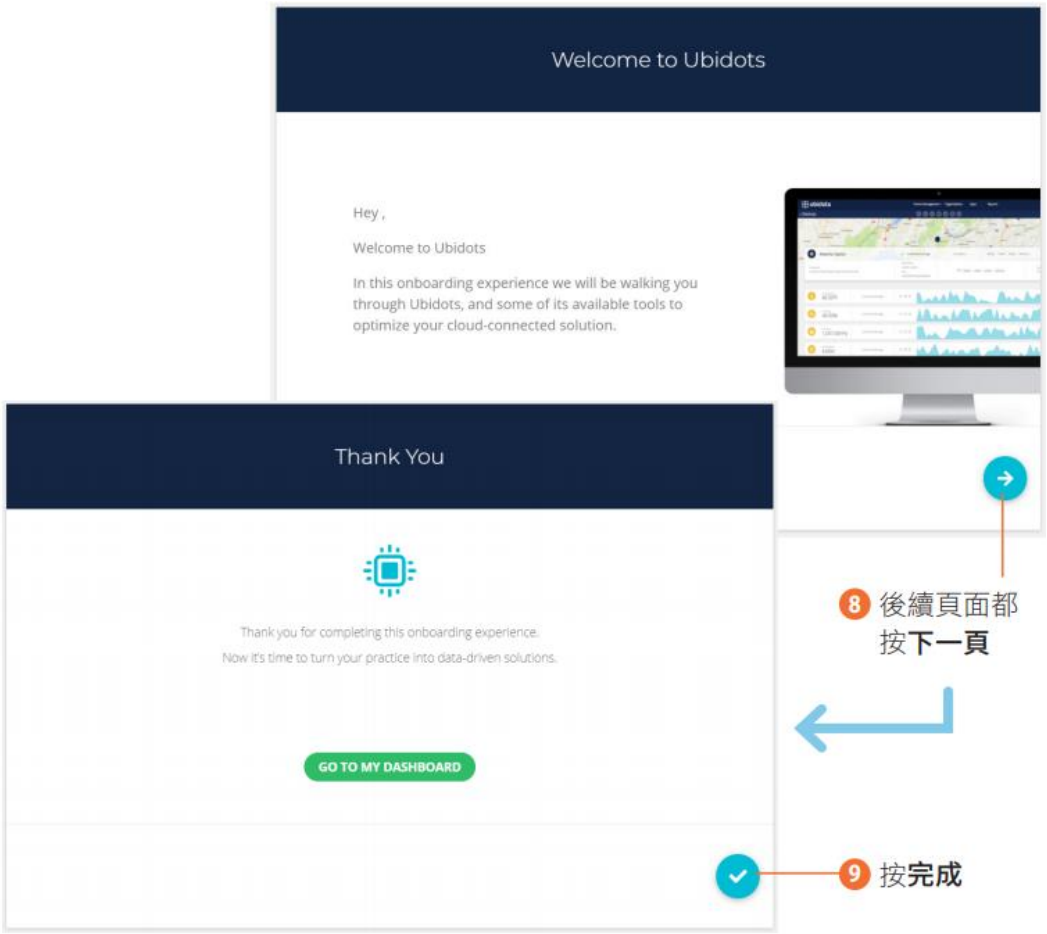
NEXT

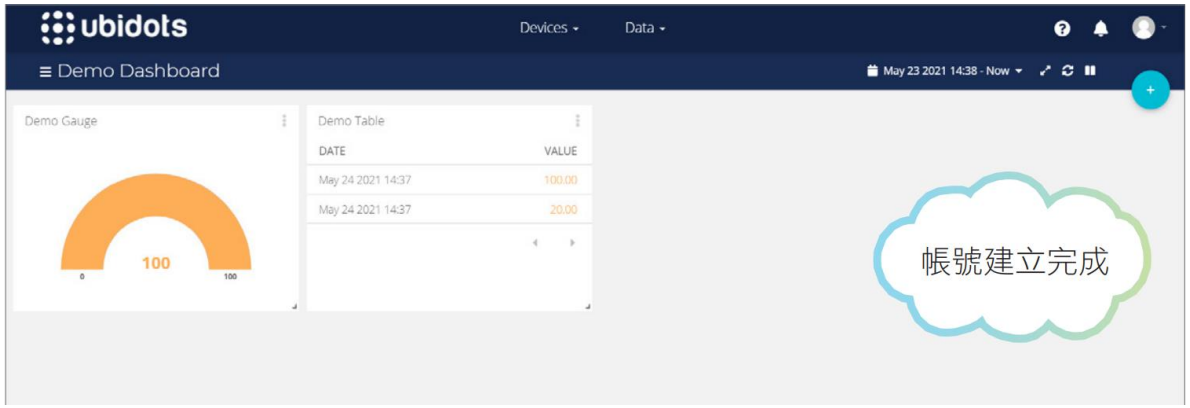
6-1 Ubidots



NEXT

6-1 Ubidots


Welcome to Ubidots
Hey ,
Welcome to Ubidots
In this onboarding experience we will be walking you through Ubidots, and some of its available tools to optimize your cloud-connected solution.
Thank You
Thank you for completing this onboarding experience.
Now it's time to turn your practice into data-driven solutions.
GO TO MY DASHBOARD
8 後續頁面都按下一頁
9 按完成


ubidots
Devices
Data
Demo Dashboard
May 23 2021 14:38 - Now
Demo Gauge
100
Demo Table
DATE
VALUE
May 24 2021 14:37
100.00
May 24 2021 14:37
20.00
帳號建立完成

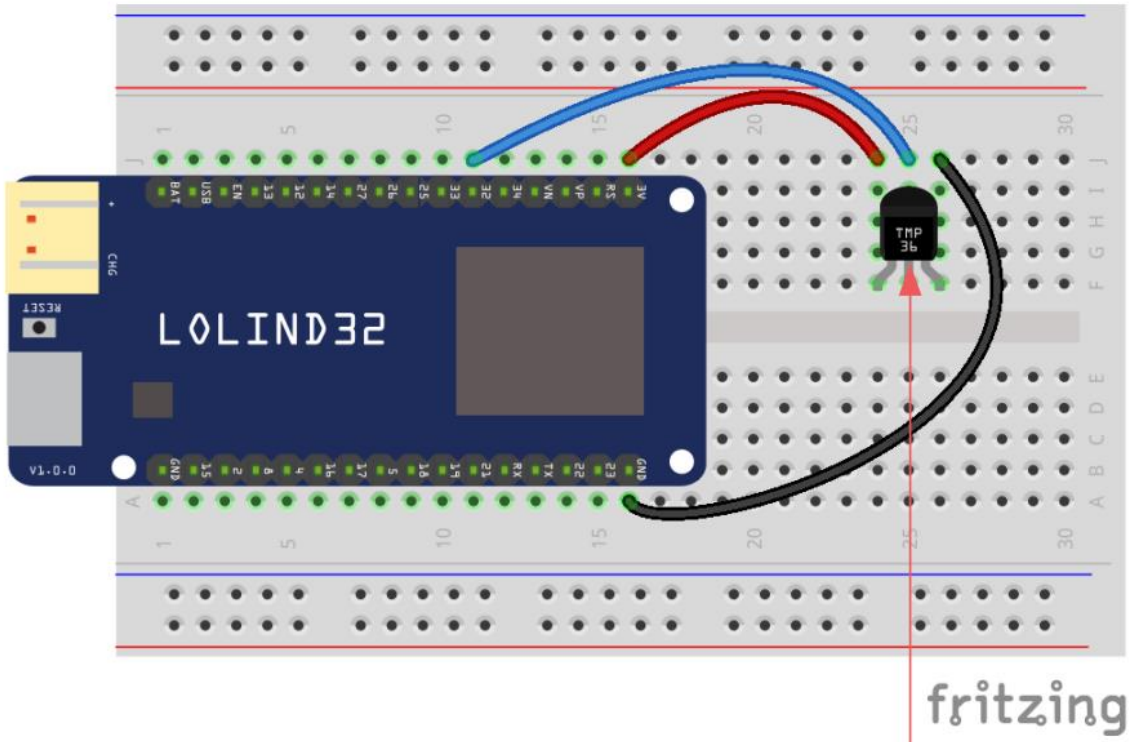
LAB11 雲端溫度紀錄儀

實驗目的	將溫度值傳送到 Ubidots 物聯網平台
材料	• ESP32 • TMP36 溫度感測器 • 杜邦線若干條 • 排針 • 麵包板 ⚠ 同 LAB04

NEXT

LAB11 雲端溫度紀錄儀

接線圖



有字的面朝下

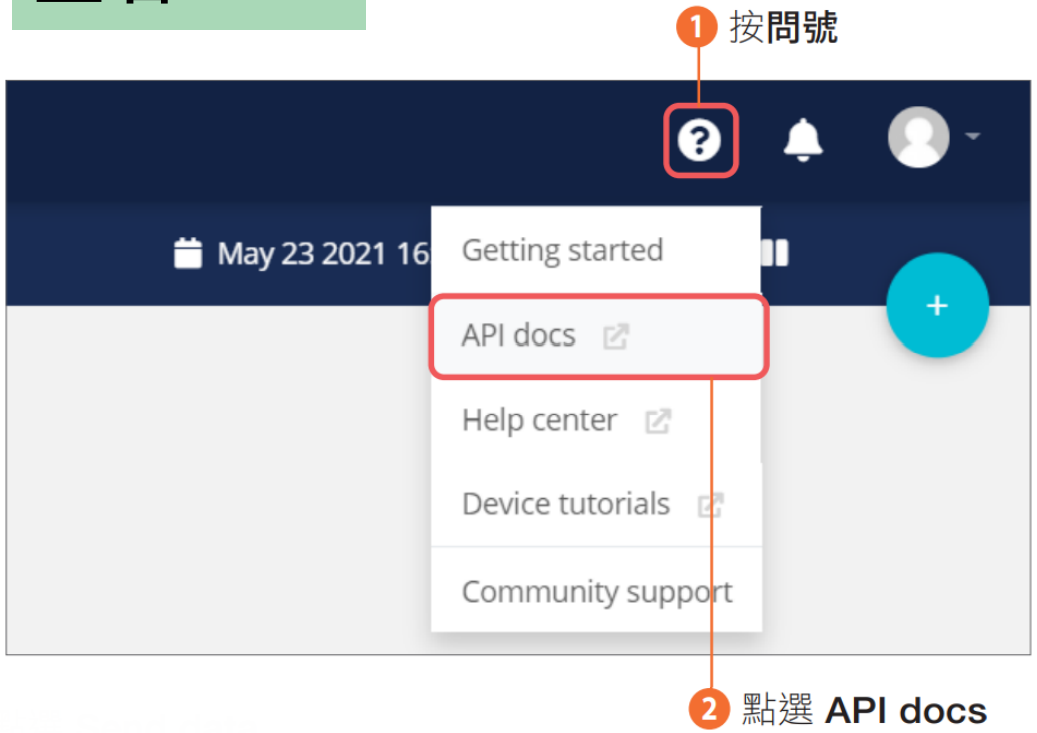
ESP32	TMP36 溫度感測器
3V	左邊腳
32	中間腳
GND	右邊腳

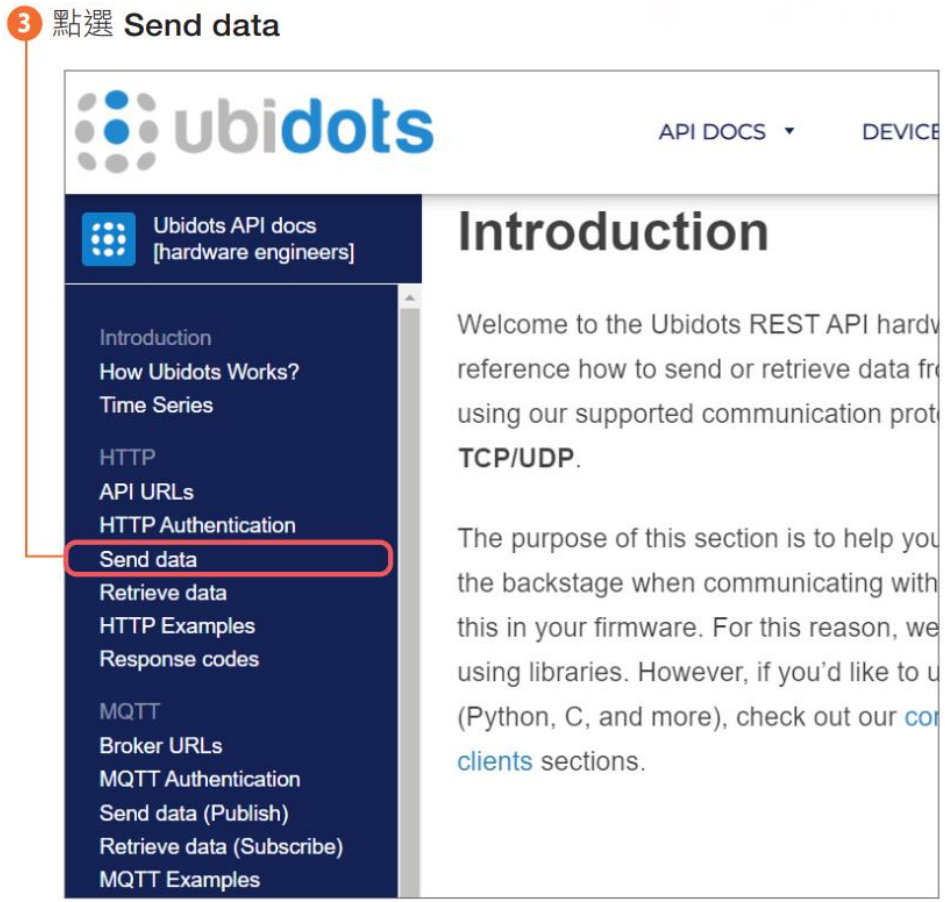
NEXT

LAB11 雲端溫度紀錄儀

設計原理

查看 API





NEXT

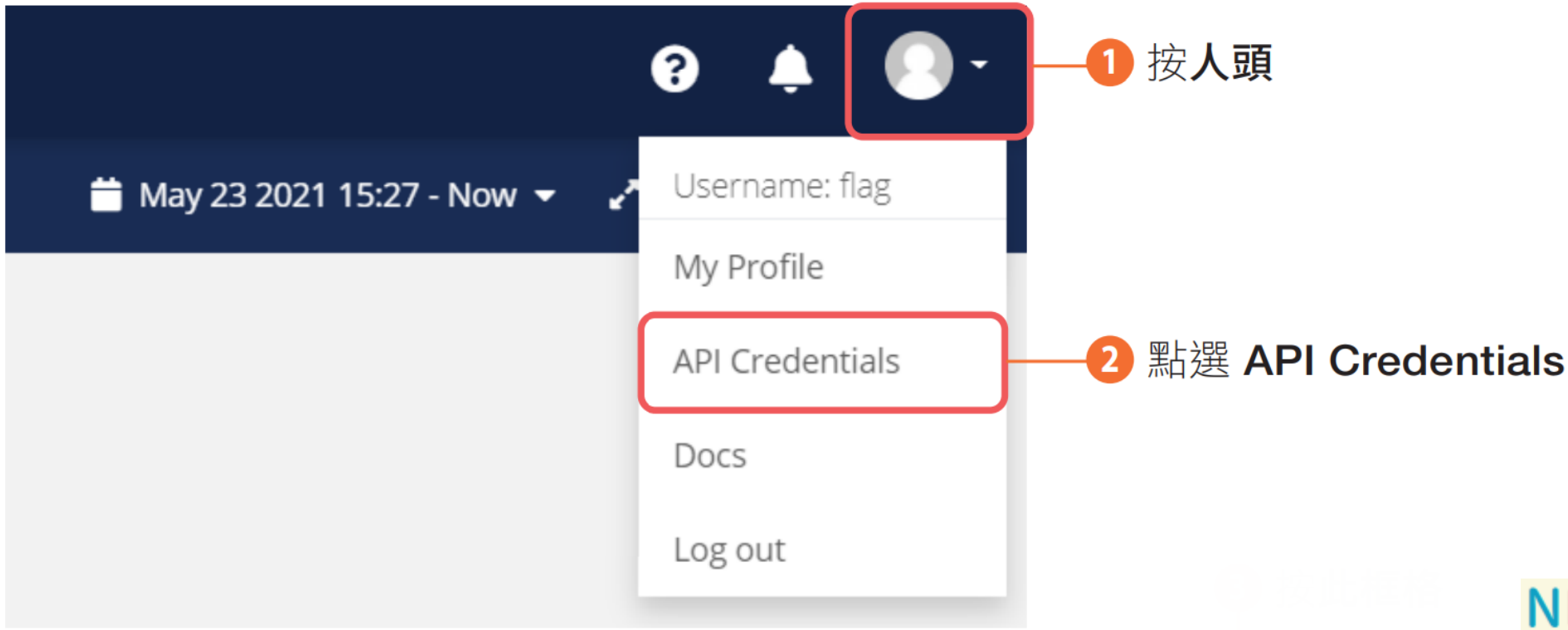
LAB11 雲端溫度紀錄儀

API 請求網址

上傳時需要的資料

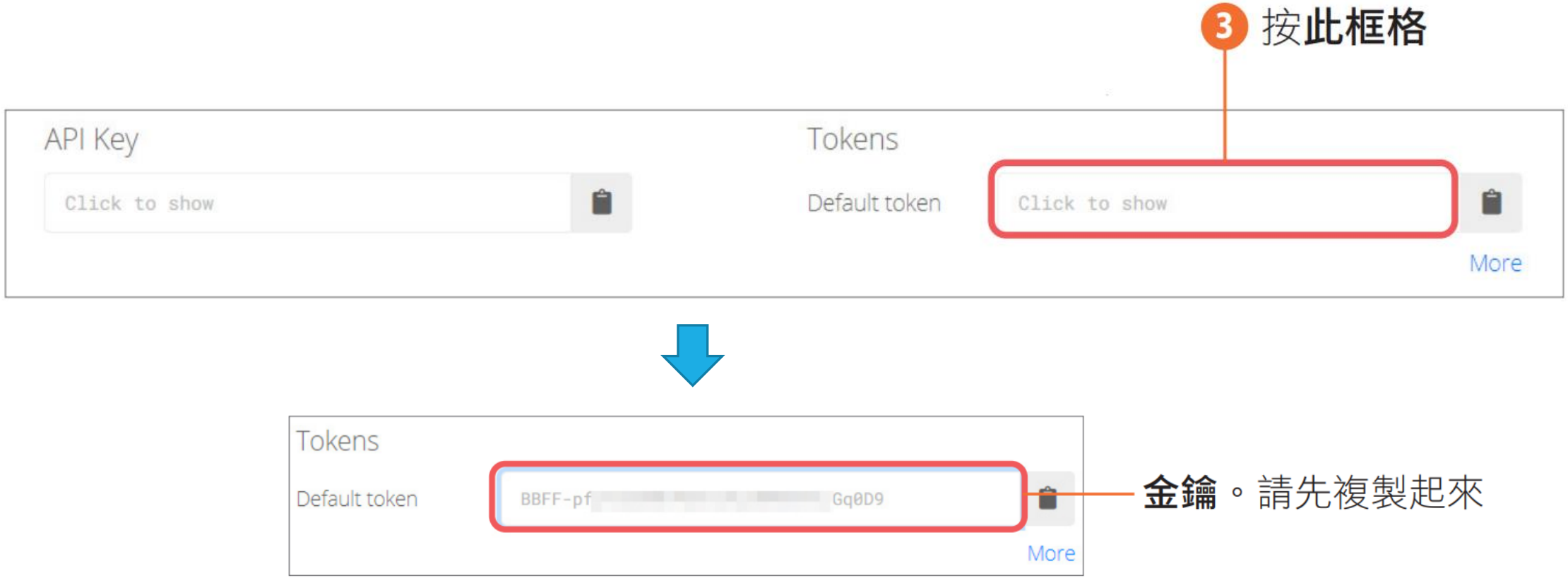
LAB11 雲端溫度紀錄儀

金鑰(X-Auth-Token)



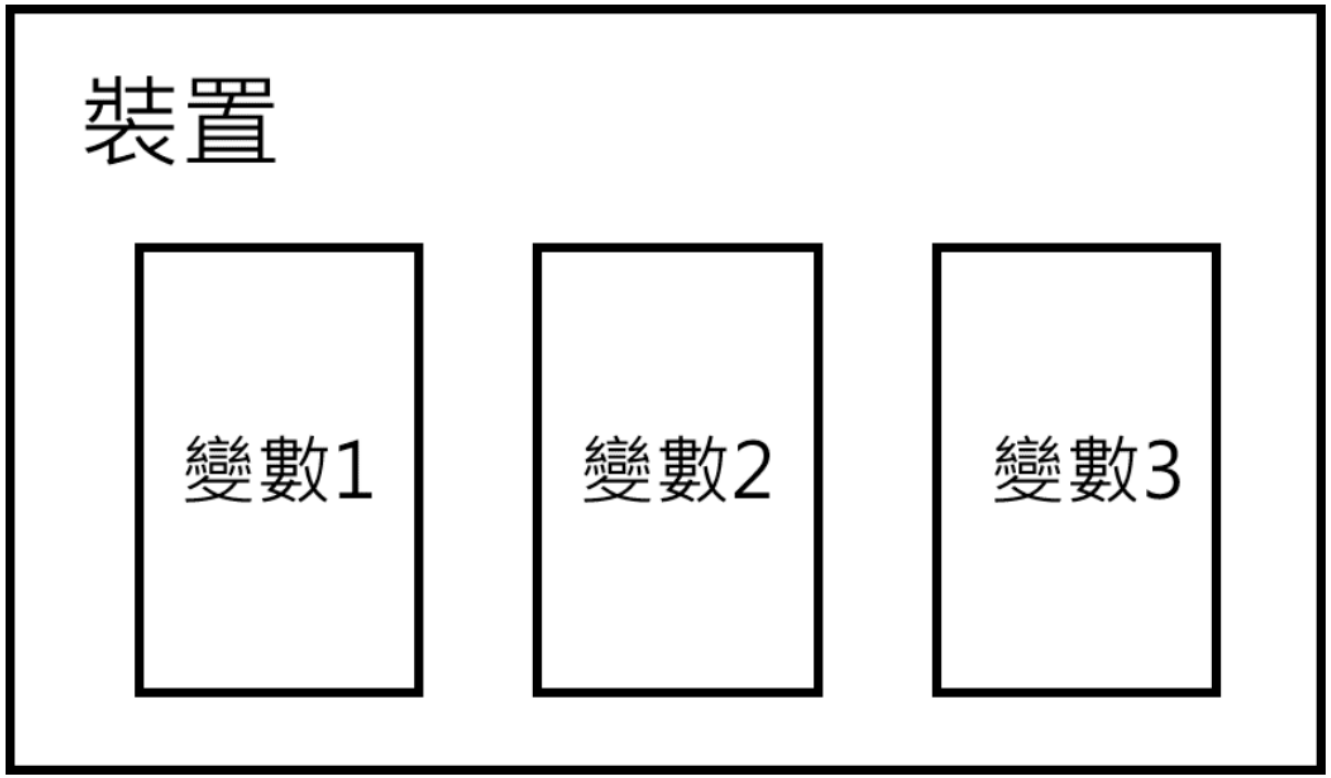
NEXT

LAB11 雲端溫度紀錄儀



LAB11 雲端溫度紀錄儀

裝置(device)、變數(variable)



⚠ 免費帳號最多 3 個裝置

⚠ 不用先建立**裝置**和**變數**

LAB11 雲端溫度紀錄儀

發出請求

```
>>>  import urequests
>>>  data = {"變數名稱":資料} ←以 Python 字典來設定要上傳的變數和資料
>>>  headers = {"X-Auth-Token":金鑰}
>>>  urequests.post("請求網址", json = data, headers = headers)
```

LAB11 雲端溫度紀錄儀

程式設計

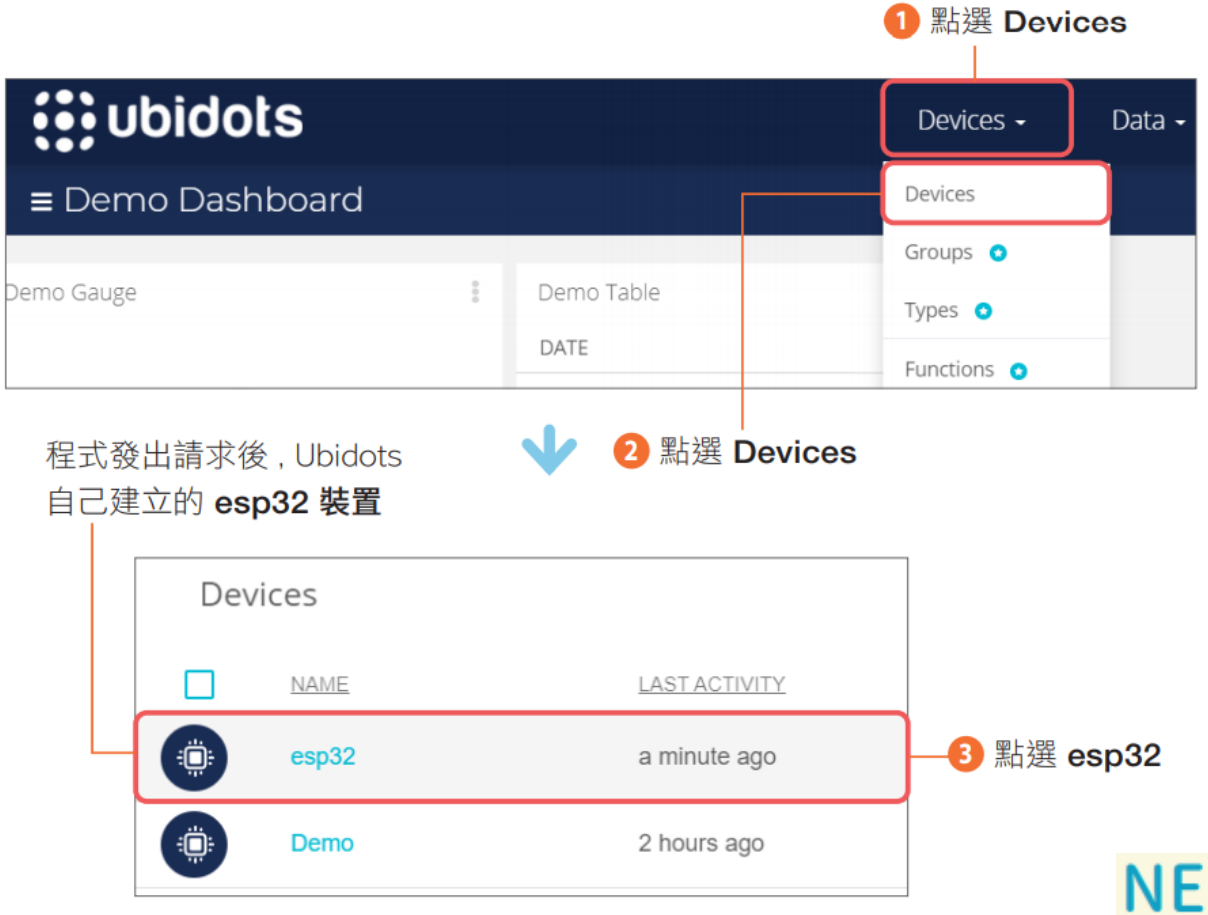
```
01  from machine import Pin, ADC
02  import time
03  import network
04  import urequests
05  import gc
06
07  adc_pin=Pin(32)
08  adc = ADC(adc_pin)
09  adc.width(ADC.WIDTH_12BIT)
10  adc.atten(ADC.ATTN_11DB)
11
12  # 連線至無線網路
13  sta=network.WLAN(network.STA_IF)
14  sta.active(True)
```

LAB11 雲端溫度紀錄儀

實測

互動環境(Shell)

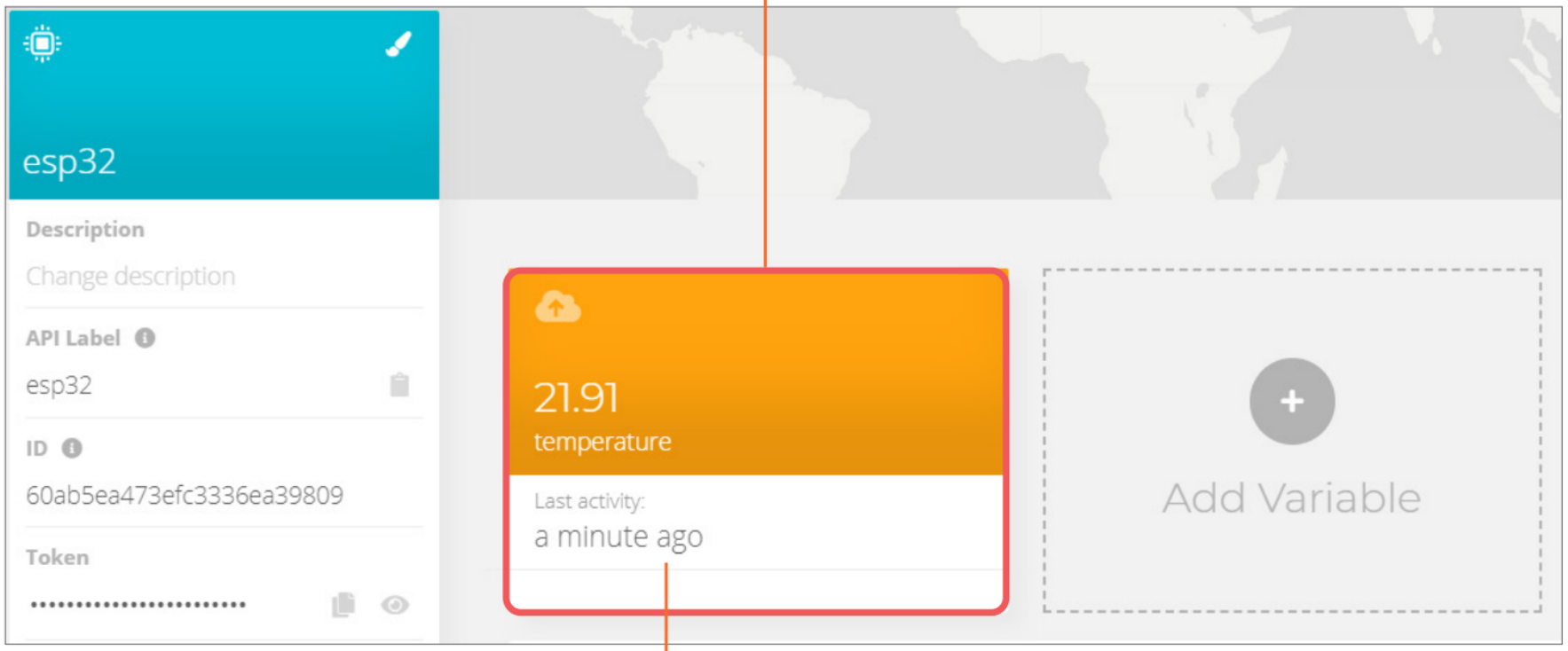
```
MicroPython v1.14 on 20
Type "help()" for more
>>> %Run -c $EDITOR_CON

Wi-Fi連線成功
目前溫度: 21.91209
傳送成功
```



NEXT

LAB11 雲端溫度紀錄儀

4 點選 **temperature**



程式發出請求後，Ubidots 自己建立的 **temperature 變數**

NEXT

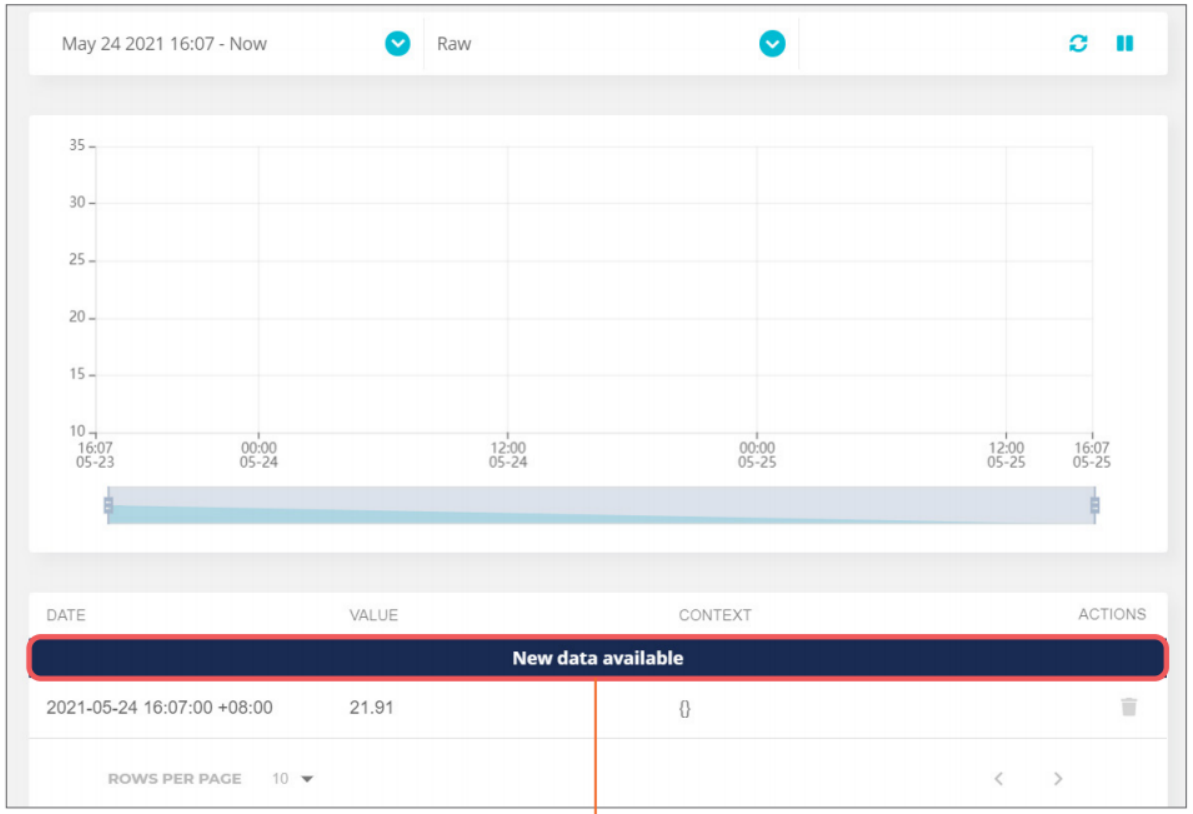
LAB11 雲端溫度紀錄儀

程式上傳的溫度

NEXT

LAB11 雲端溫度紀錄儀

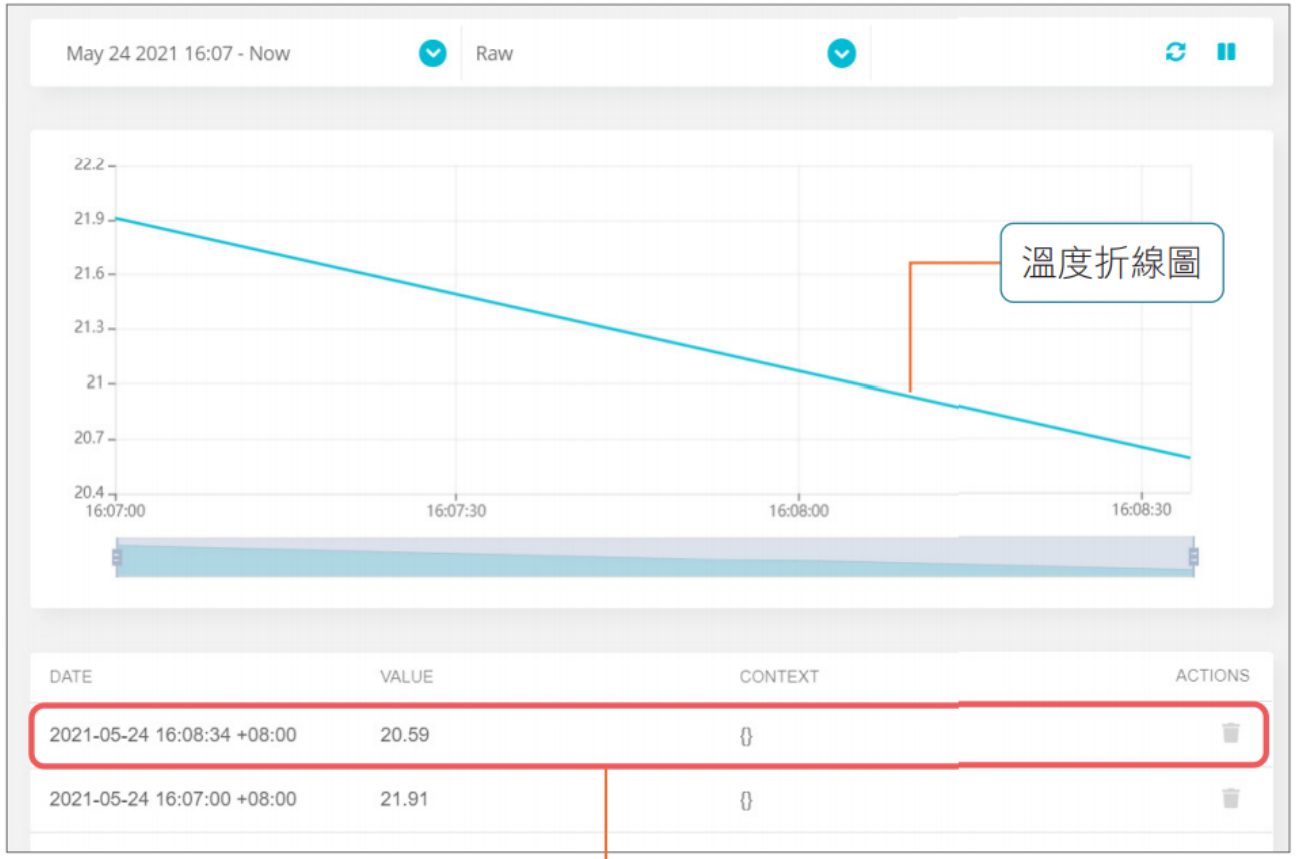
```
MicroPython v1.14 on 2021-02-02; ESP32 m
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  Wi-Fi連線成功
  目前溫度: 21.91209
  傳送成功
  目前溫度: 20.5934
  傳送成功
```



上傳成功後，會出現 **New data available** 按鈕，
點選它來更新資料

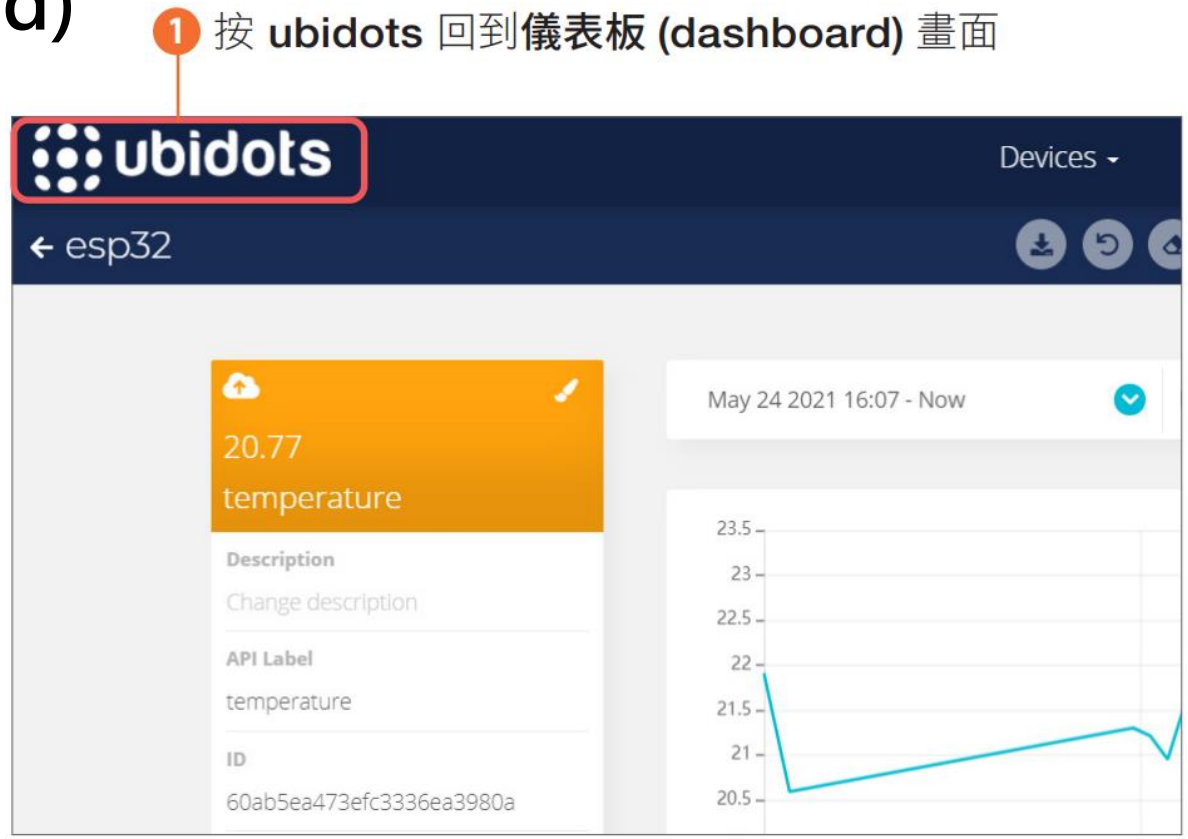
NEXT

LAB11 雲端溫度紀錄儀



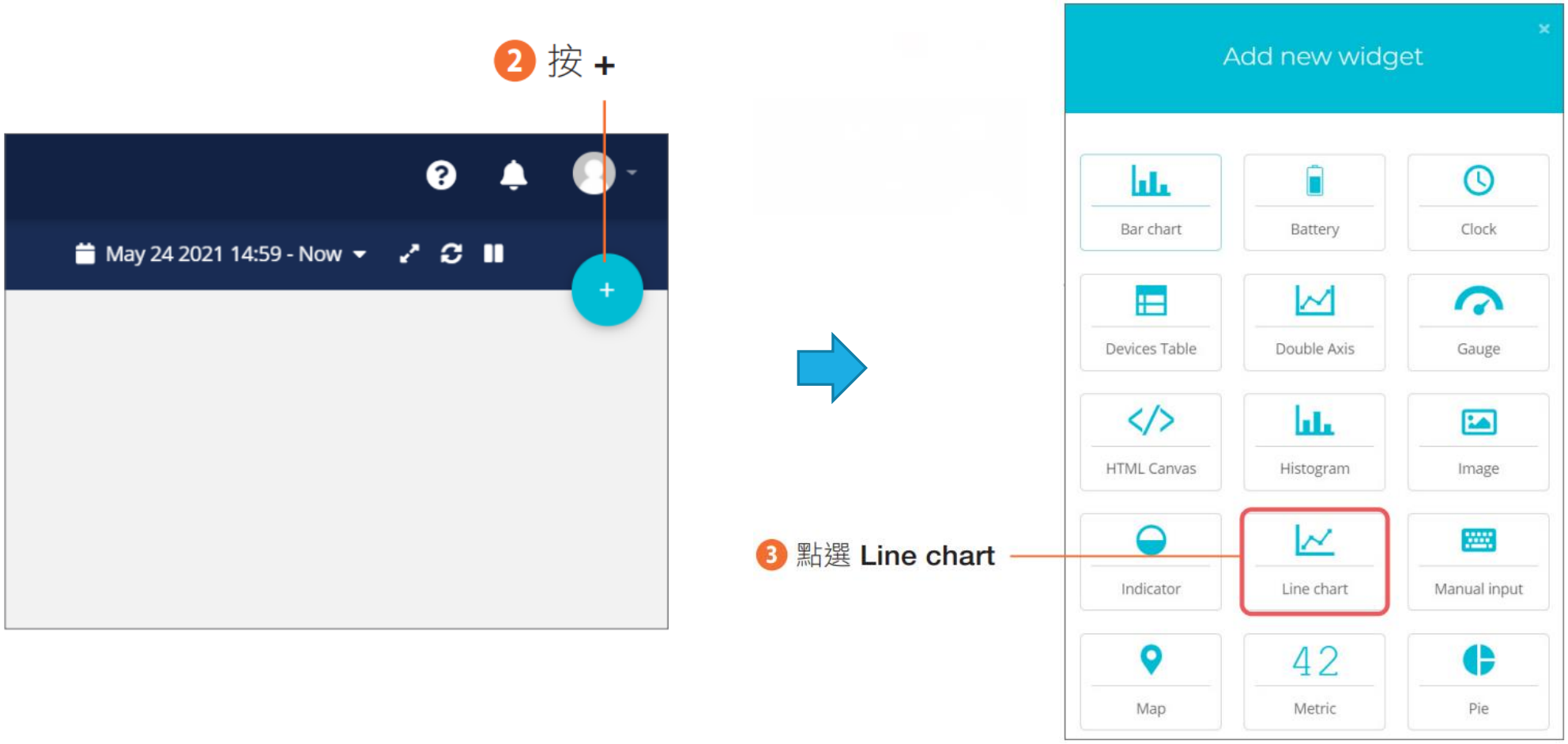
LAB11 雲端溫度紀錄儀

儀錶板(dashboard)



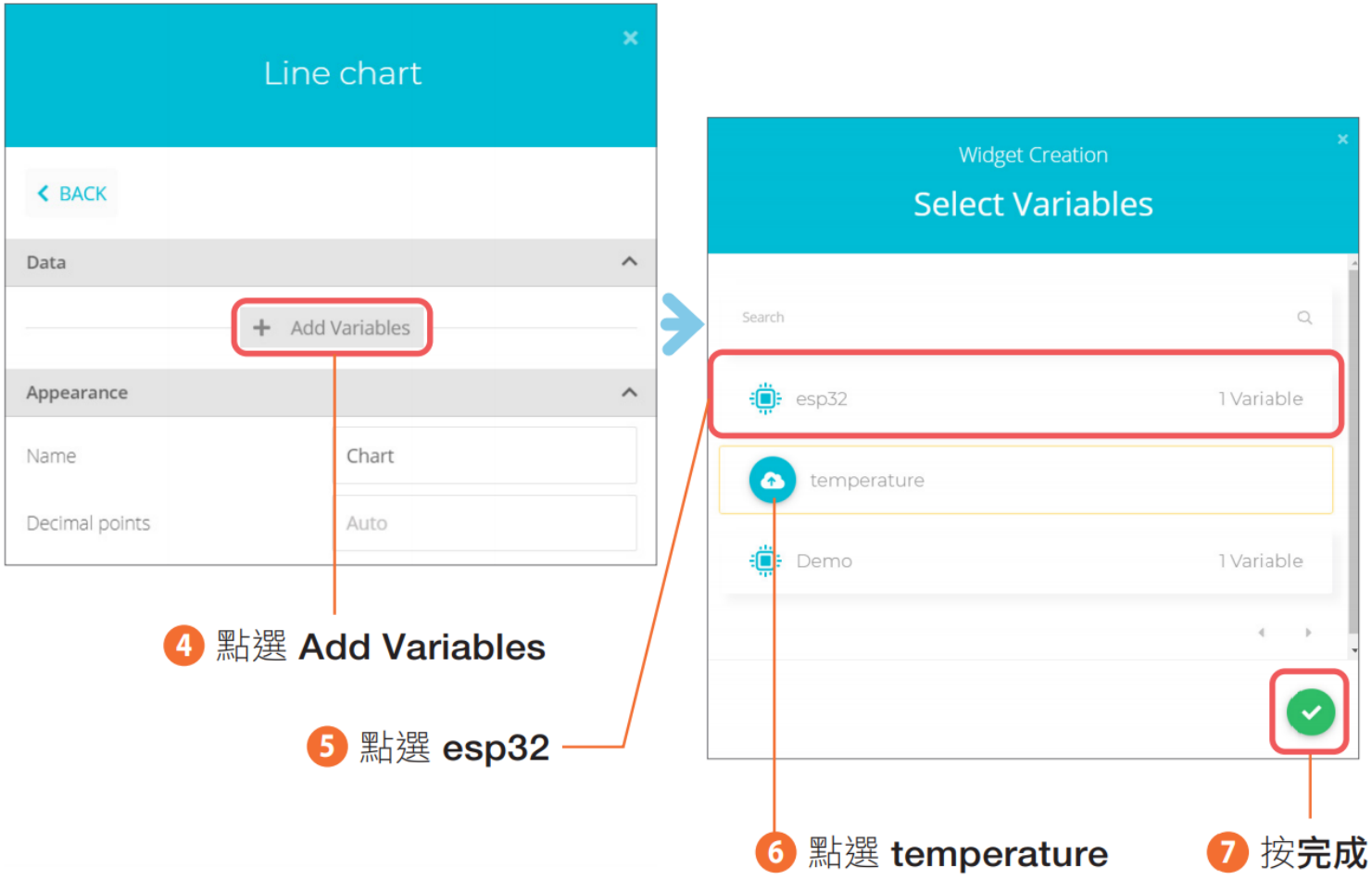
NEXT

LAB11 雲端溫度紀錄儀



NEXT

LAB11 雲端溫度紀錄儀



NEXT

LAB11 雲端溫度紀錄儀

LAB11 雲端溫度紀錄儀

儀錶板(dashboard)分享功能

NEXT

LAB11 雲端溫度紀錄儀

儀錶板(dashboard)分享功能

3 點選**複製連結**

4 貼上連結

目錄

7-1 讓 ESP32 控制板變成網站

ESPWebServer

上傳模組：

7-1 讓 ESP32 控制板變成網站

啟用網站

```
import ESPWebServer                 #  匯入模組
ESPWebServer.begin(80)              #  啟用網站
```

80 稱為**連接埠編號**, 就像是公司內的分機號碼, 其中 80 號連接埠是網站預設使用的編號

7-1 讓 ESP32 控制板變成網站

處理指令

決定如何處理接收到的指令：

```
ESPWebServer.onPath("/Door", handleCmd)
```

瀏覽器網址中指令路徑的方法：

```
http://192.168.100.38/Door
```

附加參數的方法：

```
http://192.168.100.38/Door?status=open
```

NEXT

7-1 讓 ESP32 控制板變成網站

處理網站指令的函式：

```
def handleCmd(socket, args):
    if "status" in args:                    # 判斷是否有名為 status 的參數
        if args["status"] == "open":        # 判斷 status 參數內容是否為 open
            ...
        if args["status"] == "close":
            ...
```

7-1 讓 ESP32 控制板變成網站

回應資料給瀏覽器

狀態碼

```
# 指令正確執行
ESPWebServer.ok(socket, "200", "OK")
# 若指令執行發生錯誤，例如參數不正確
ESPWebServer.err(socket, "400", "ERR")
```

處理指令的函式收到的傳輸用物件

回傳給瀏覽器的資料

7-1 讓 ESP32 控制板變成網站

檢查新收到的請求指令

```
while True:
    ESPWebServer.handleClient()
```

在無窮迴圈中持續檢查是否有新的指令

7-1 讓 ESP32 控制板變成網站

取得 ESP32 的 IP


```
>>>  sta.ifconfig()
('192.168.100.39', '255.255.255.0', '192.168.100.254',
 '168.95.192.1')
```

```
>>>  sta.ifconfig()[0]
192.168.100.39
```

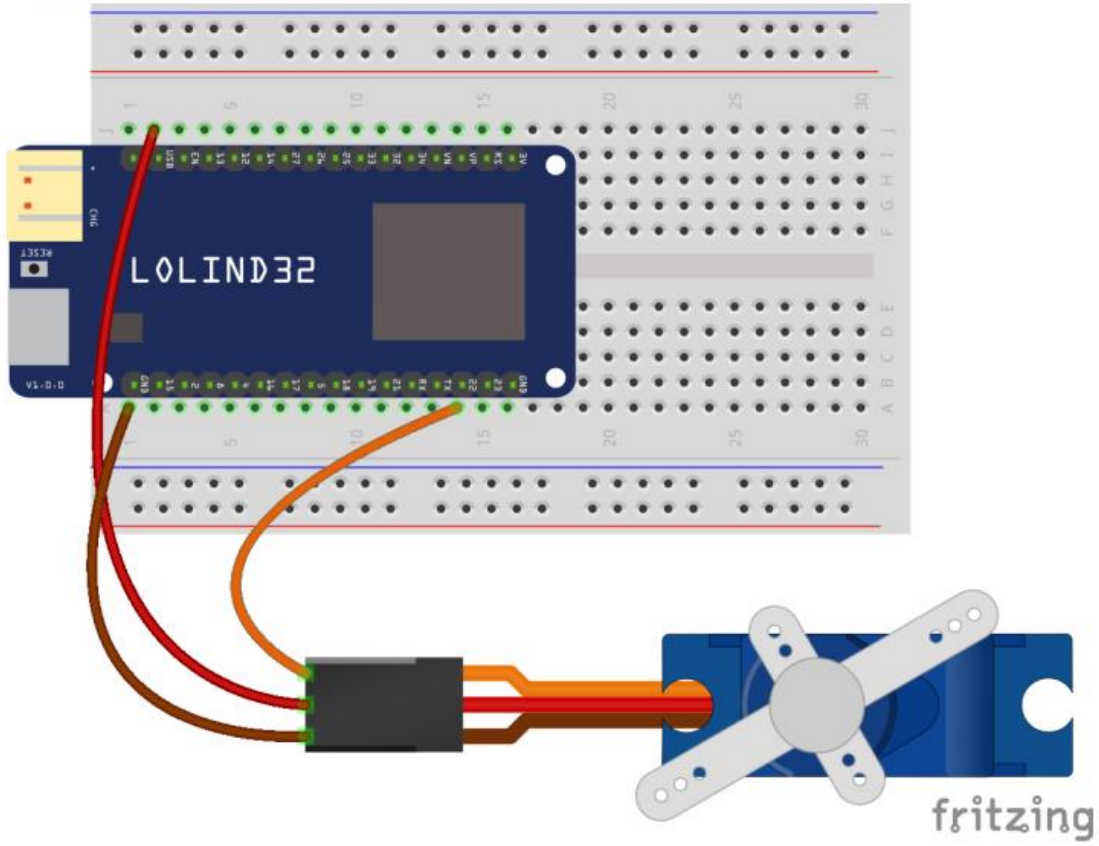
LAB12 無線網路門鎖遙控器

實驗目的	使用手機或電腦的瀏覽器當作控制器來控制伺服馬達。
材料	• ESP32 • 伺服馬達 • 麵包板 • 杜邦線若干條 • 排針 ⚠ 同 LAB02

NEXT

LAB12 無線網路門鎖遙控器

線路圖



ESP32 腳位	伺服馬達
USB	紅線
GND	棕線
22	橘線

⚠ USB 腳位會輸出 5V 電壓。

NEXT

LAB12 無線網路門鎖遙控器

設計原理

假設 ESP32 的 IP 位址為 192.168.100.38, **打開的指令：**

```
http://192.168.100.38/Door?status=open
```

關閉的指令：

```
http://192.168.100.38/Door?status=close
```

NEXT

LAB12 無線網路門鎖遙控器

ESP32 處理指令的函式根據參數值決定伺服馬達的角度：

```
if args["status"] == "open":    # 判斷 status 參數內容是否為 open
    my_servo.write_angle(0)     # 馬達轉至 0 度
if args["status"] == "close":   # 判斷 status 參數內容是否為 close
    my_servo.write_angle(90)    # 馬達轉至 90 度
```

NEXT

LAB12 無線網路門鎖遙控器

程式設計

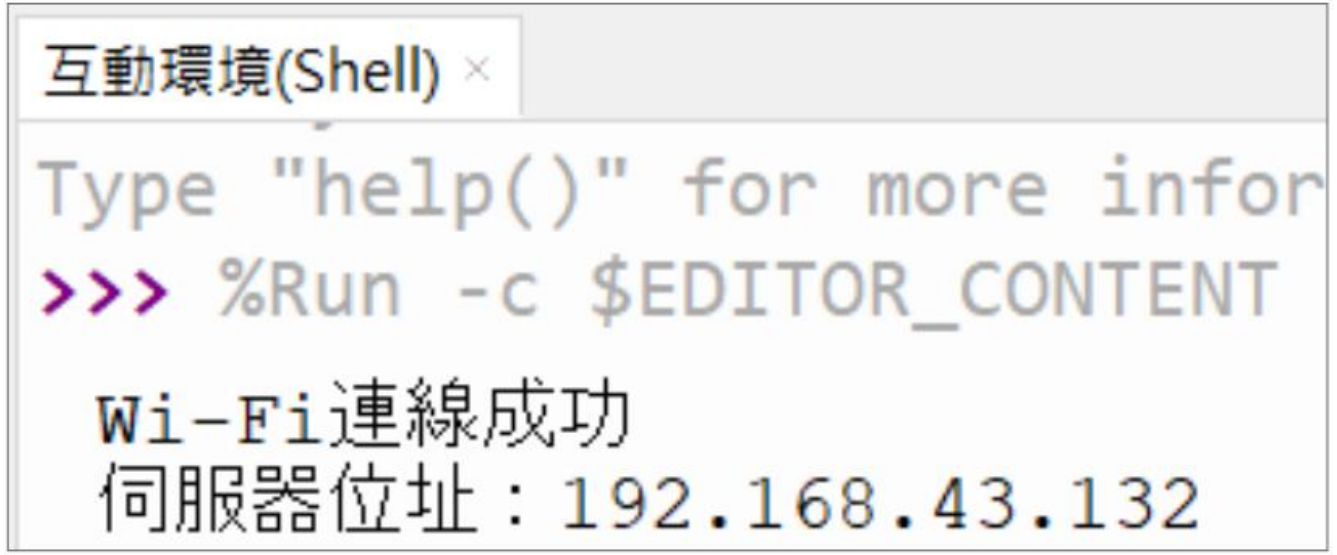
```
01  from servo import Servo
02  import network
03  import ESPWebServer
04  from machine import Pin
05
06  def handleCmd(socket, args):
07      if 'status' in args:
08          print(args['status'])
```

NEXT

LAB12 無線網路門鎖遙控器

實測

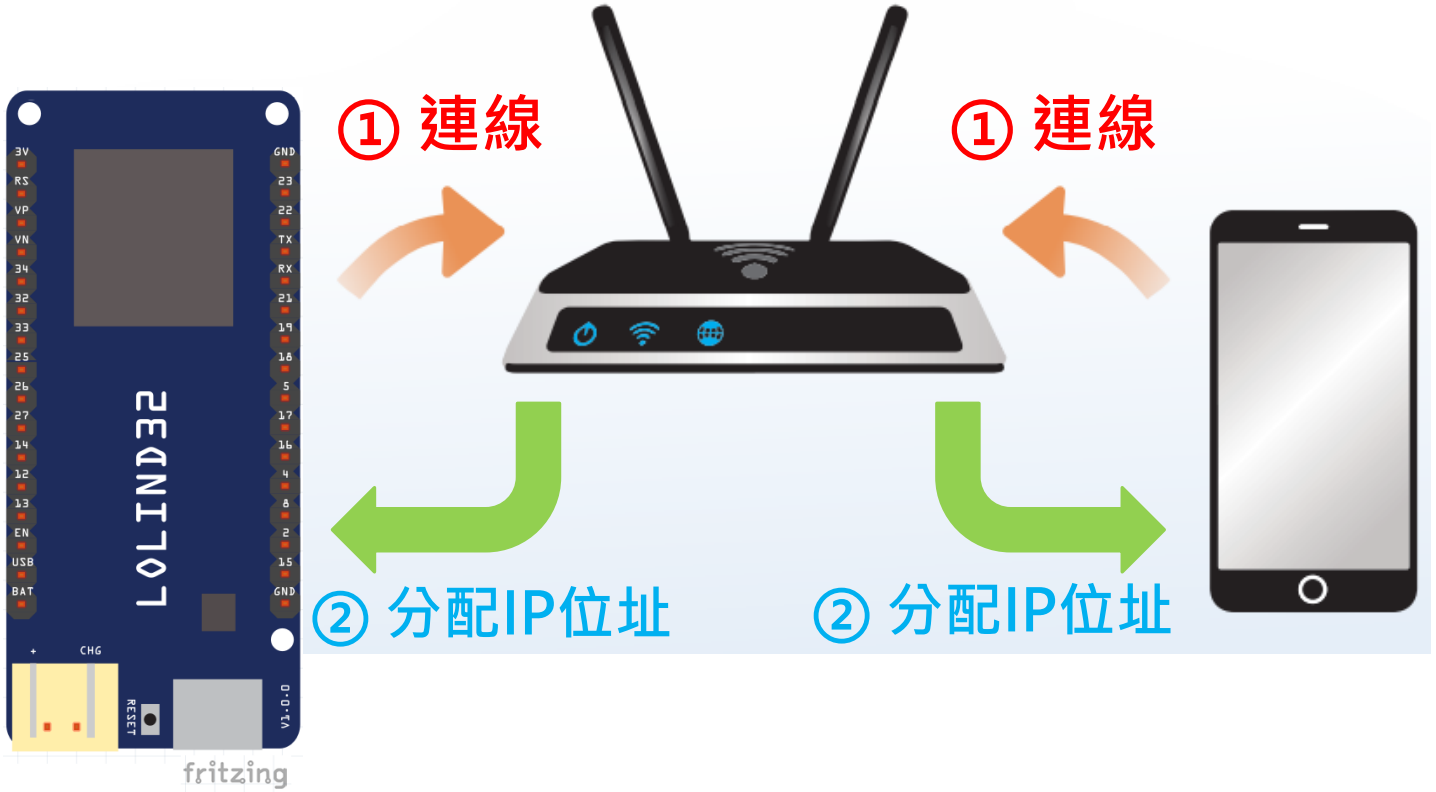
ESP32 連線無線網路後, 會顯示 ESP32 分配到的 IP 位址：



NEXT

LAB12 無線網路門鎖遙控器

使用手機或電腦連接至 ESP32 連的無線網路：



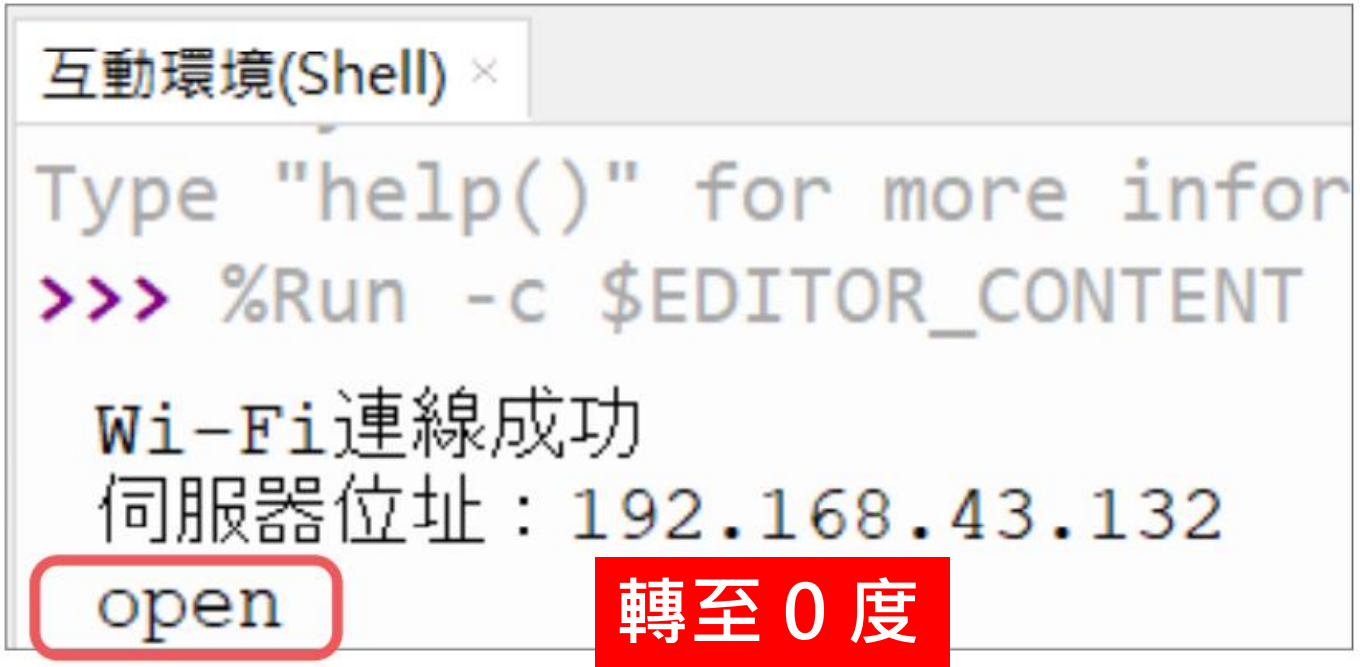
NEXT

LAB12 無線網路門鎖遙控器

手機或電腦

打開

在網址列輸入 **http**
址 /Door?status=
執行『打開』指令



轉至 0 度

ESP32 回傳給瀏覽器的文字。

NEXT

LAB12 無線網路門鎖遙控器

手機或電腦

關閉

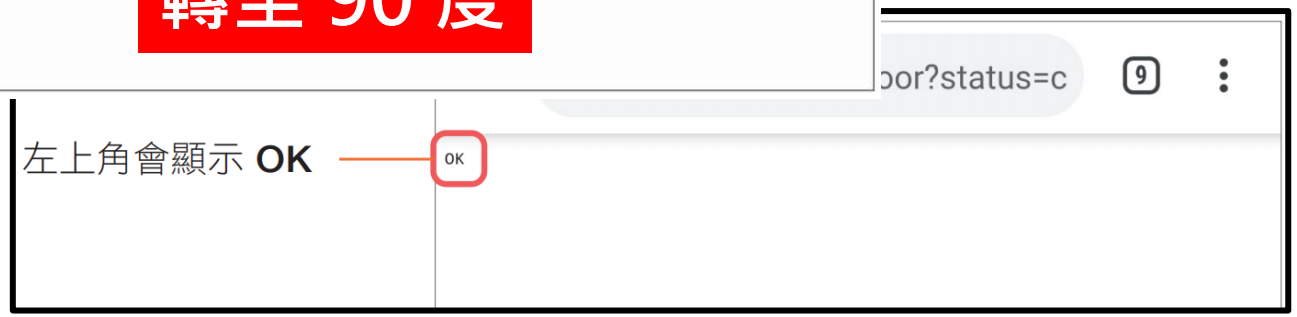
輸入 http://IP 位
Door?status=cl
來執行『關閉』

互動環境(Shell)

```
Type "help()" for more infor
>>> %Run -c $EDITOR_CONTENT

  Wi-Fi連線成功
  伺服器位址：192.168.43.132
  open
  close
```

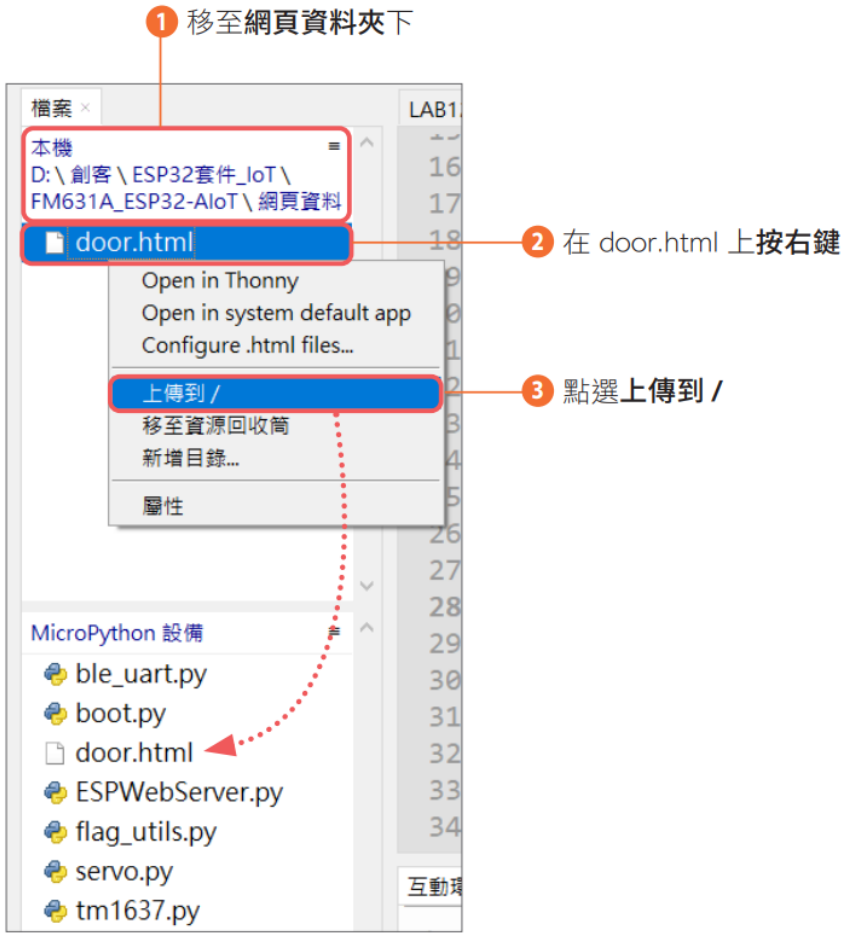
轉至 90 度

左上角會顯示 OK



7-2 使用 HTML 網頁簡化操作

上傳 HTML 網頁到 ESP32



LAB13 網頁門鎖遙控器

實驗目的	使用 HTML 網頁代替手動輸入網址。
材料	同 LAB12

線路圖

同 LAB12

NEXT

LAB13 網頁門鎖遙控器

設計原理


```
01  <!DOCTYPE html>
02  <html>
03  <head>
04    <meta charset='UTF-8'>
05    <meta name='viewport'
06      content='width=device-width, initial-scale=1.0'>
07    <title>門鎖</title>
08  </head>
09  <body>
10    <h1>
11      <a href='/Door?status=open'>打開</a> 或
12      <a href='/Door?status=close'>關閉</a></h1>
13  </body>
14  </html>
```

NEXT

LAB13 網頁門鎖遙控器

程式設計

同 LAB12

NEXT

LAB13 網頁門鎖遙控器

實測

ESP32 連線無線網路後, 會顯示 ESP32 分配到的 IP 位址：

NEXT

LAB13 網頁門鎖遙控器

使用手機或電腦連接至 ESP32 連的無線網路, 並開啟瀏覽器：

NEXT

LAB13 網頁門鎖遙控器

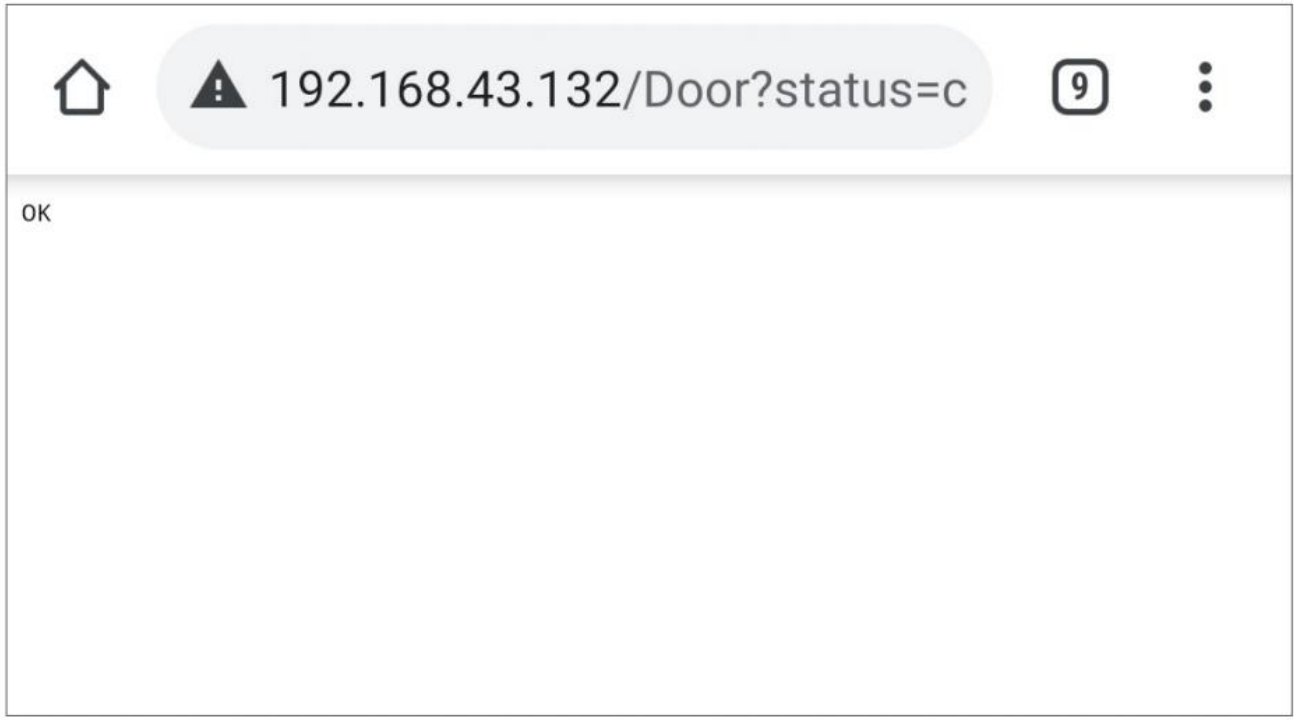
ESP32 回傳網頁：

⚠ 如果沒有出現以上畫面，可以檢查網路連線是否與 ESP32 相同，或是沒有上傳網頁資料(可參考 P.74)。

NEXT

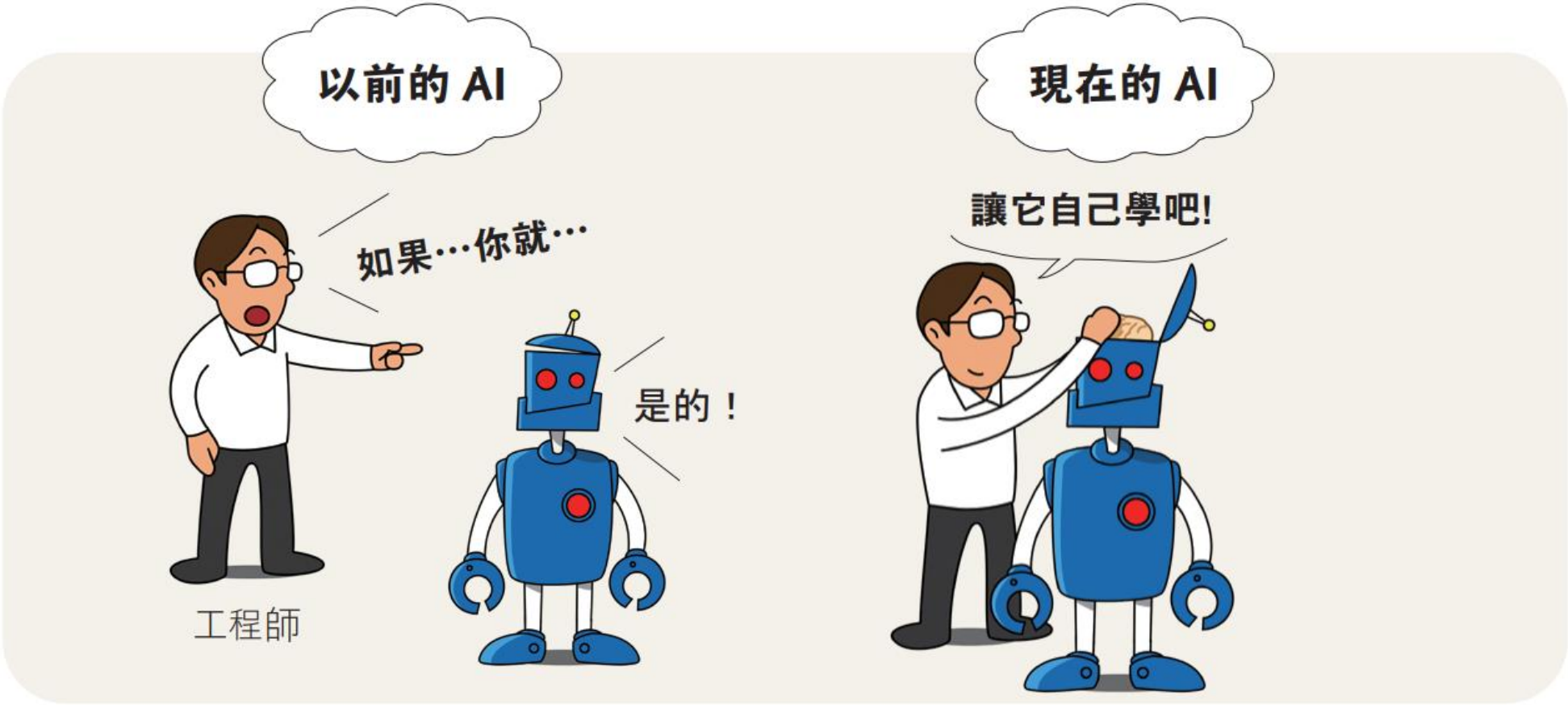
LAB13 網頁門鎖遙控器

點選『打開』或『關閉』：



目錄

8-1 AI 簡介



8-2 人臉偵測

1. 相機確認人臉位置並聚焦

2. 微軟推出一個有趣的
   年齡測試, 可以根據
   人臉判斷年齡

8-2 人臉偵測

開啟攝影機與年齡偵測

8-3 GitHub Pages

問題： 使用攝影機需要透過 HTTPS 加密協定載入，第 7 章的 ESPWebServer 模組沒有提供此功能。

解決方案： 將網頁架設在 GitHub Pages 上。

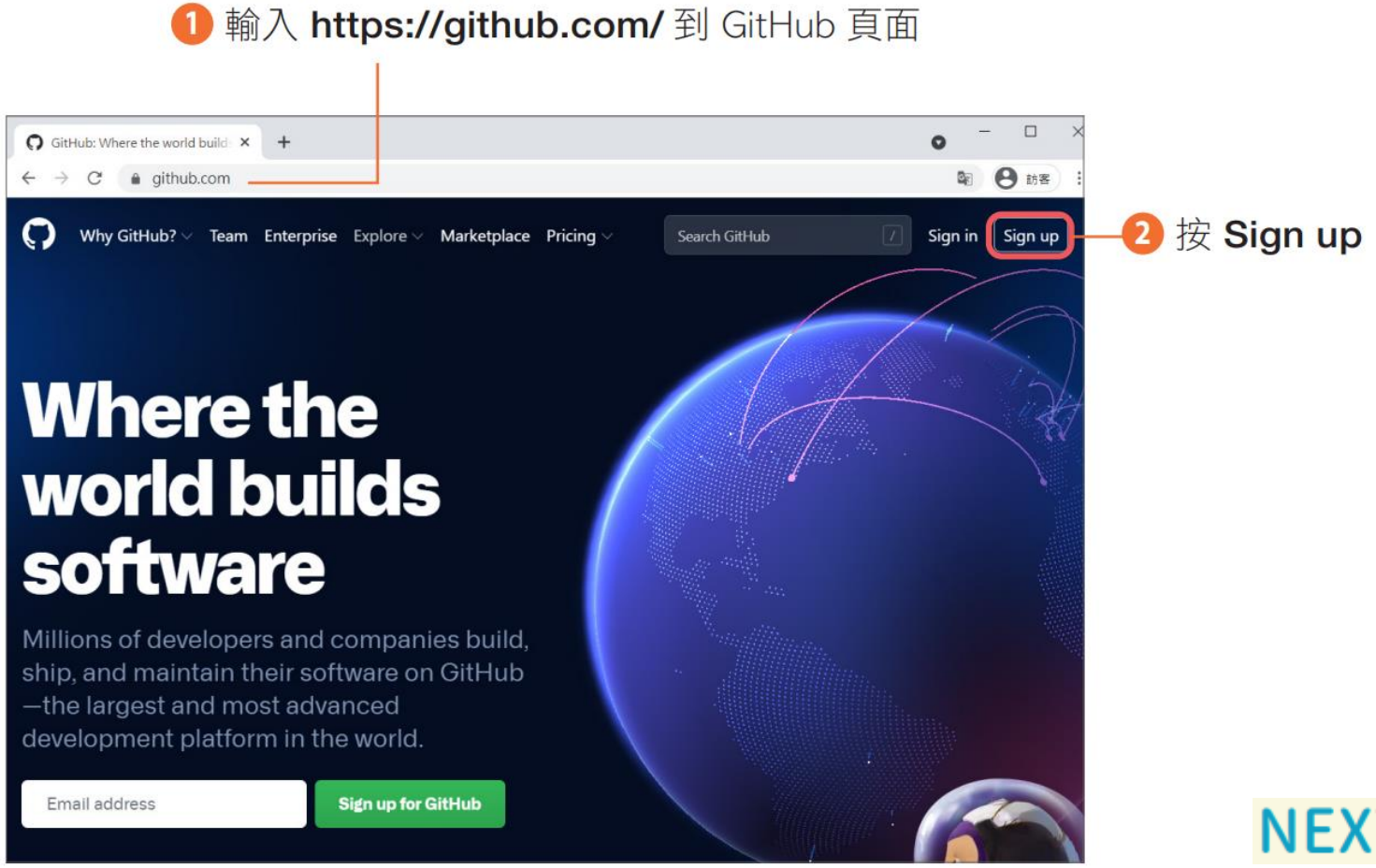
8-3 GitHub Pages

HTML 檔上傳至 GitHub, 即可將 GitHub Pages 當作網頁伺服器使用

8-3 GitHub Pages

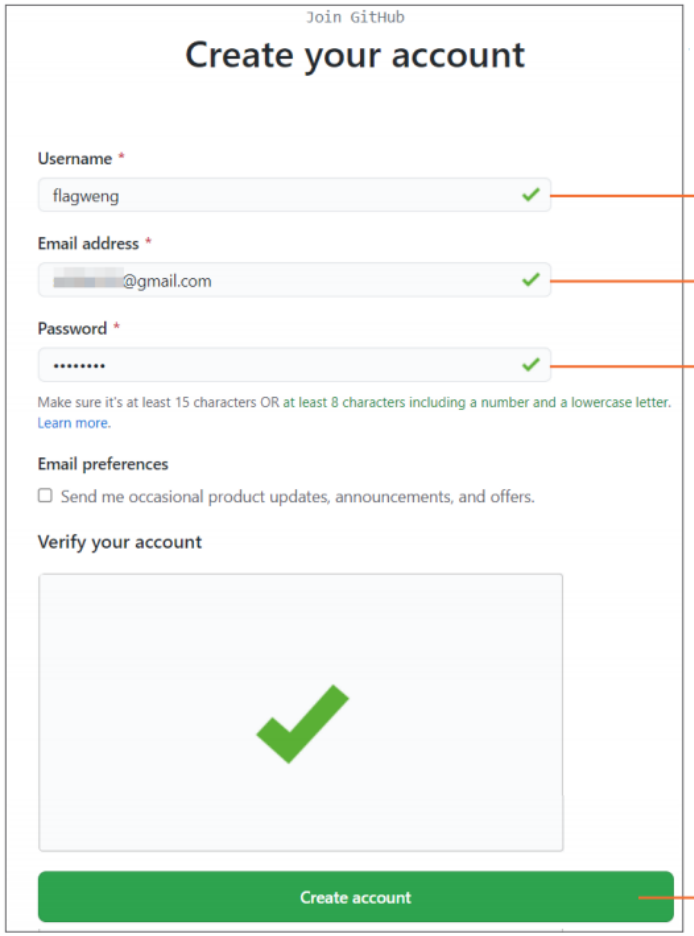
註冊 GitHub 帳號

網址列輸入：
https://github.com/



NEXT

8-3 GitHub Pages

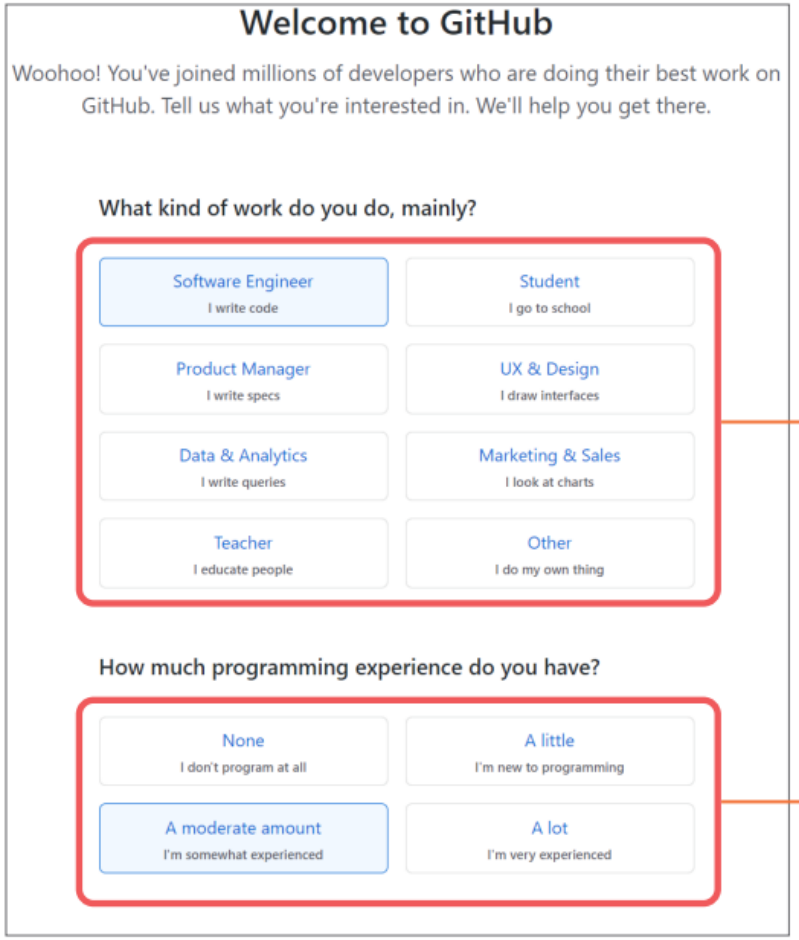


3 輸入使用者名稱

4 輸入電子信箱

5 輸入密碼

6 按 Create account



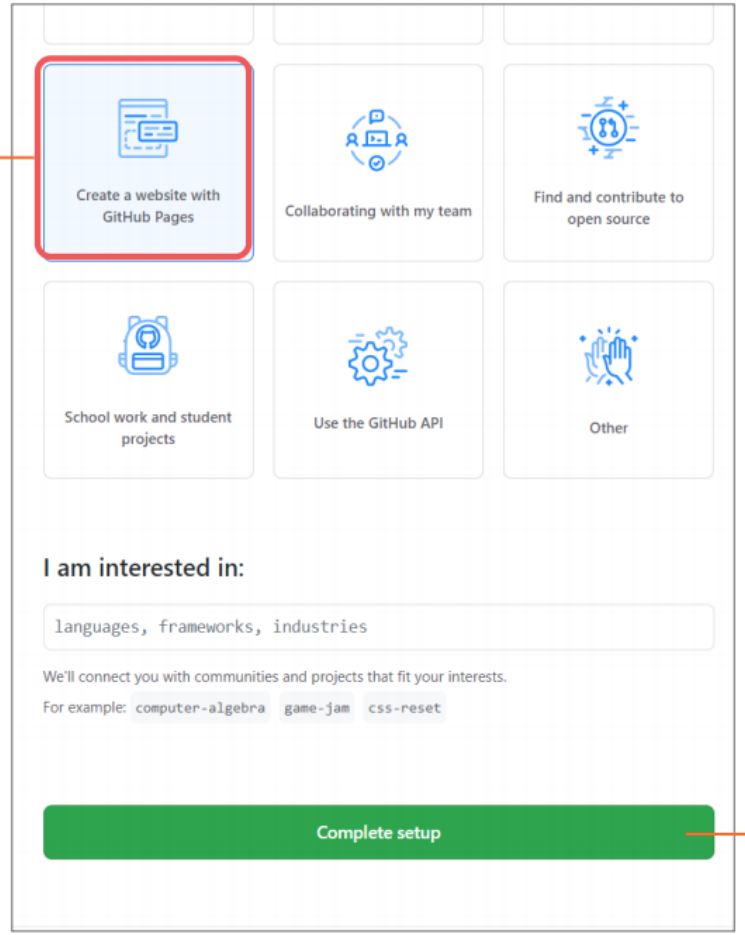
7 選擇職業

8 選擇程式經驗

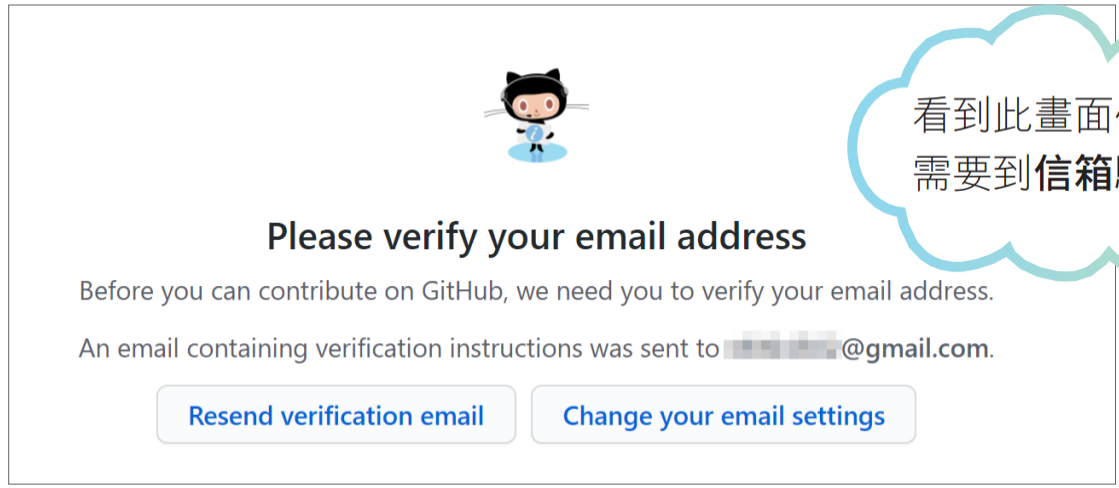
NEXT

8-3 GitHub Pages

9 選擇 **Create a website with GitHub Pages**



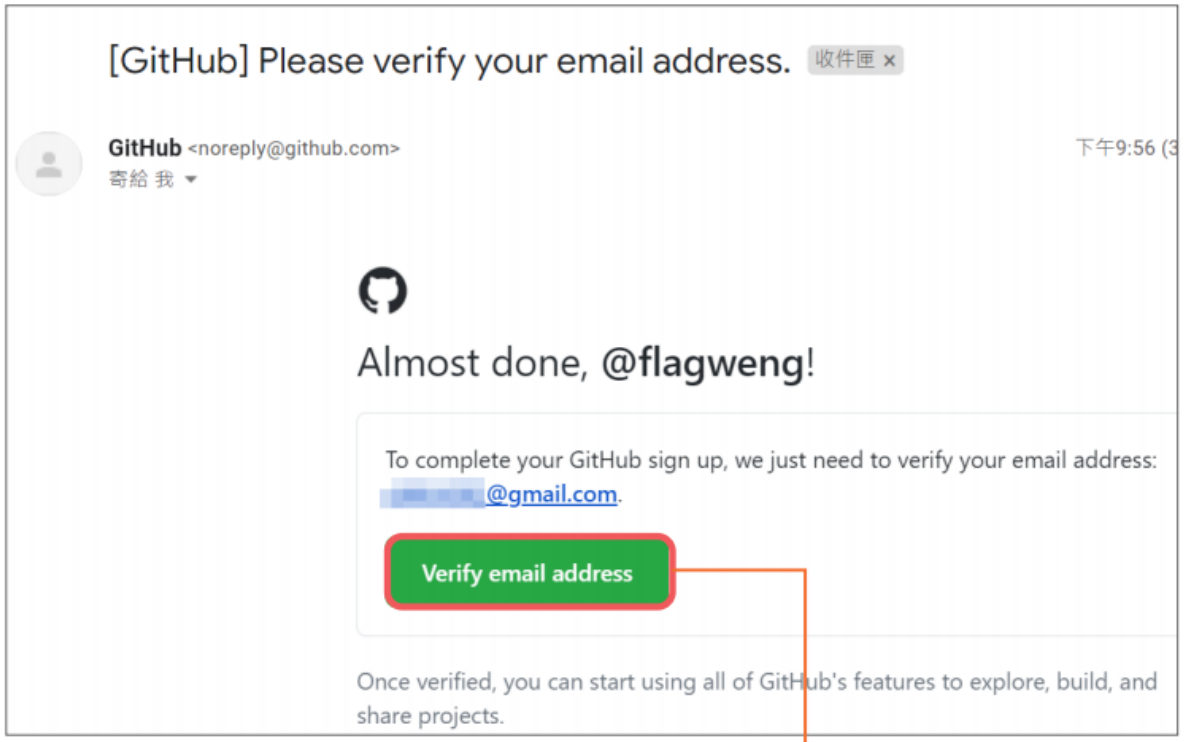
10 按 **Complete setup**



看到此畫面代表需要到**信箱驗證**

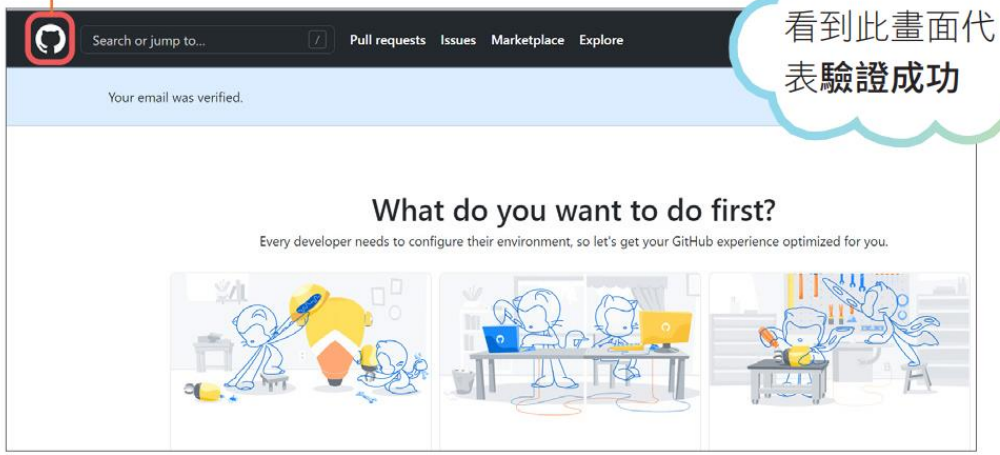
NEXT

8-3 GitHub Pages



11 在收到的驗證信中**按此驗證**

12 按此回到 GitHub 主畫面



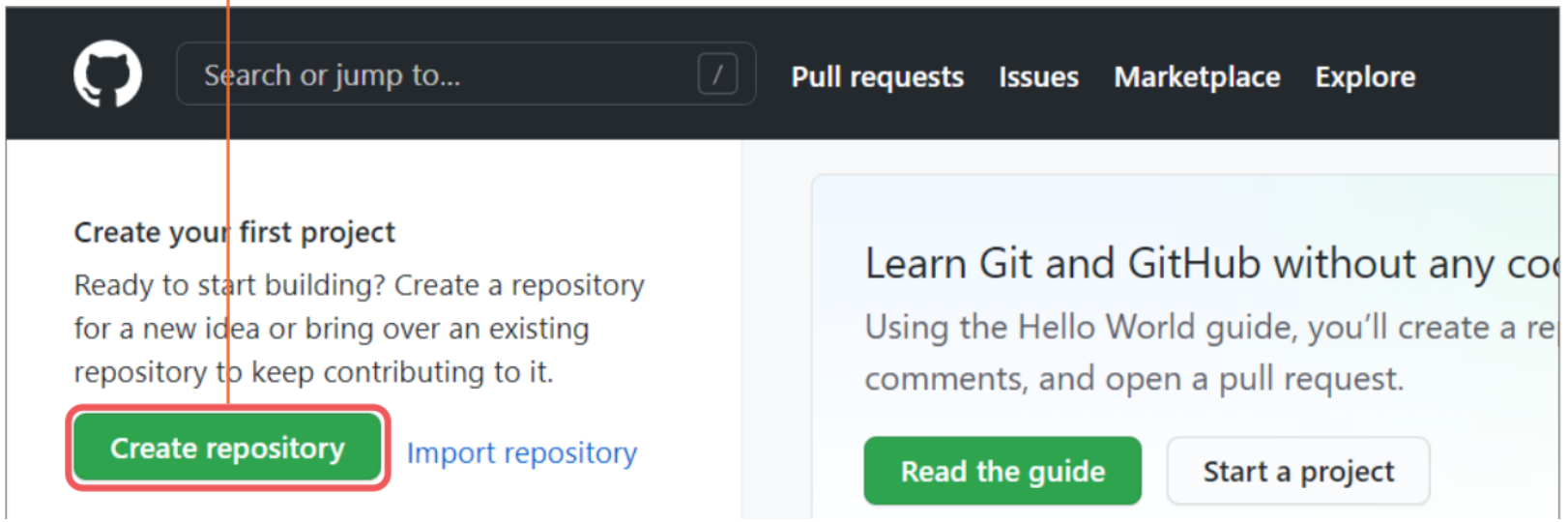
看到此畫面代表**驗證成功**

8-3 GitHub Pages

使用 GitHub

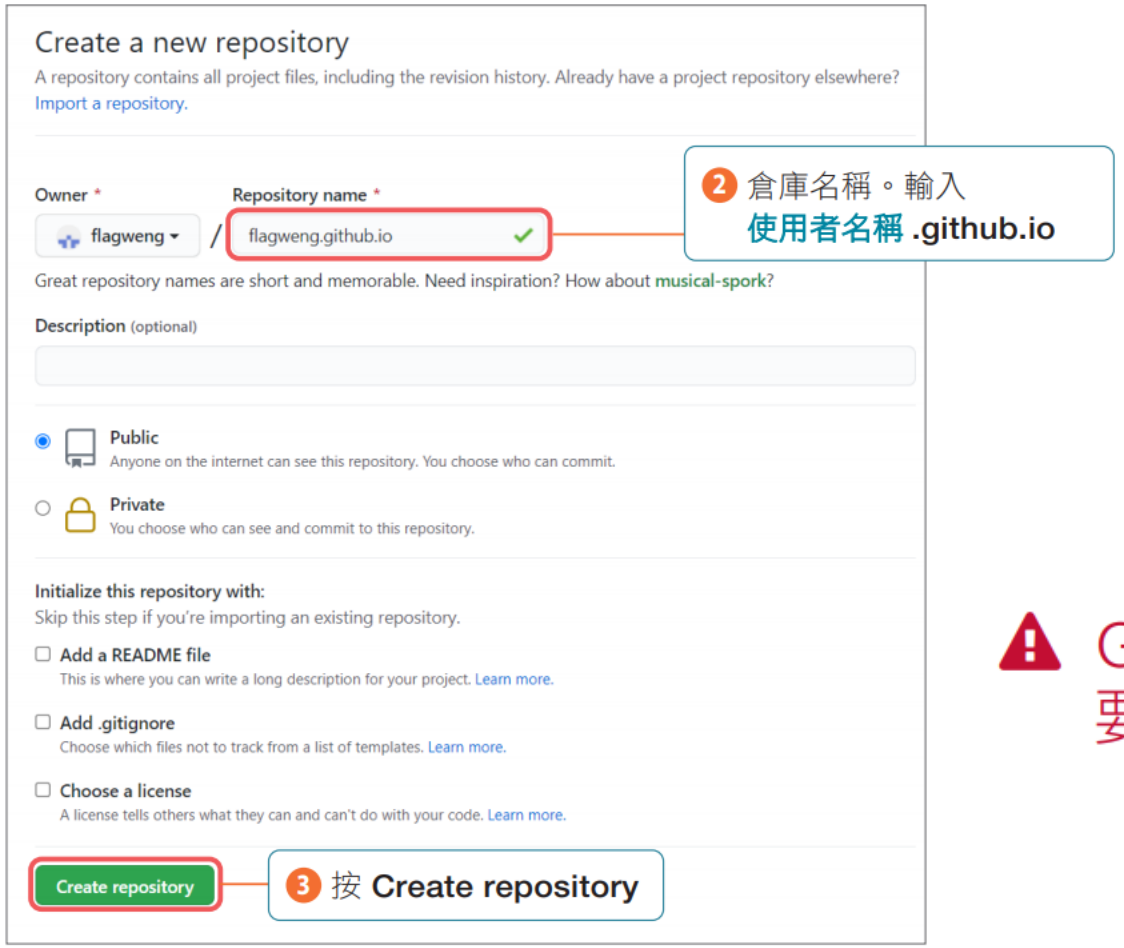
建立倉庫 **專屬於 GitHub Pages 的倉庫**

1 按 **Create repository** 建立倉庫



NEXT

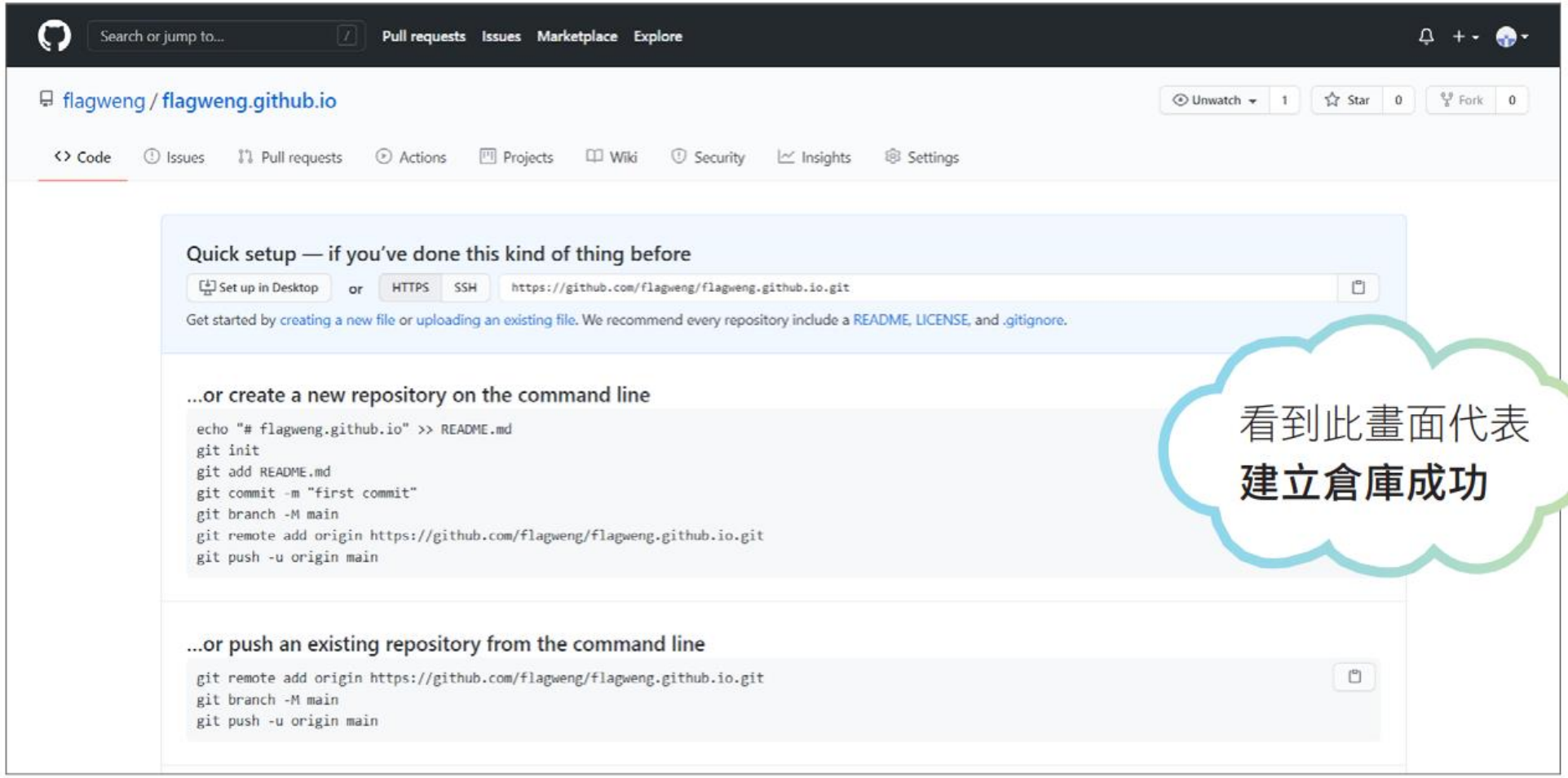
8-3 GitHub Pages



⚠ Github Pages 的倉庫名稱一定要是**使用者名稱**加上 **.github.io**。

NEXT

8-3 GitHub Pages

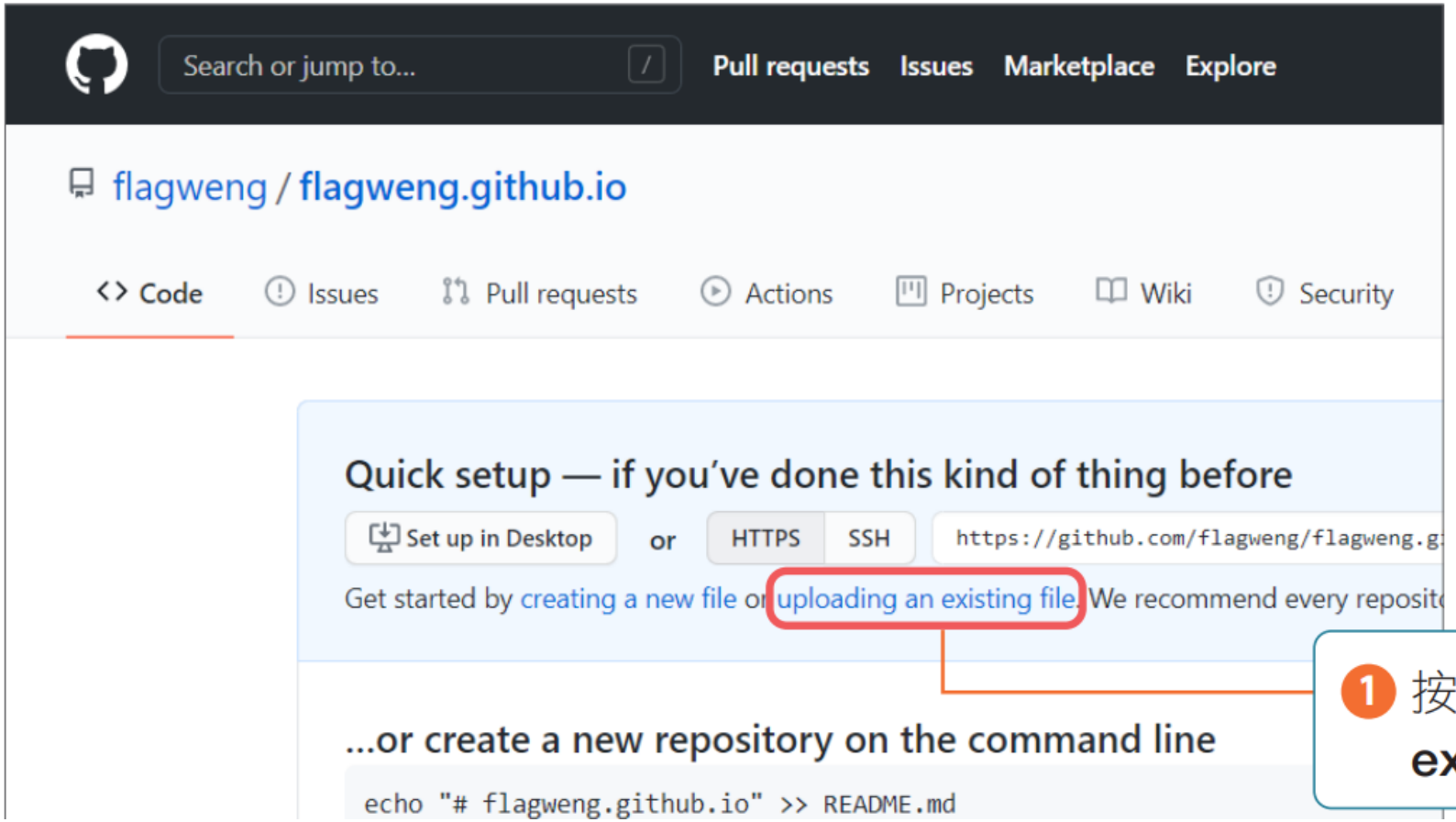


8-3 GitHub Pages

上傳檔案

NEXT

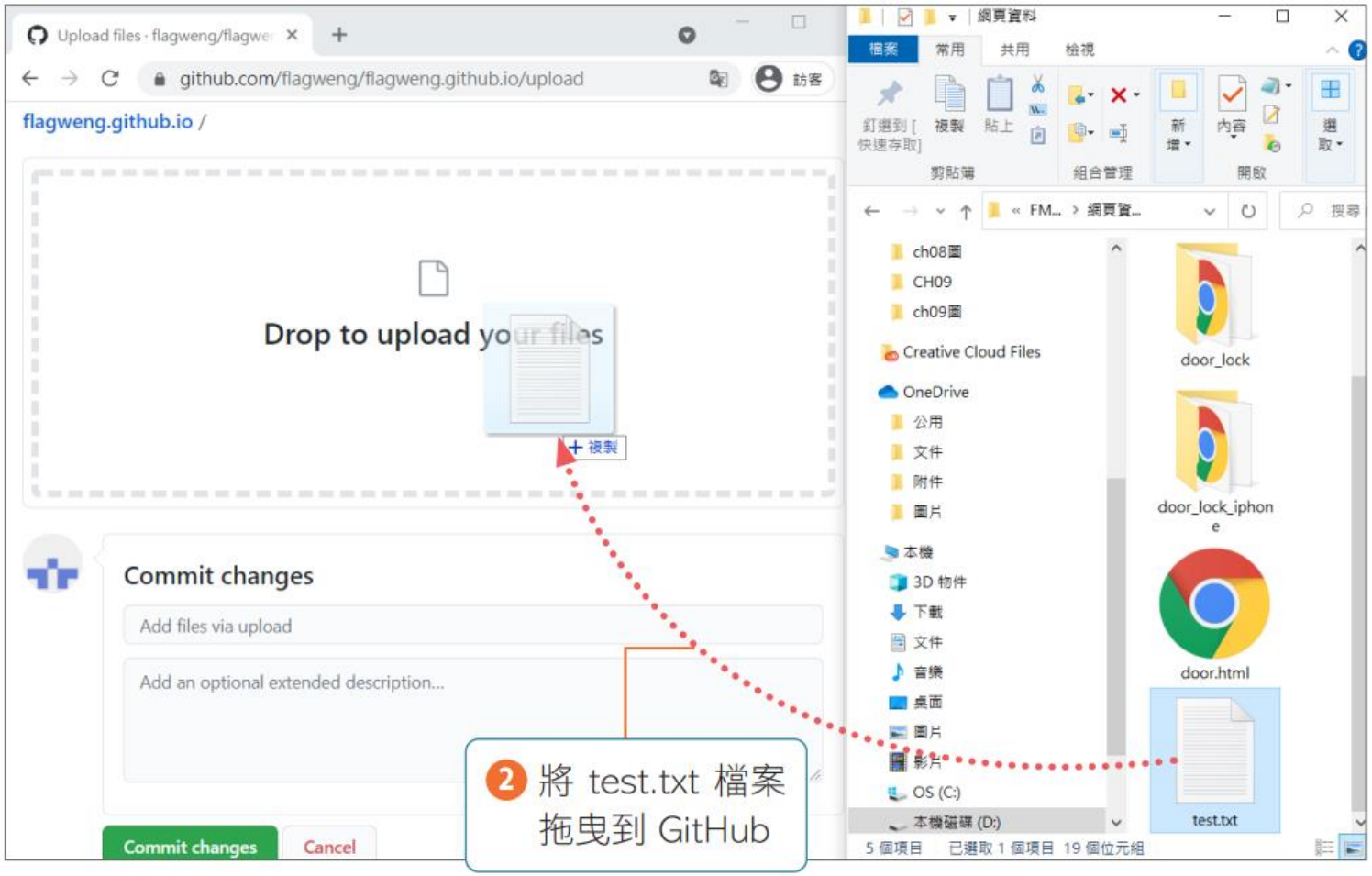
8-3 GitHub Pages



1 按 uploading an existing file

NEXT

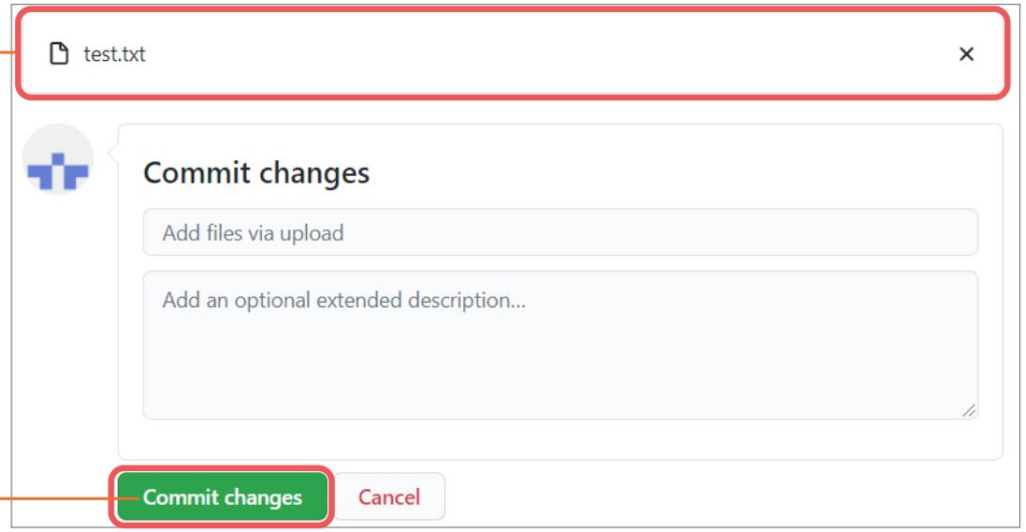
8-3 GitHub Pages



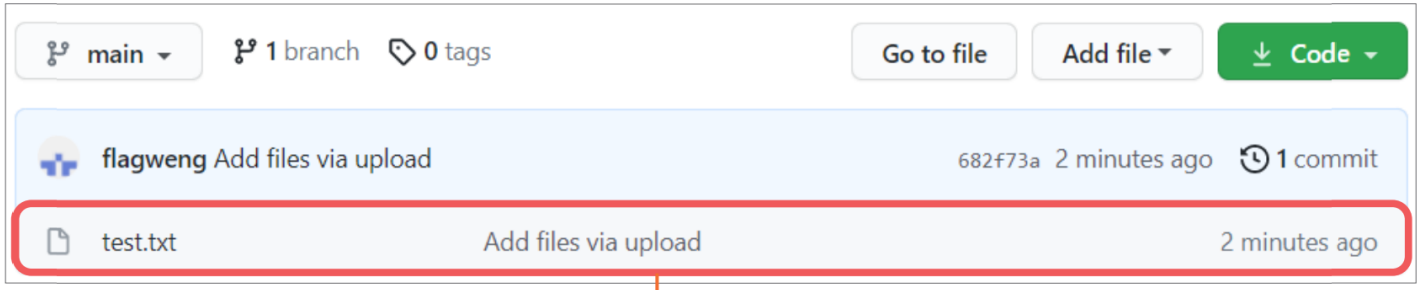
NEXT

8-3 GitHub Pages

看到 test.txt
代表上傳完畢



3 上傳完畢後，
按 **Commit changes**



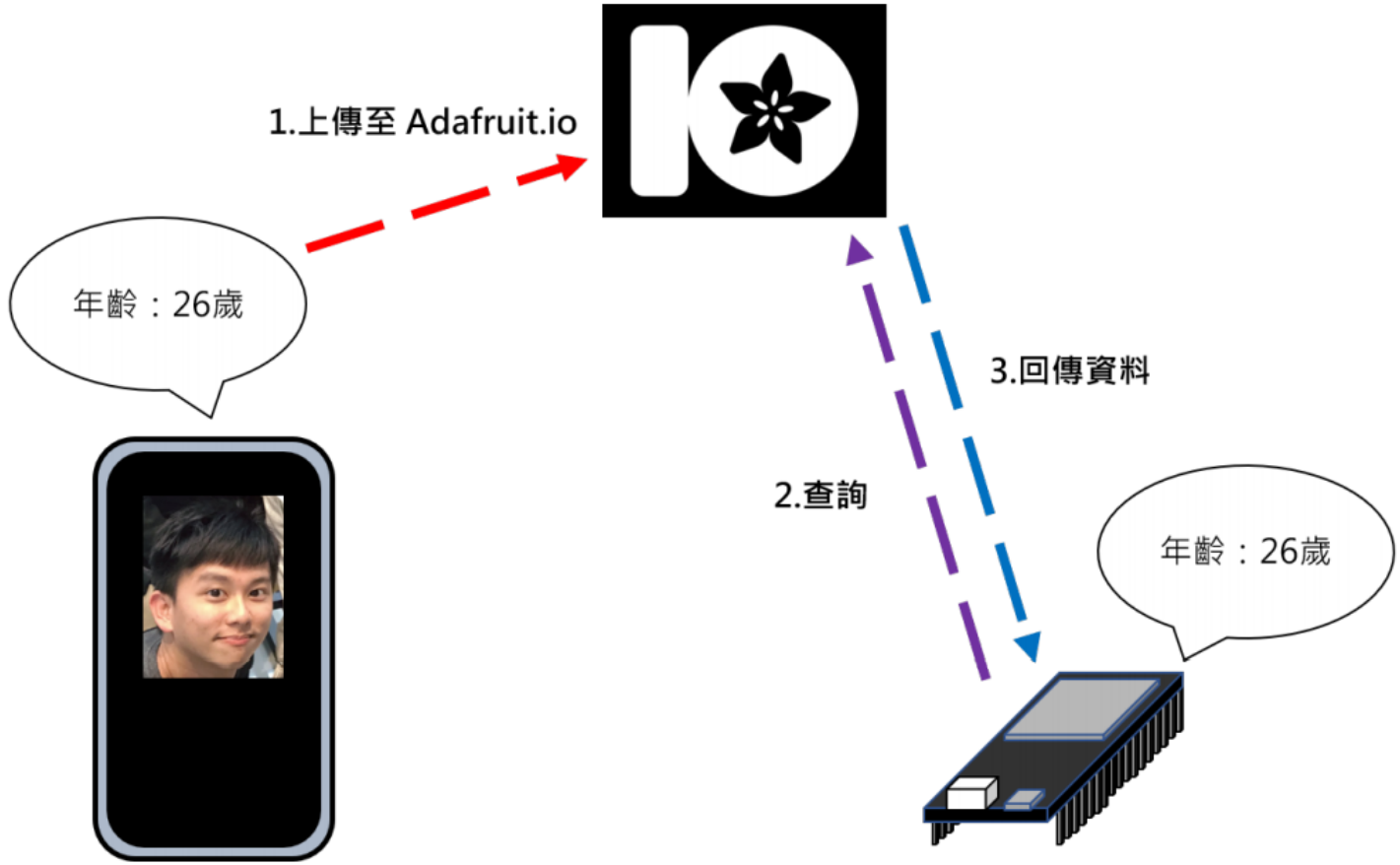
看到 test.txt 已成功上傳到 GitHub

8-3 GitHub Pages

網

8-4 Adafruit IO

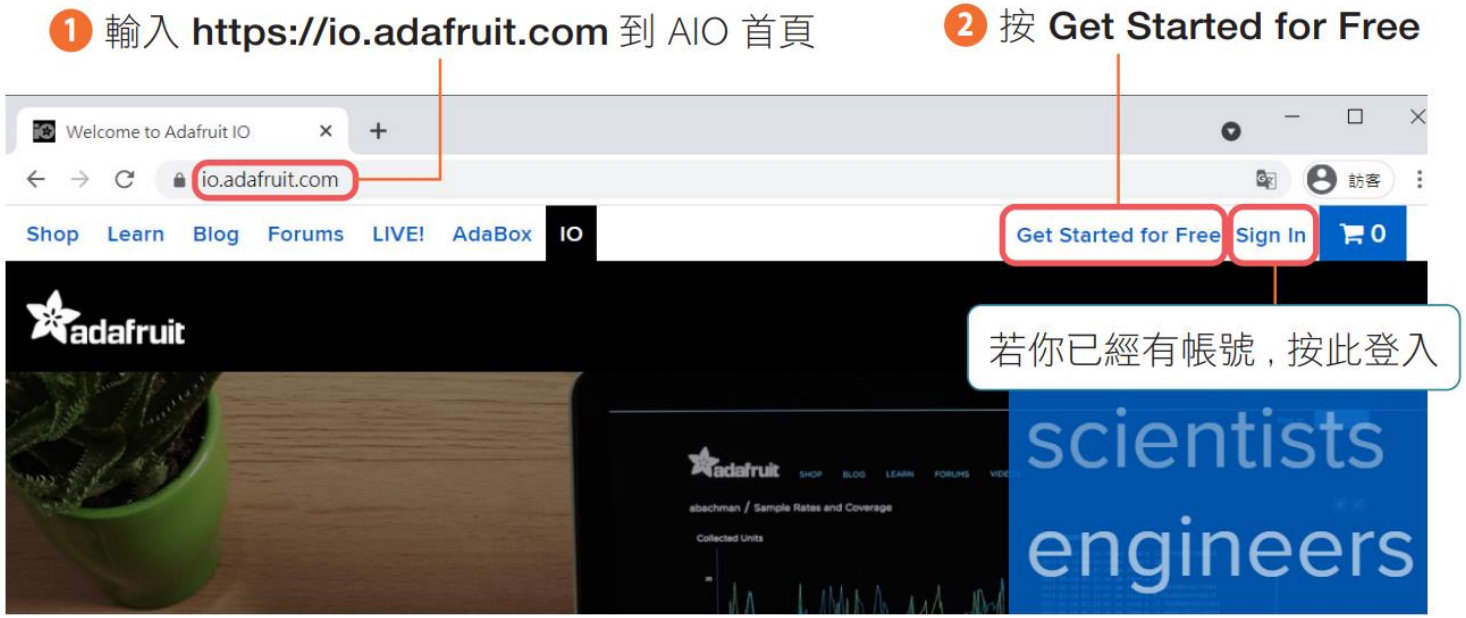
Adafruit IO 是 Adafruit 公司提供的 IoT 平台, 它可以用來存放和取得資料



8-4 Adafruit IO

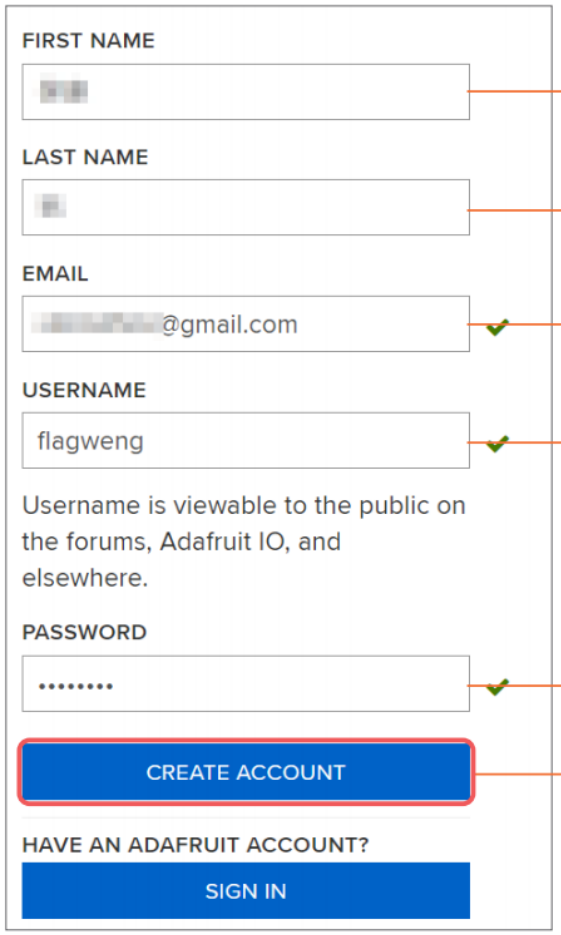
註冊帳號

網址列輸入：https://io.adafruit.com



NEXT

8-4 Adafruit IO



3 輸入**名字**（中英文皆可）

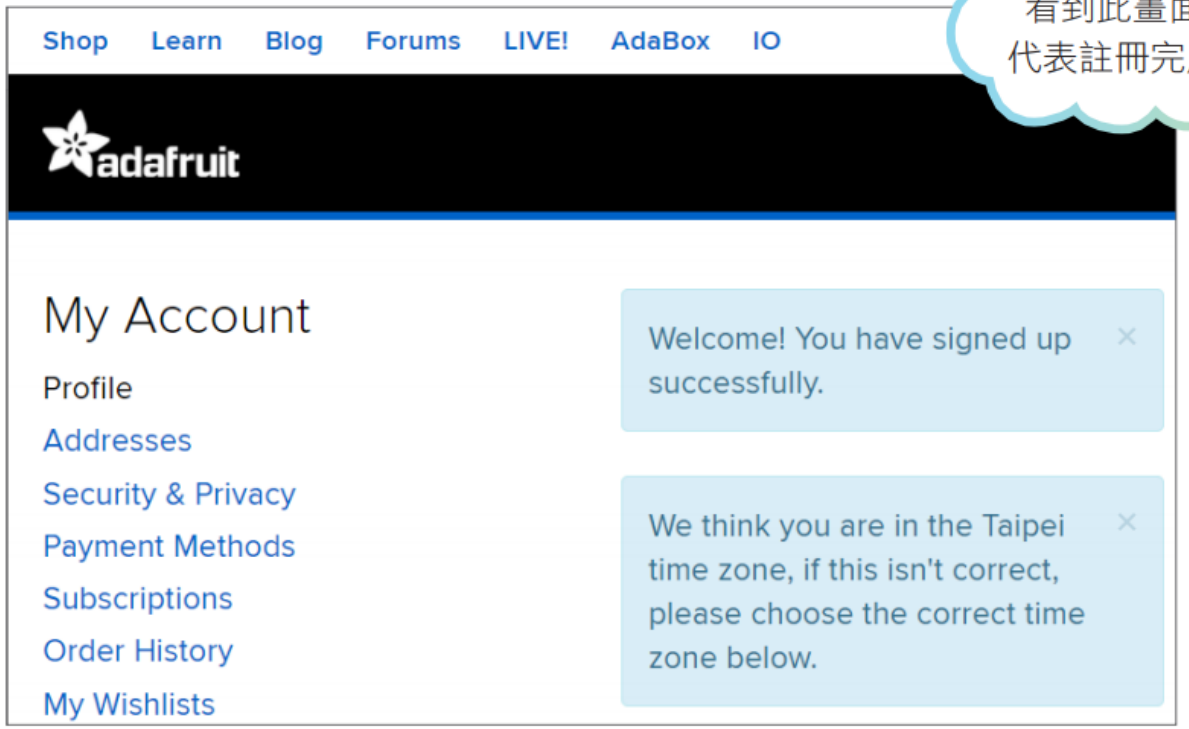
4 輸入**姓氏**（中英文皆可）

5 輸入**信箱**

6 輸入使用者名稱**（請輸入英文）**

7 輸入**密碼**

8 按 **CREATE ACCOUNT**



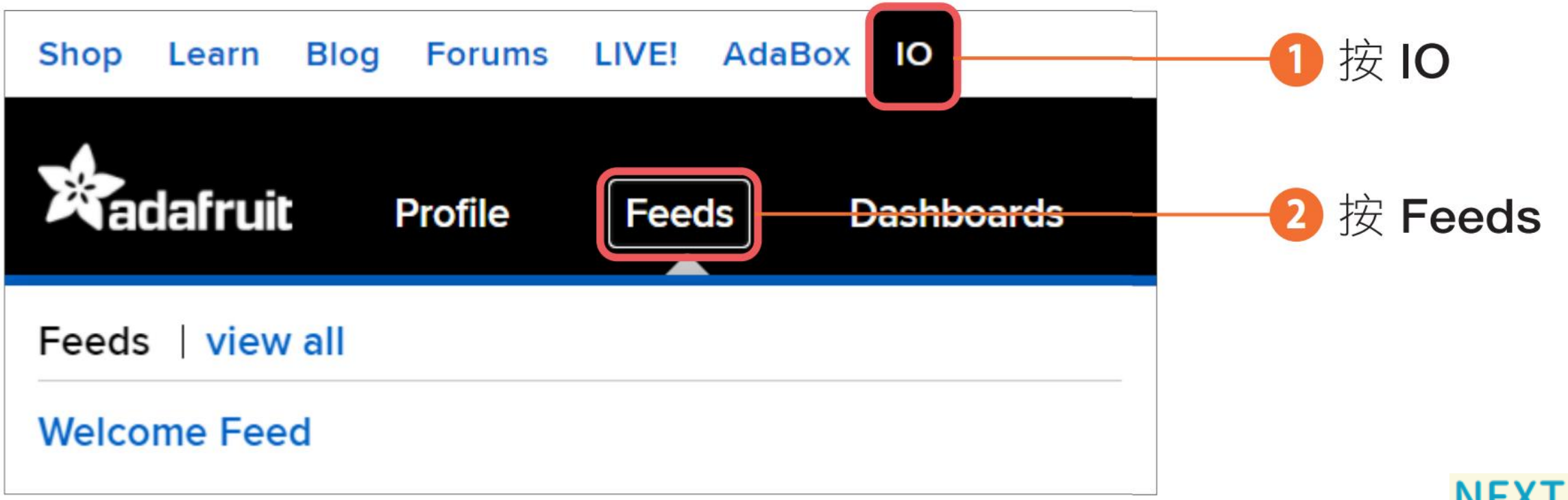
看到此畫面
代表註冊完成

NEXT

8-4 Adafruit IO

建立 Feed

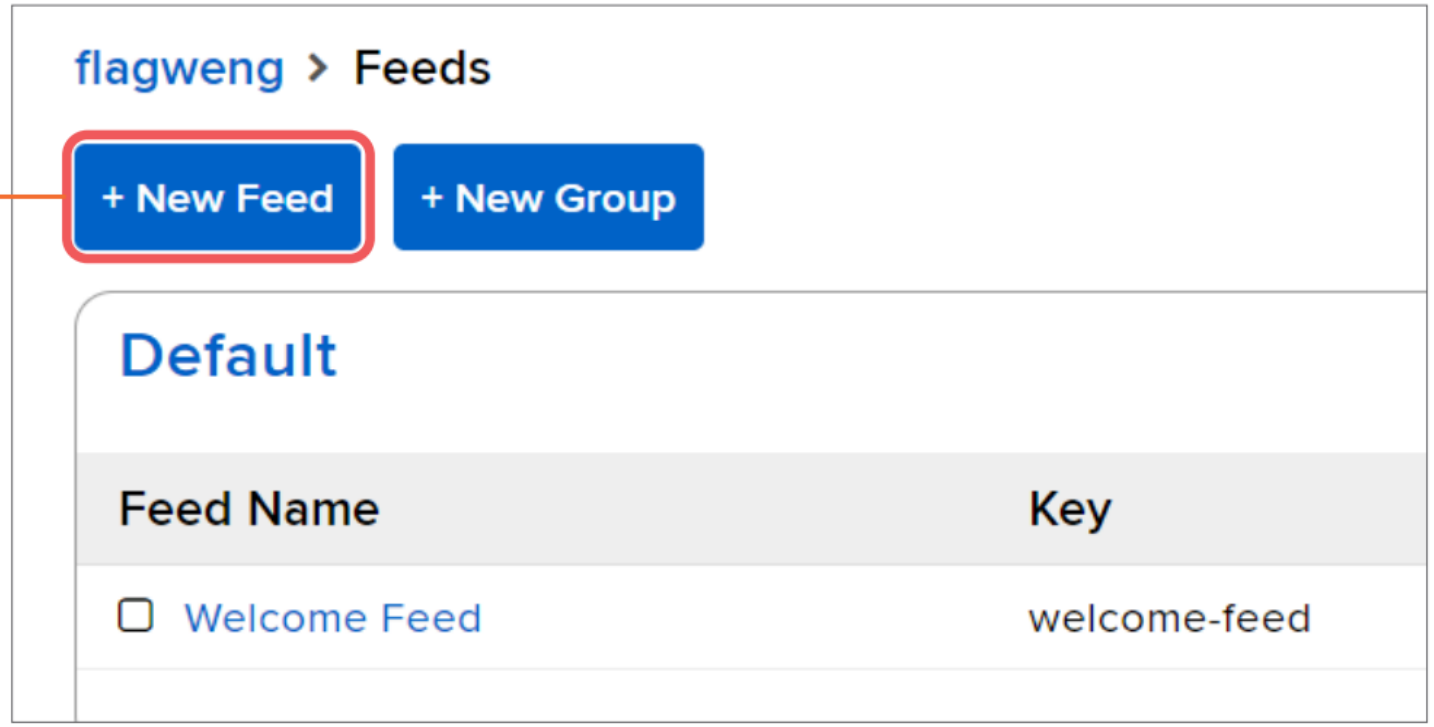
Feed 是資料來源的意思, 我們上傳的資料都會存放在 Feeds 頁次



NEXT

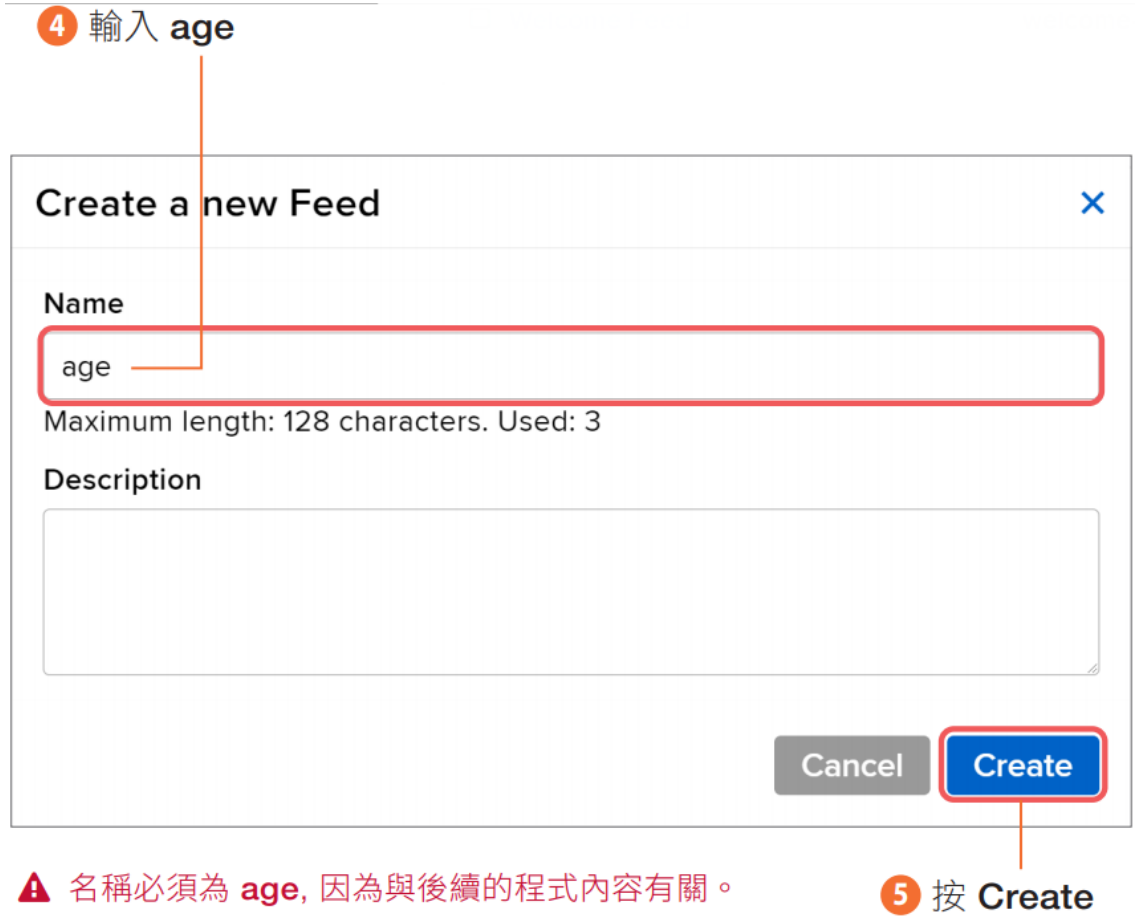
8-4 Adafruit IO

3 按 +New Feed



NEXT

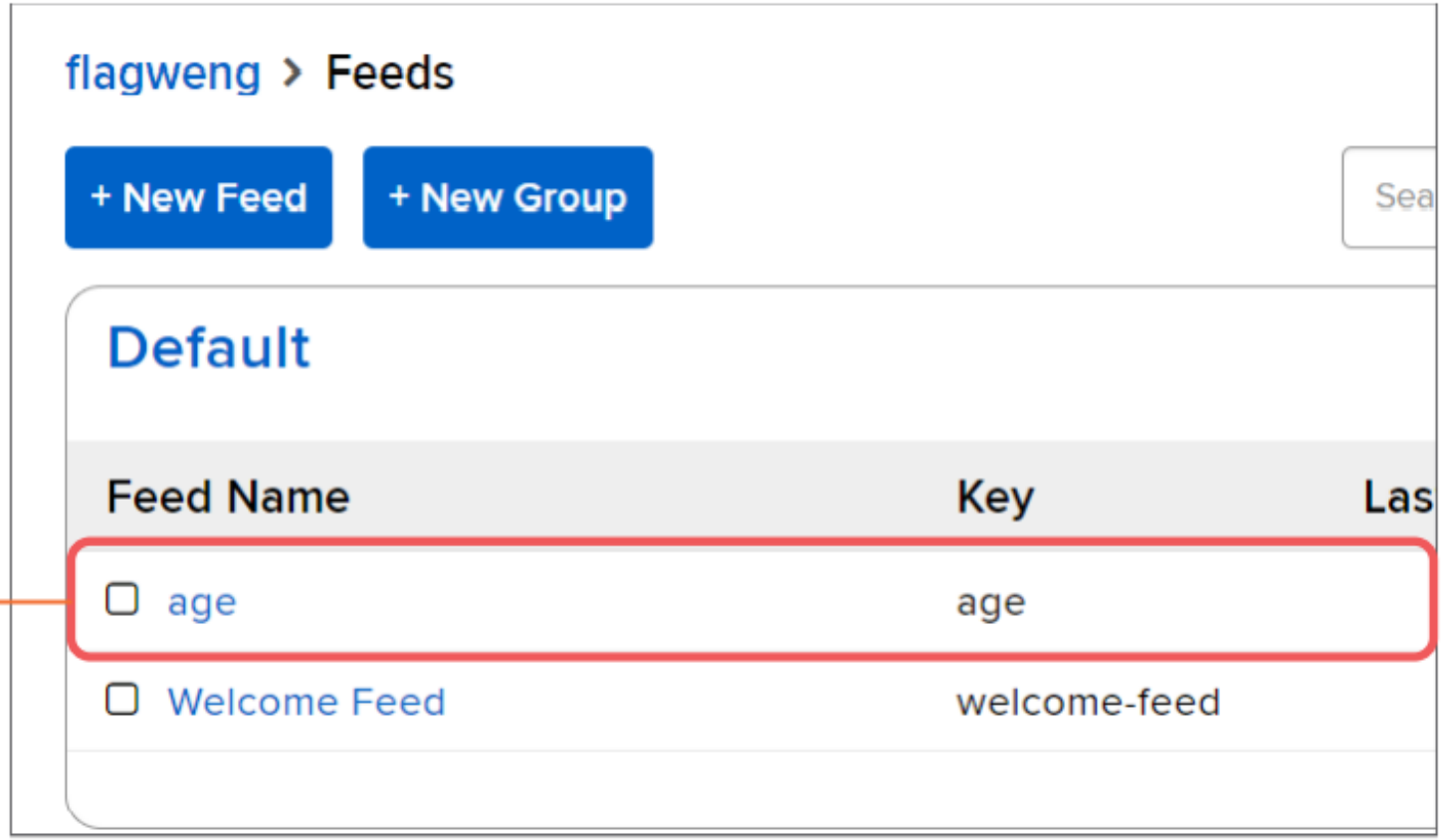
8-4 Adafruit IO



NEXT

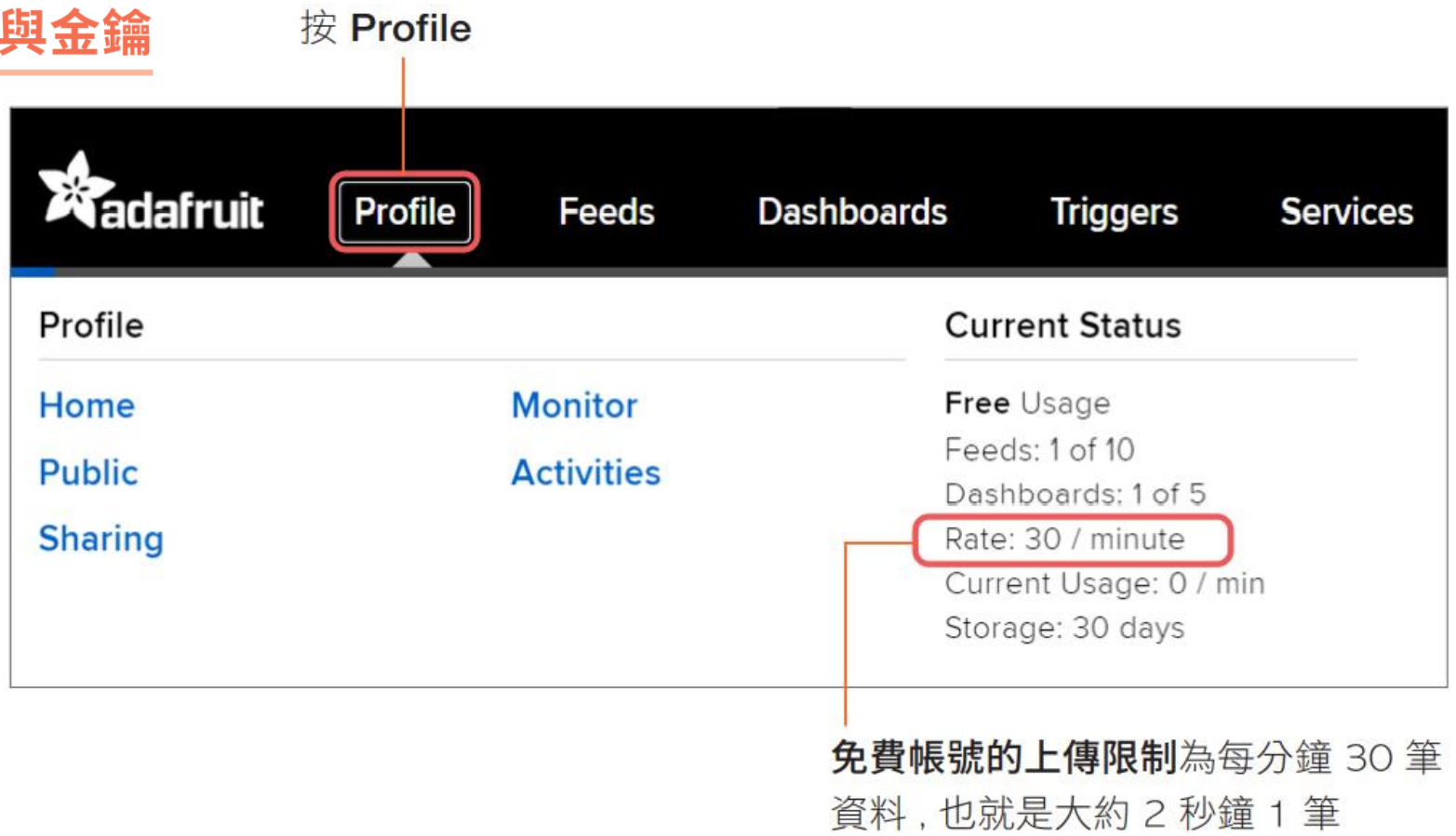
8-4 Adafruit IO

即可看到 **age**
建立成功



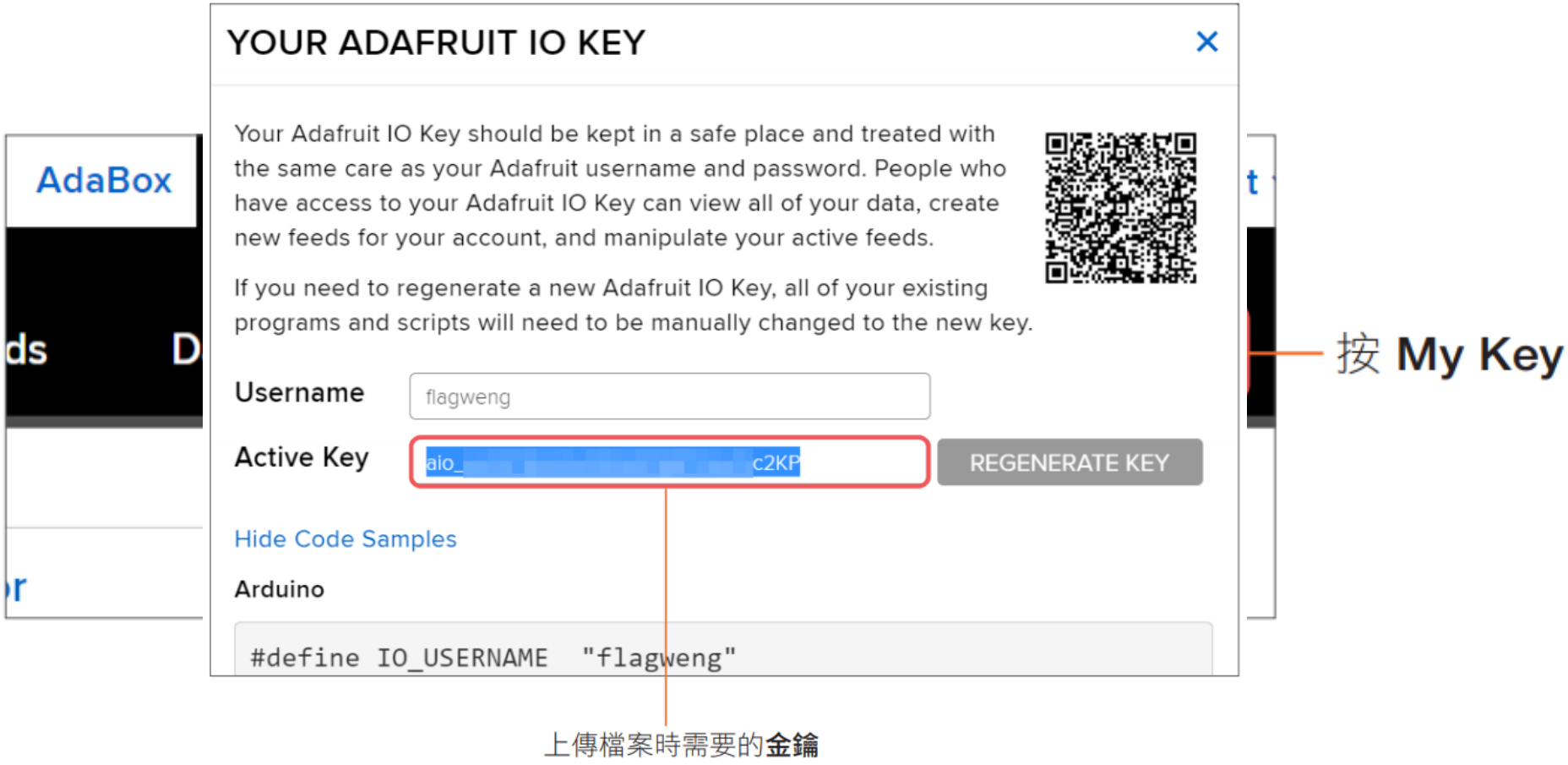
8-4 Adafruit IO

上傳限制與金鑰



免費帳號的上傳限制為每分鐘 30 筆資料，也就是大約 2 秒鐘 1 筆

8-4 Adafruit IO



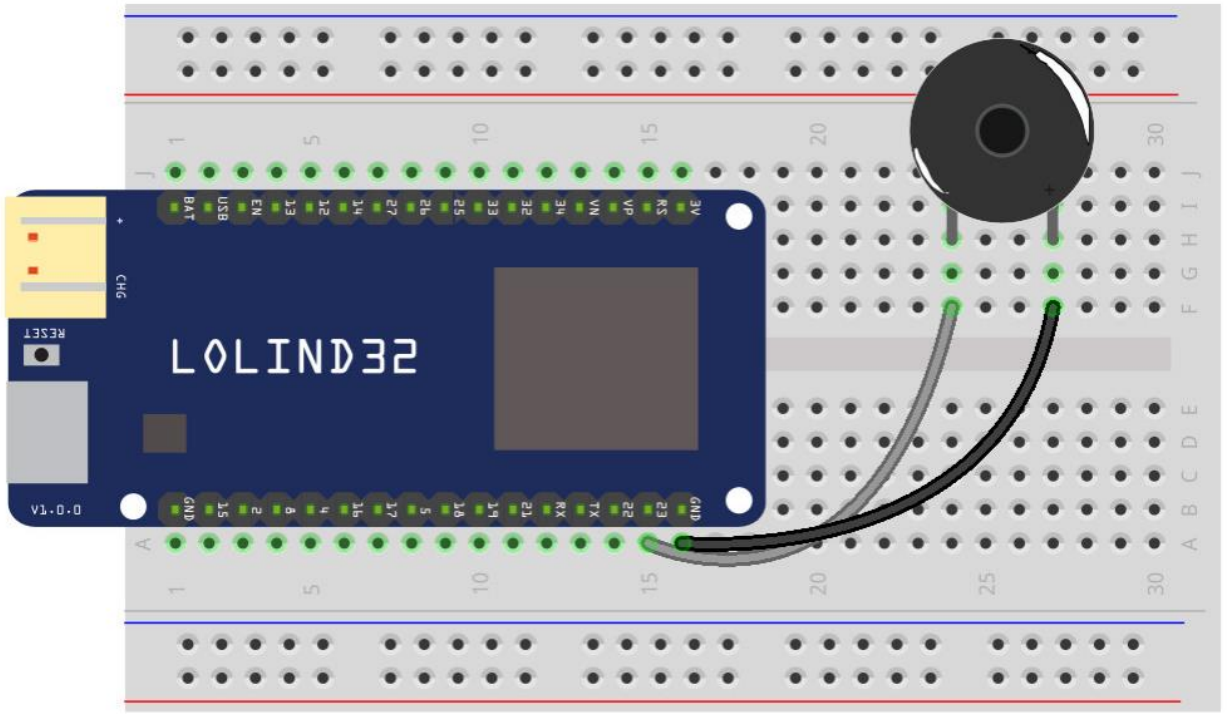
LAB14 遊戲室年齡監控站

實驗目的	有個兒童遊戲室年齡限制為 10 歲，為了避免大人隨意闖入，使用 AI 辨識人臉的年齡，如果年齡超過限制，就會發出警報。
材料	• ESP32 • 蜂鳴器 • 麵包板 • 杜邦線若干條 • 排針

NEXT

LAB14 遊戲室年齡監控站

線路圖



ESP32 腳位	無源蜂鳴器
23	左右腳皆可
GND	左右腳皆可

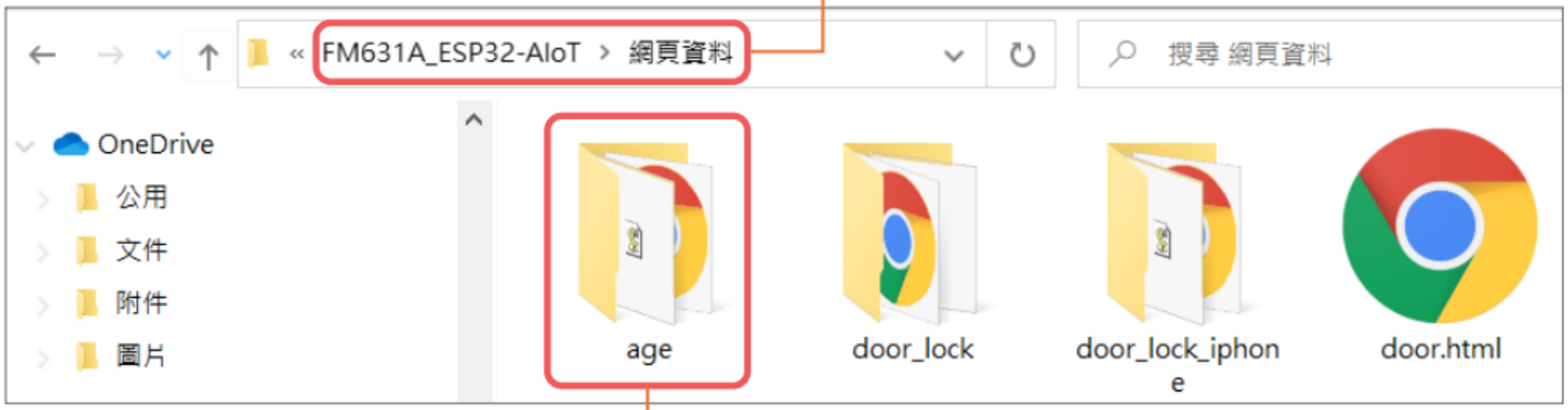
NEXT

LAB14 遊戲室年齡監控站

設計原理

上傳網頁檔案

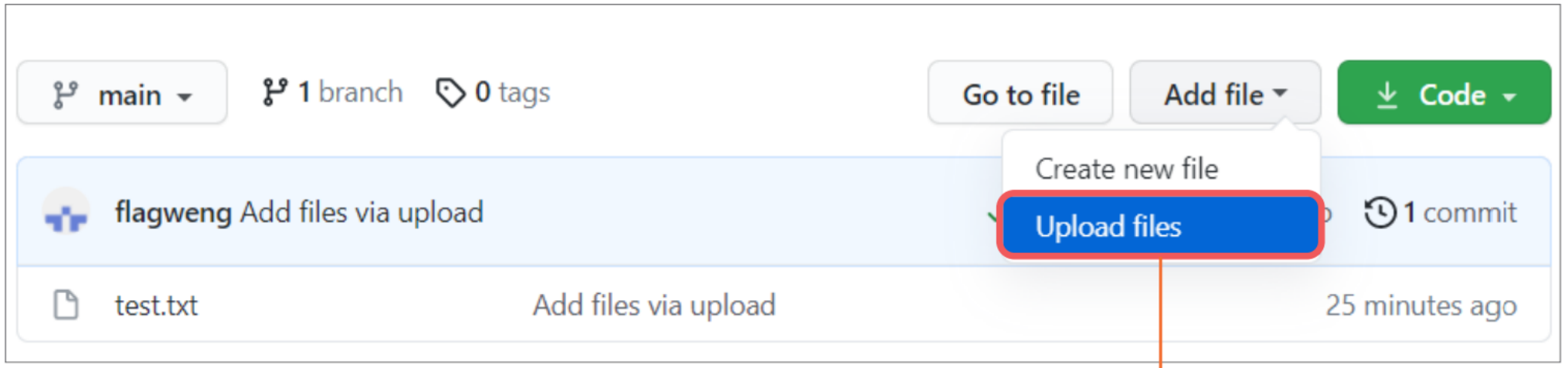
1 開啟**下載檔案**，並切換到『網頁資料』資料夾



年齡探測器的網頁資料

NEXT

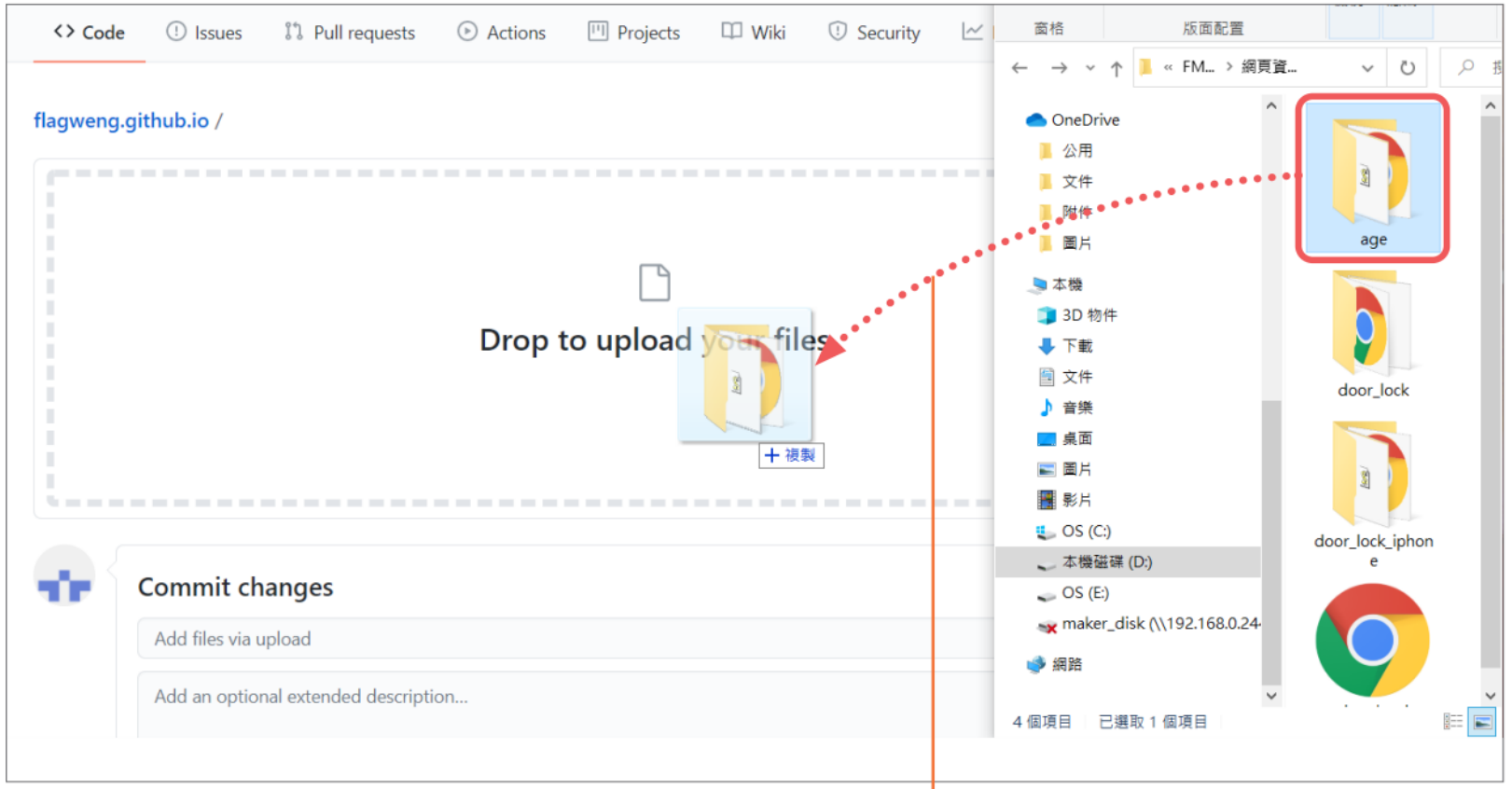
LAB14 遊戲室年齡監控站



2 到 GitHub 網頁按 **Add file/Upload files**

NEXT

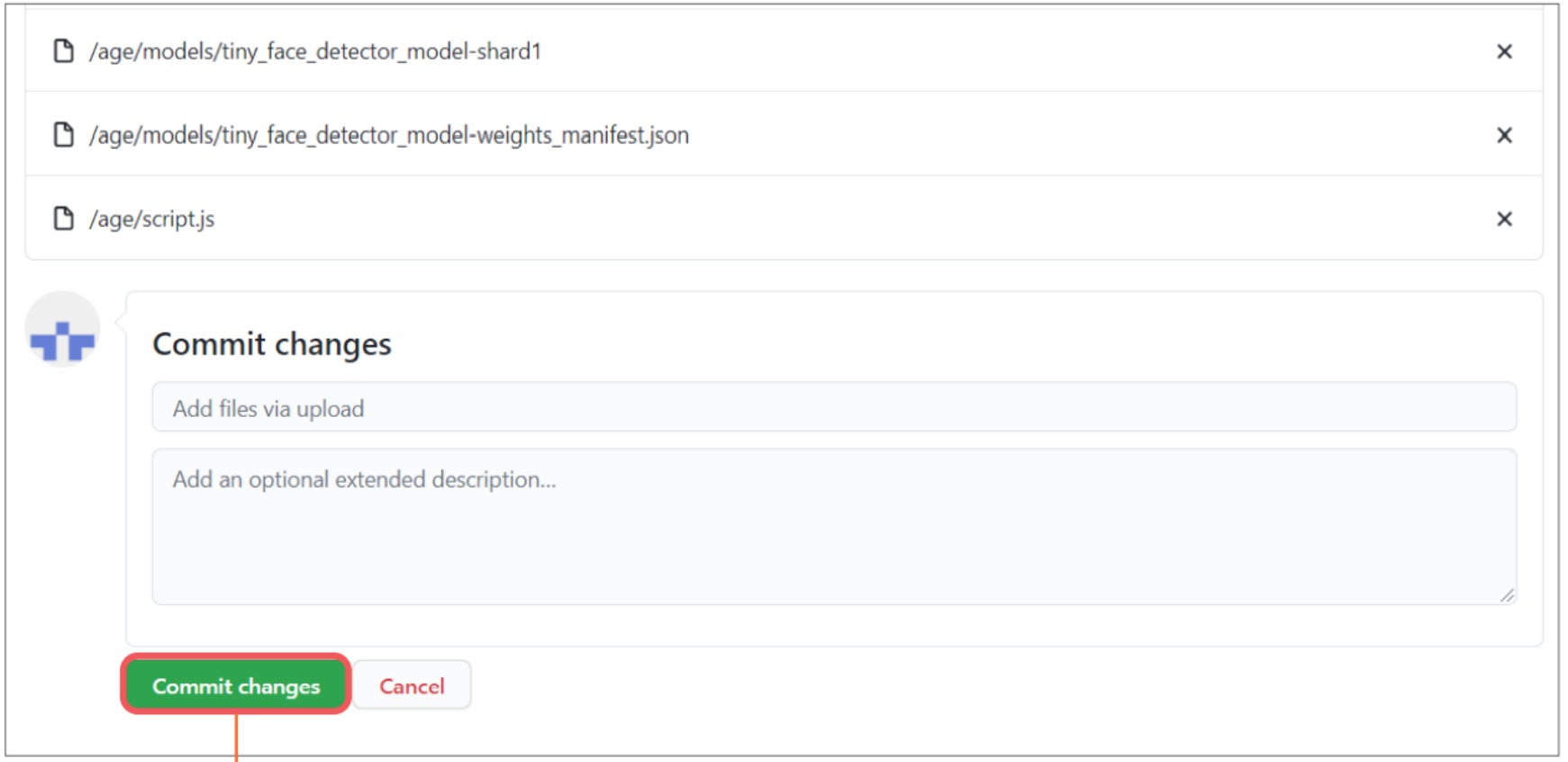
LAB14 遊戲室年齡監控站



3 將 **age 資料夾**拖曳到 GitHub

NEXT

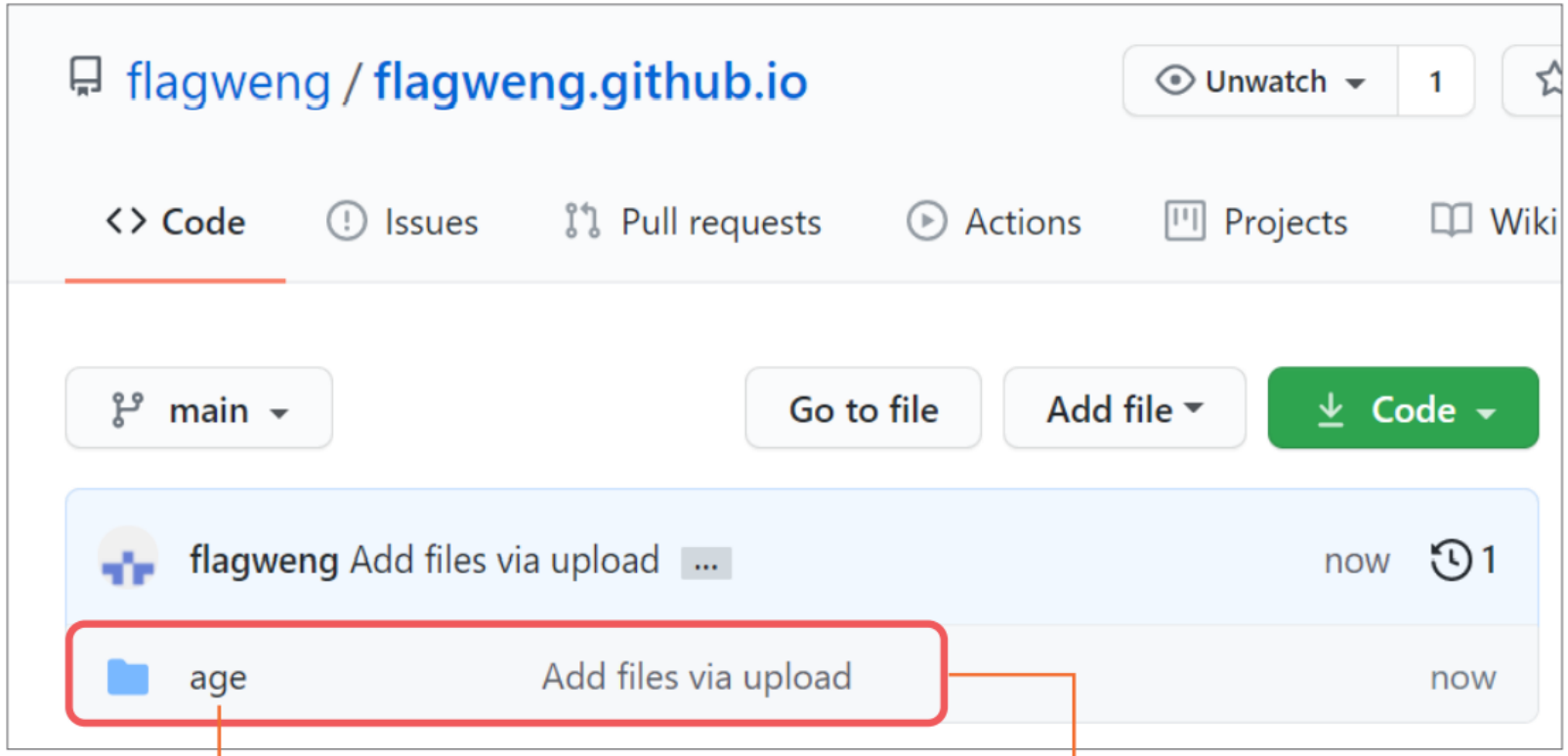
LAB14 遊戲室年齡監控站



4 等待資料全部上傳完畢後，按 **Commit changes**

NEXT

LAB14 遊戲室年齡監控站

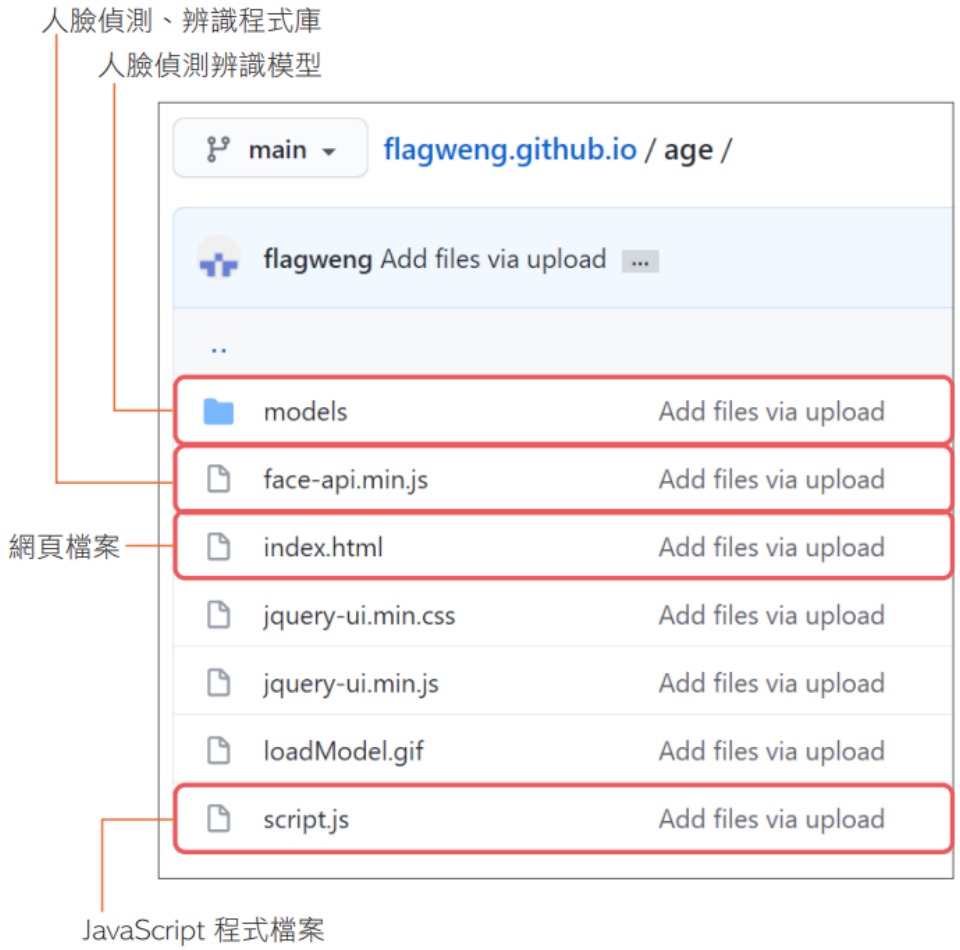


5 按 **age**

看到 age 資料夾已成功上傳到 GitHub

NEXT

LAB14 遊戲室年齡監控站



NEXT

LAB14 遊戲室年齡監控站

人臉偵測

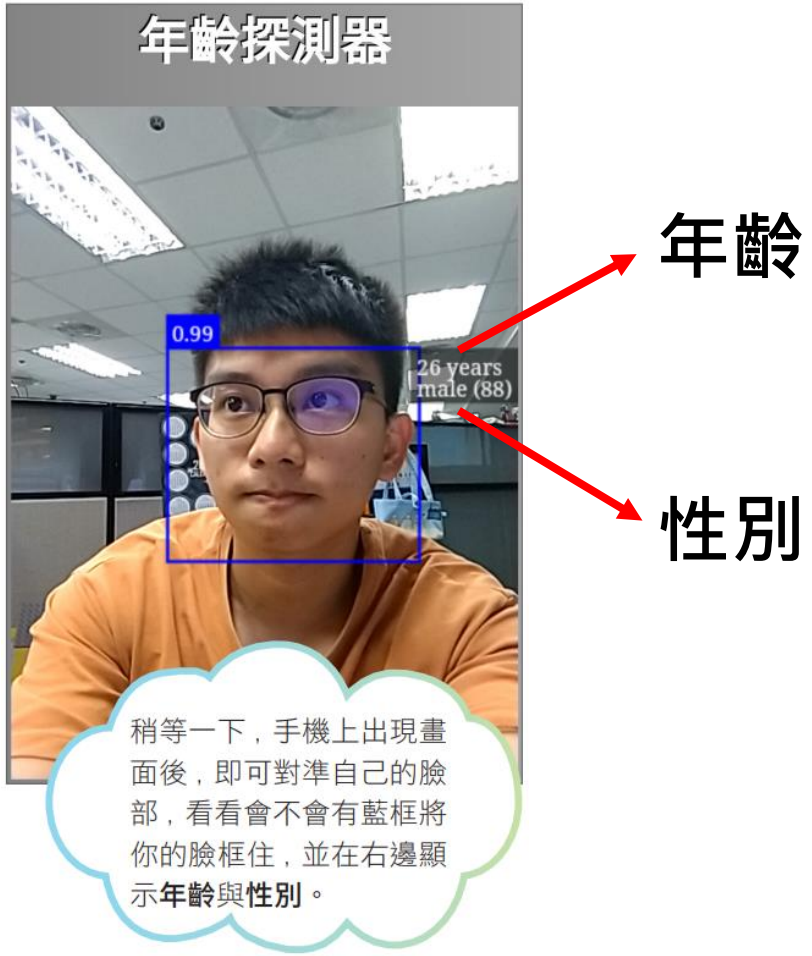
網址列輸入：

```
https://使用者名稱.github.io/age/index.html
```

⚠ 請填入自己 github 帳號的使用者名稱。

NEXT

LAB14 遊戲室年齡監控站



年齡

性別

NEXT

LAB14 遊戲室年齡監控站

上傳資料至 AIO

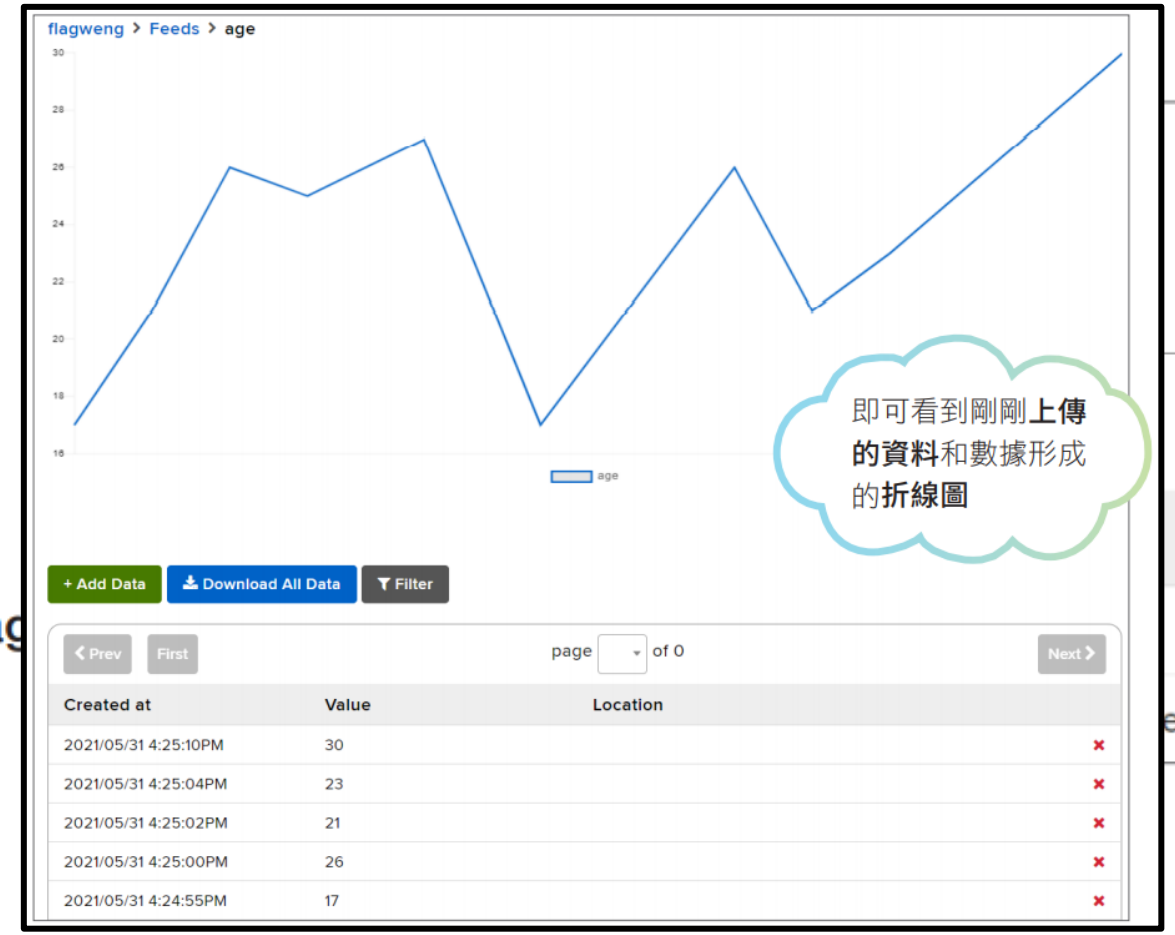
8-4節內容

⚠ 輸入使用者名稱和金鑰後，網頁會自動記住，
就算重新開啟瀏覽器也不用再次輸入囉！

NEXT

LAB14 遊戲室年齡監控站

到 AIO 頁面點選 ag

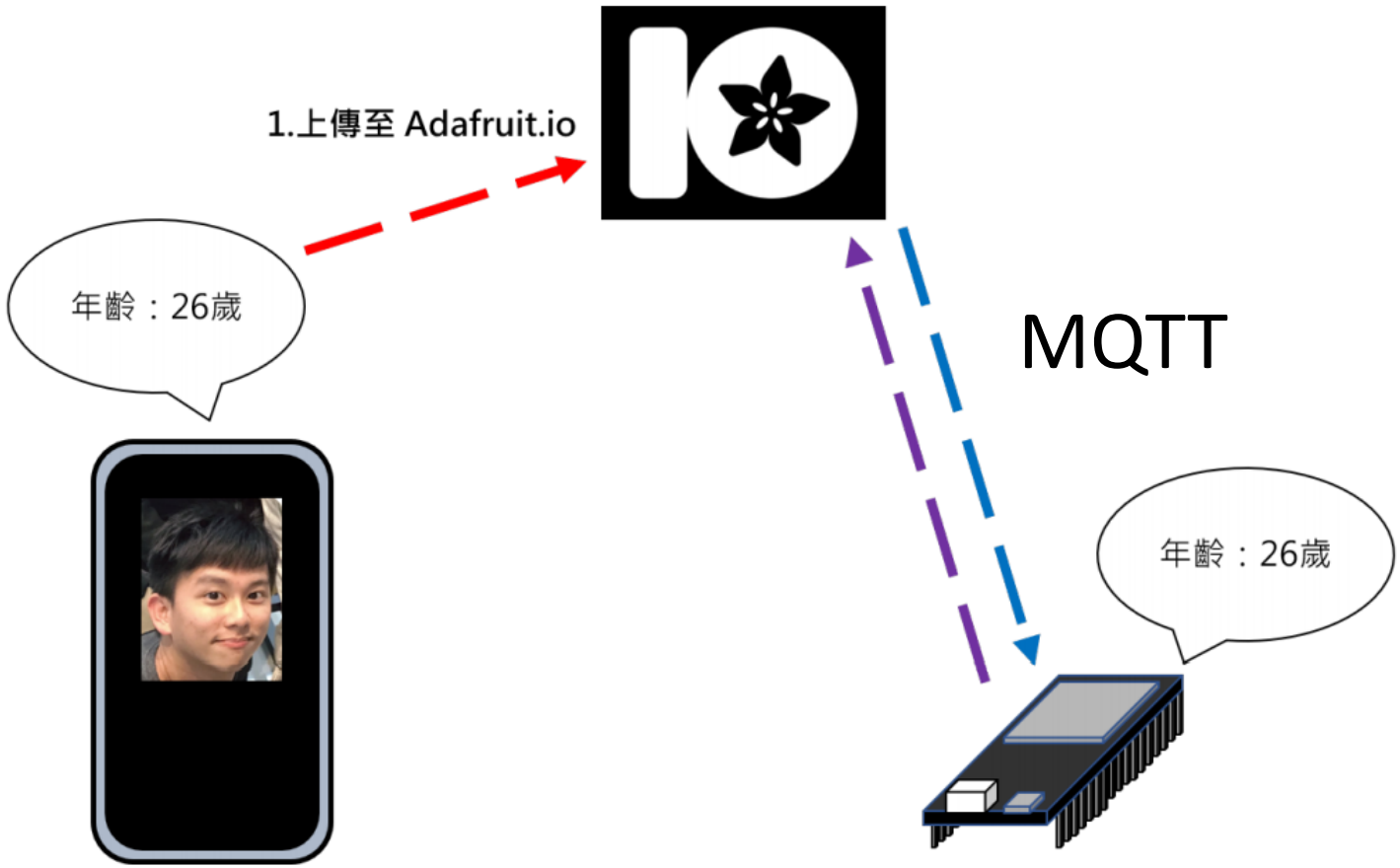


e-feed

NEXT

LAB14 遊戲室年齡監控站

MQTT



NEXT

LAB14 遊戲室年齡監控站

MQTT 通訊協定簡介

MQTT 是一種中介服務

1. MQTT 中介伺服器 (broker)：負責轉送訊息
2. 發佈端 (publisher)：專門發送資料到 MQTT 中介伺服器
3. 訂閱端 (subscriber)：專門接收資料

NEXT

LAB14 遊戲室年齡監控站

MQTT 中介伺服器

主機網址	io.adafruit.com
連接埠編號	1883
使用者帳號	AIO 使用者名稱
密碼	AIO 金鑰

NEXT

LAB14 遊戲室年齡監控站

umqtt 模組

匯入類別：

```
from umqtt.robust import MQTTClient
```

建立物件：

```
client = MQTTClient(client_id="age",              # 用戶端識別名稱
                    server = io.adafruit.com,     # 中介伺服器網址
                    user = "AIO 使用者名稱",      # 使用者名稱
                    password = "AIO 金鑰")        # 名稱
```

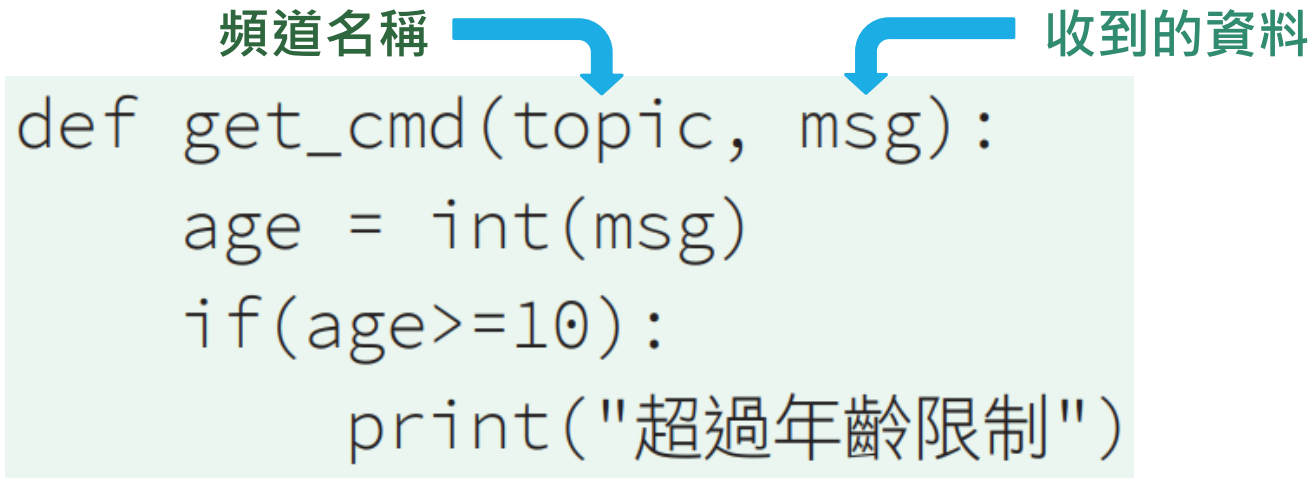
NEXT

LAB14 遊戲室年齡監控站

連接 MQTT 中介伺服器：

```
client.connect()
```

收到訂閱資料時會自動呼叫的函式：



NEXT

LAB14 遊戲室年齡監控站

註冊為收到訂閱資料時的處理函式：

```
client.set_callback(get_cmd)
```

向伺服器訂閱頻道：

AIO 的頻道格式

```
client.subscribe(b"使用者名稱/feeds/age")
```

不斷檢查是否有新資料：

```
while True:
    client.check_msg()
```

NEXT

LAB14 遊戲室年齡監控站

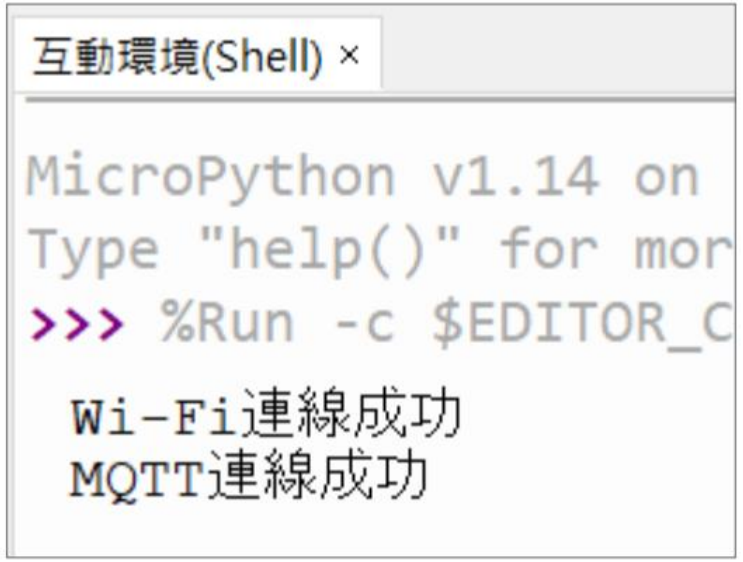
程式設計

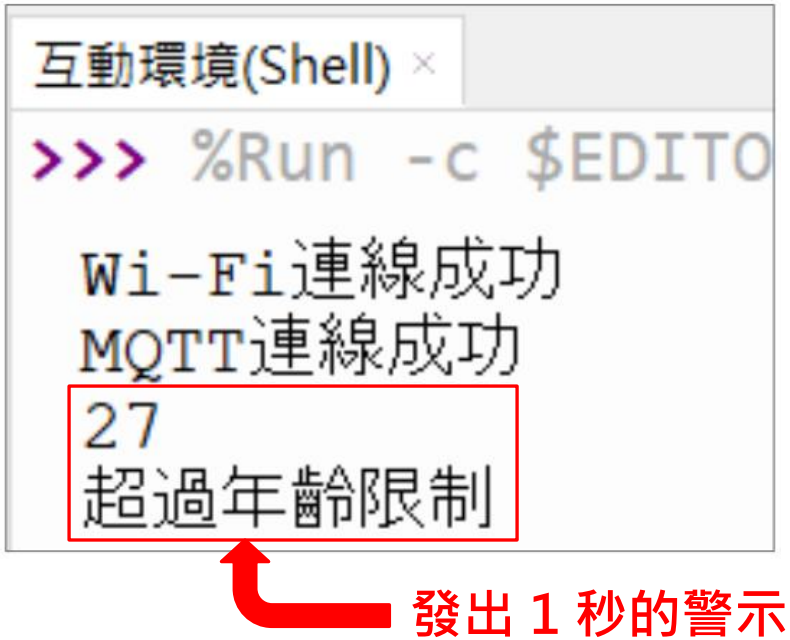
```
01  from umqtt.robust import MQTTClient
02  from machine import Pin,PWM
03  import network
04  import time
05
06  # 連線至無線網路
07  sta=network.WLAN(network.STA_IF)
08  sta.active(True)
09  sta.connect('無線網路名稱','無線網路密碼')
10  while not sta.isconnected() :
11      pass
12  print('Wi-Fi連線成功')
13
14  # mqtt參數
```

NEXT

LAB14 遊戲室年齡監控站

實測





發出 1 秒的警示聲

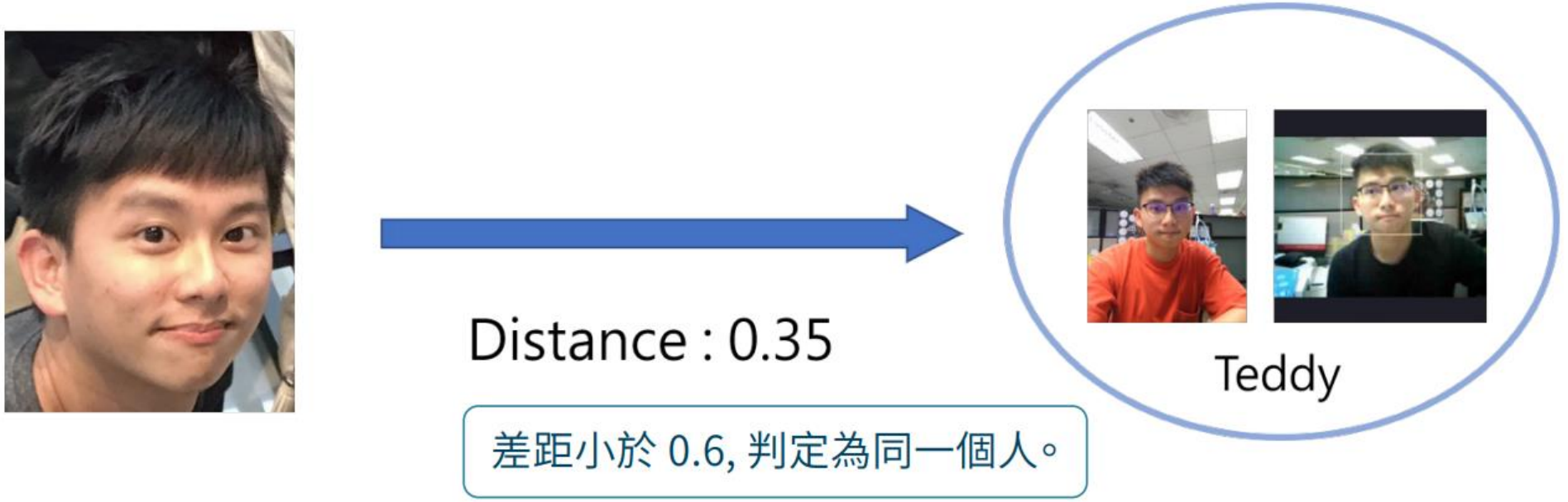
8-5 人臉辨識

face-api.js 除了**人臉偵測**外, 還可以**辨識人臉**。辨識人臉包含**辨識身分**和**差距(Distance)**。

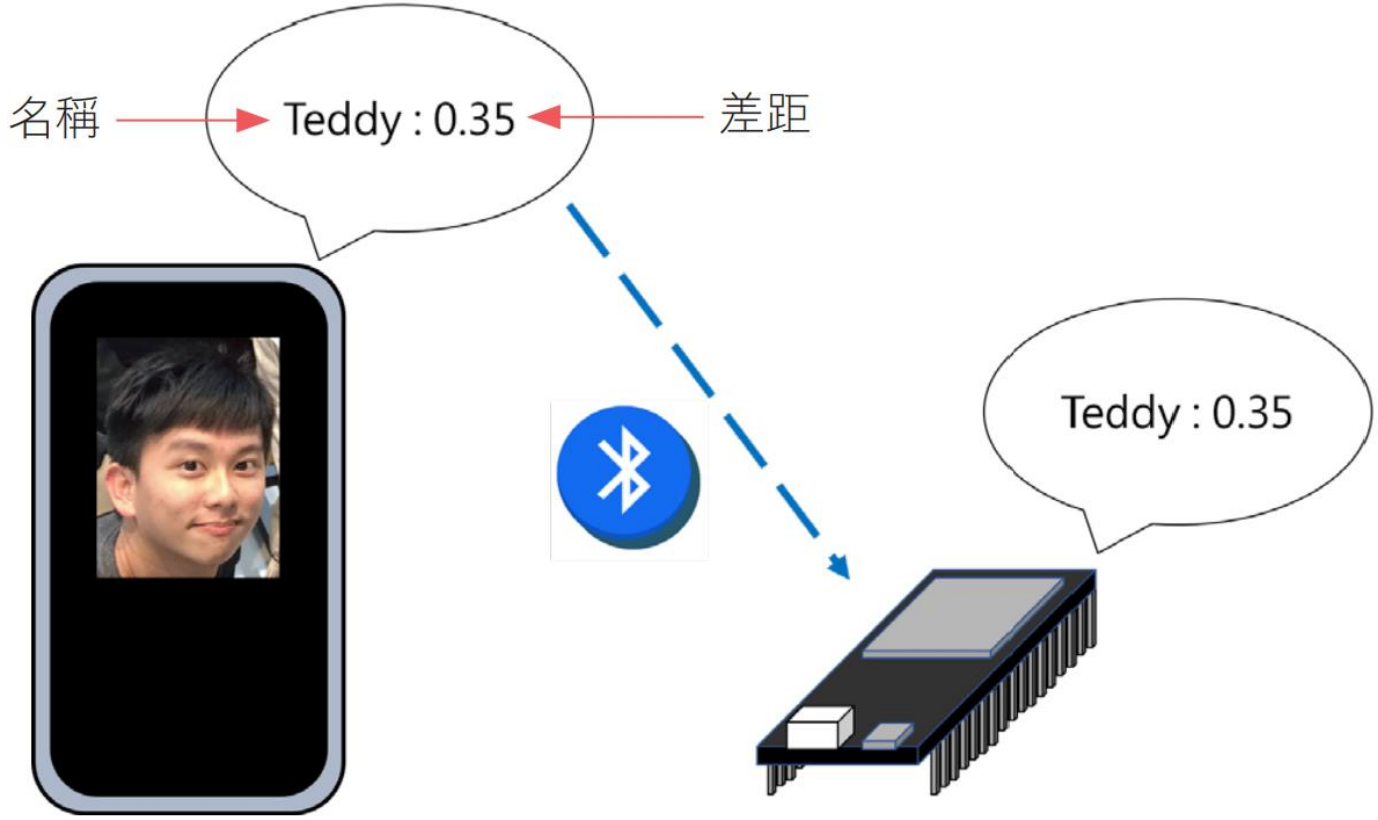
辨識身分：

8-5 人臉辨識

差距(Distance) 。數值會介於 0 到 1 之間：



8-6 網頁藍牙 Web Bluetooth



iPhone 使用者請參考：
https://hackmd.io/@flagmaker/S1ReszUcd

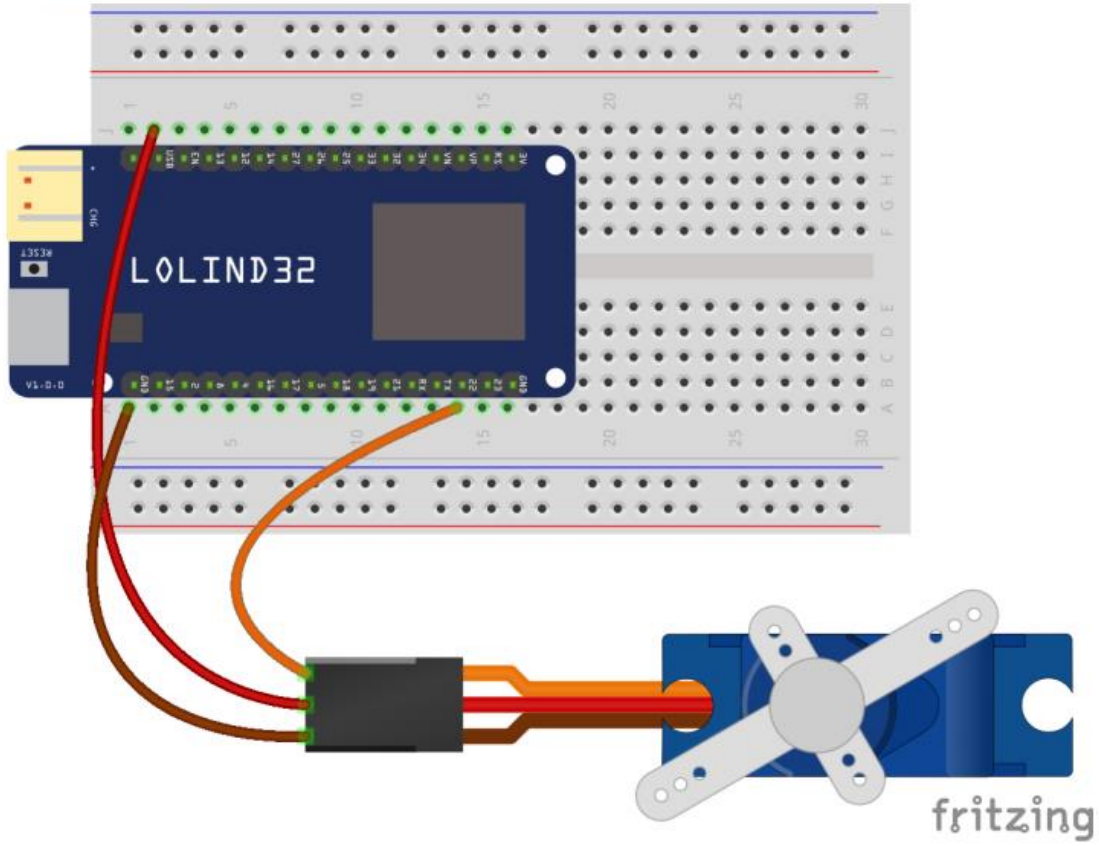
LAB15 智慧門鎖

實驗目的	將人臉辨識的結果透過『網頁藍牙』傳送給 ESP32, 如果身份正確且差距夠小就會轉動伺服馬達。
材料	• ESP32 • 伺服馬達 • 麵包板 • 杜邦線若干條 • 排針 ⚠ 同 **LAB02**

NEXT

LAB15 智慧門鎖

線路圖



ESP32 腳位	伺服馬達
USB	紅線
GND	棕線
22	橘線

⚠ USB 腳位會輸出 5V 電壓。

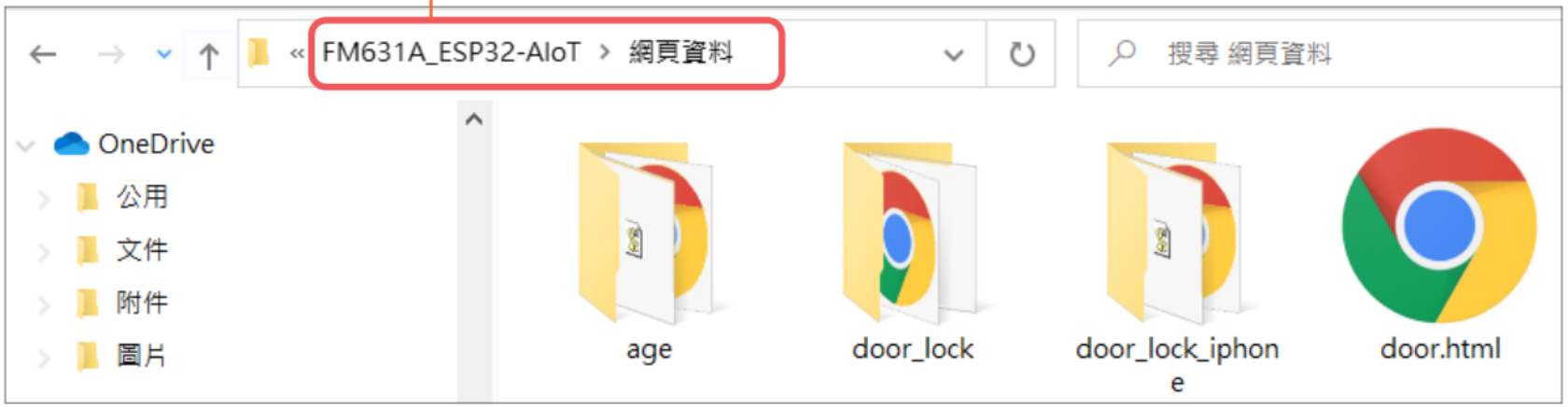
NEXT

LAB15 智慧門鎖

設計原理

建立放置人臉照片的資料夾

1 開啟**下載範例檔案的資料夾**



NEXT

LAB15 智慧門鎖

2 切換到『網頁資料 / door_lock/images』資料夾

3 以要識別的人名建立**資料夾**，本書範例為 Teddy 和 Chuan 兩個人。

NEXT

LAB15 智慧門鎖

每個名稱資料夾內，請都放入 3 張圖片，並且一定要各自命名為 1、2、3 (圖片格式限制為 jpg 和 png)

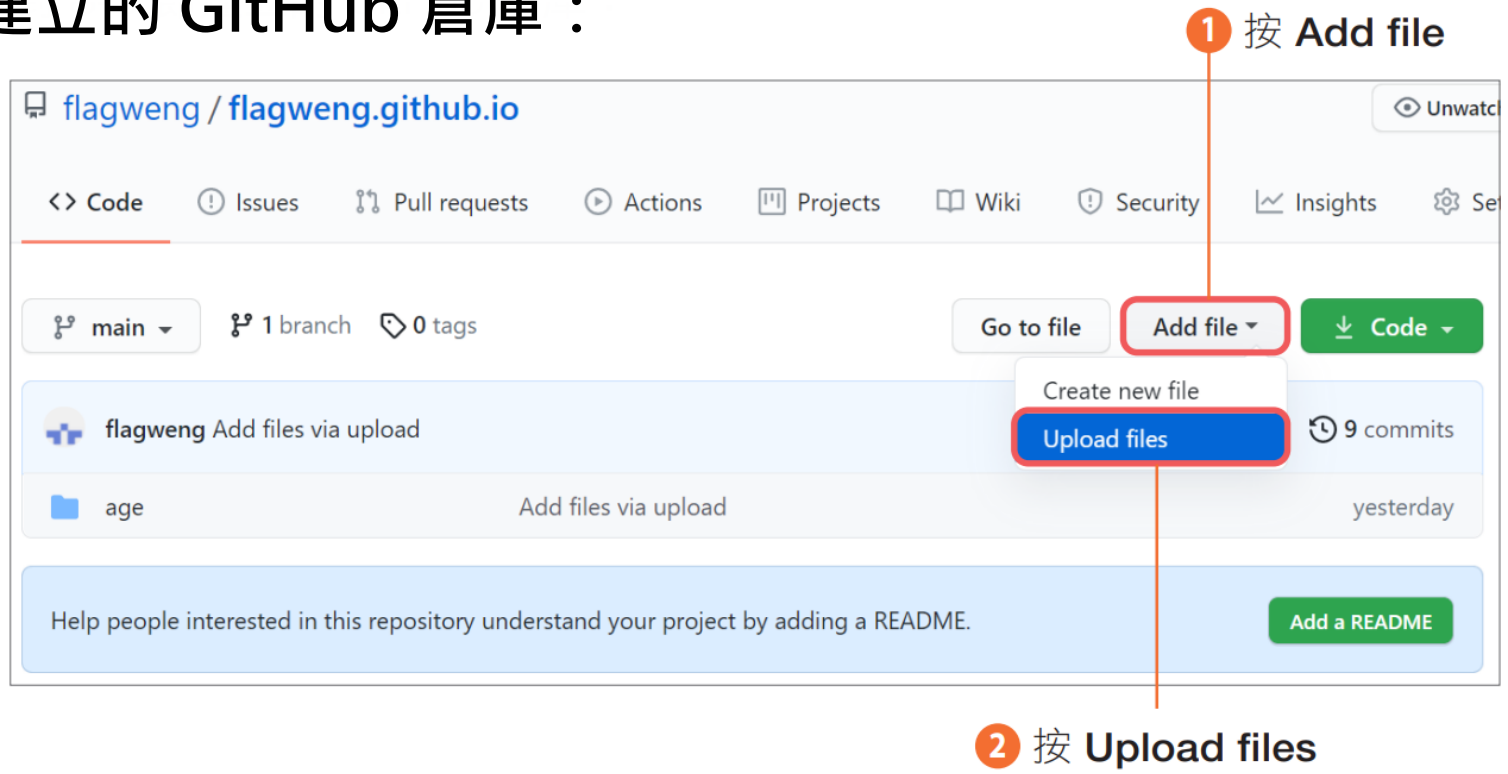
⚠ 我們提供的範例網頁中，每個名稱只會讀取**檔名為 1、2、3 的圖片**，所以可以多放圖片 (程式不會讀取到), 不能少放。如果想要網頁多讀取一些圖片，可以到 door_lock 資料夾裡的 script.js 內 (162 行) 更改。

NEXT

LAB15 智慧門鎖

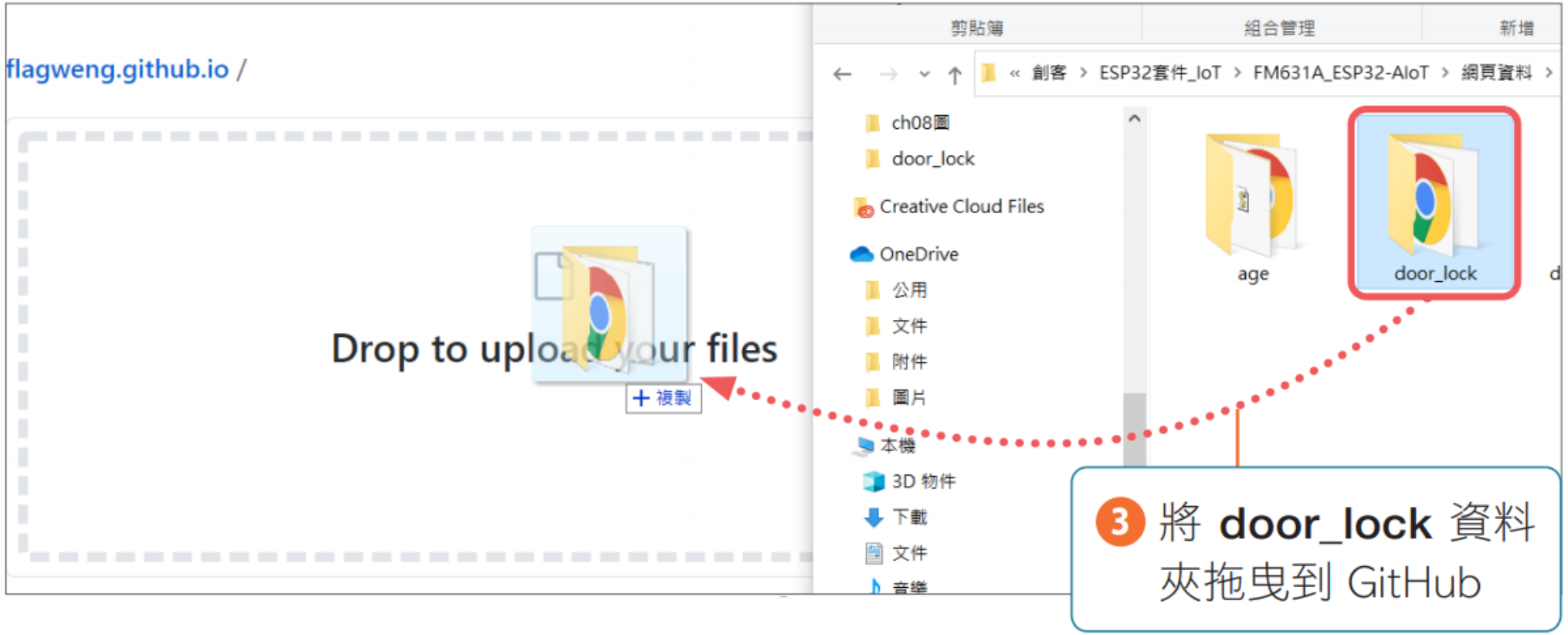
上傳網頁檔案

到 8-3 節建立的 GitHub 倉庫：



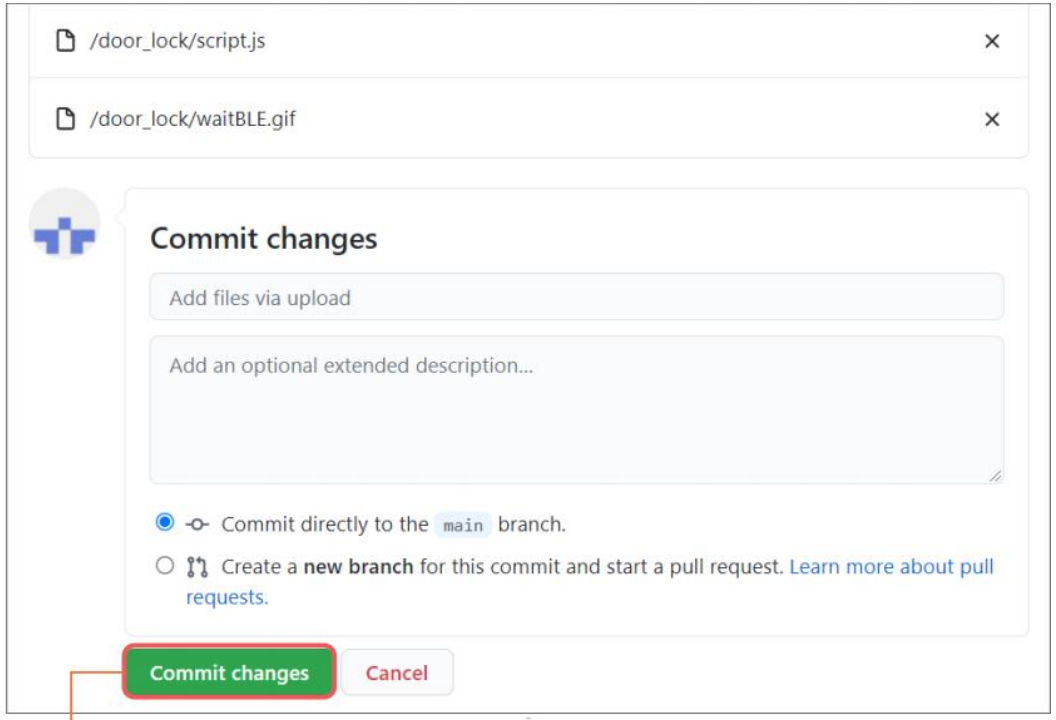
NEXT

LAB15 智慧門鎖

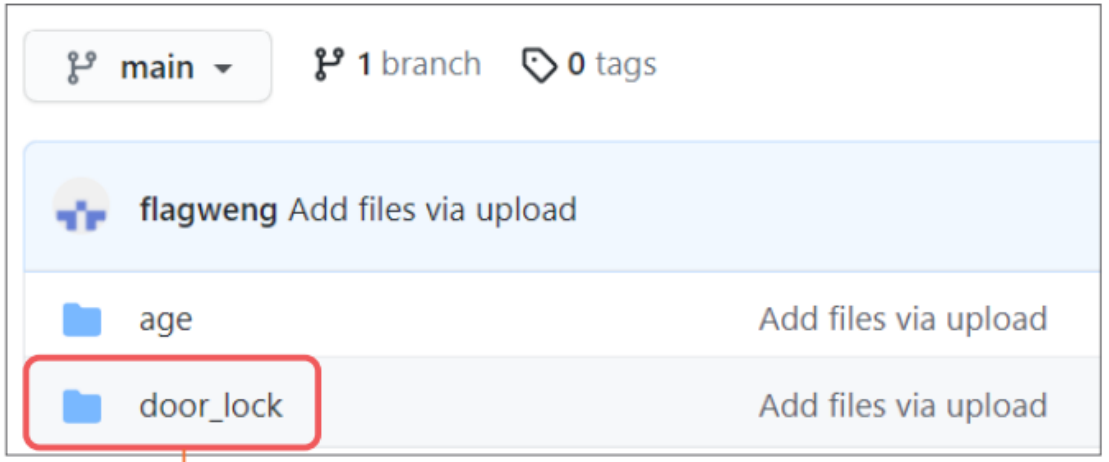


NEXT

LAB15 智慧門鎖



4 資料全部上傳完畢後，按 Commit changes



看到 door_lock 資料夾已成功上傳到 GitHub

NEXT

LAB15 智慧門鎖

開啟網頁

網址列輸入：

https://使用者名稱.github.io/door_lock/index.html

LAB15 智慧門鎖

看到此畫面即可

Teddy,Chuan

NEXT

LAB15 智慧門鎖

ESP32 解析資料

瀏覽器傳送以下格式的資料給 ESP32：

名稱:差距　例如 Teddy:0.35

建立 BLE_UART 物件, 並取得回傳值：

```
>>>  from ble_uart import BLE_UART
>>>  ble = BLE_UART("藍牙門鎖")        # 藍牙裝置取名為藍牙門鎖
>>>  getValue = ble.get()
>>>  getValue
Teddy:0.35   ←  回傳值
```

NEXT

LAB15 智慧門鎖

將『名稱』與『差距』切割：

```
>>> get_split = getValue.split(":")
>>> get_split
['Teddy','0.35']
```

切割完後, 使用索引分別取出值：

```
>>> get_split[0]
'Teddy'
>>> get_split[1]
'0.35'
```

NEXT

LAB15 智慧門鎖

開啟範例程式 LAB15
並更改第 7 行的名稱列表
和第 12 行的藍牙名稱

```
19          get_split = getValue.split(":")
20          # 名稱
21          name = get_split[0]
22          # 距離
23          dis = float(get_split[1])
24          print(name, dis)
```

NEXT

LAB15 智慧門鎖

實測

NEXT

LAB15 智慧門鎖

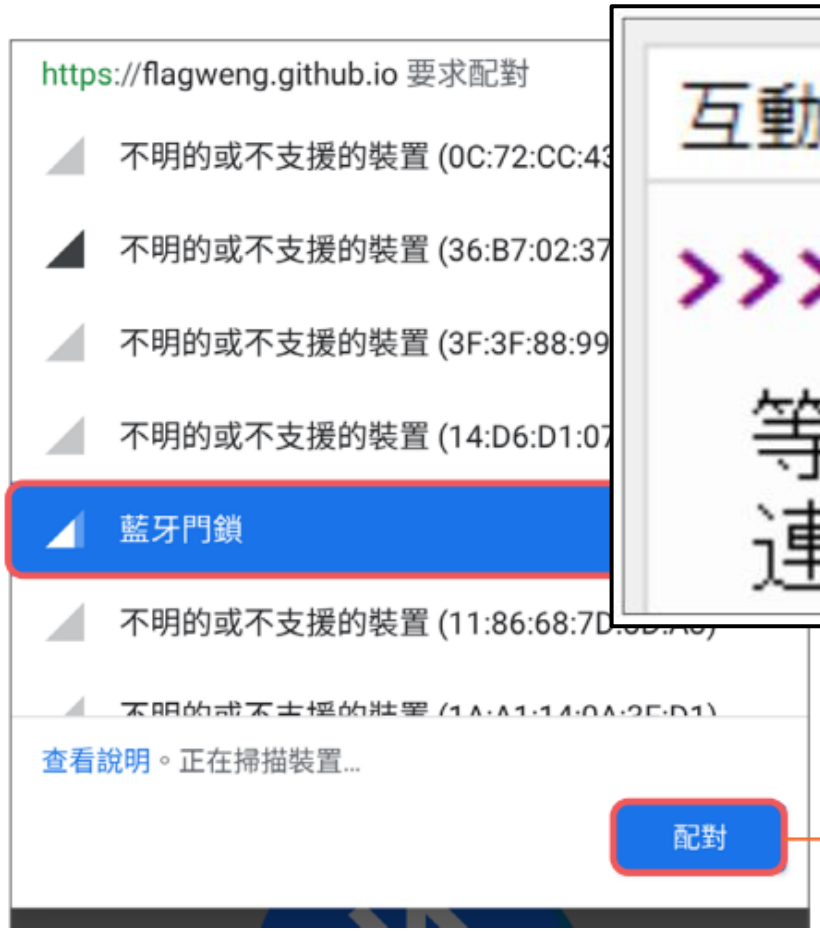
實測

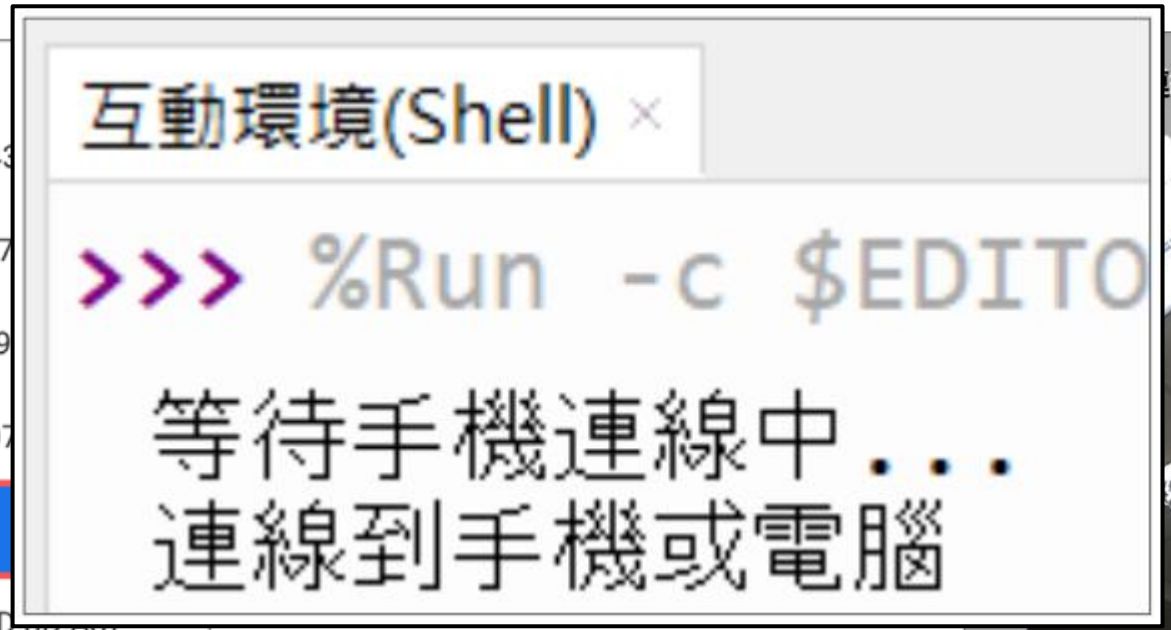
回到瀏覽器：

按按鈕前請先確認已經輸入名稱

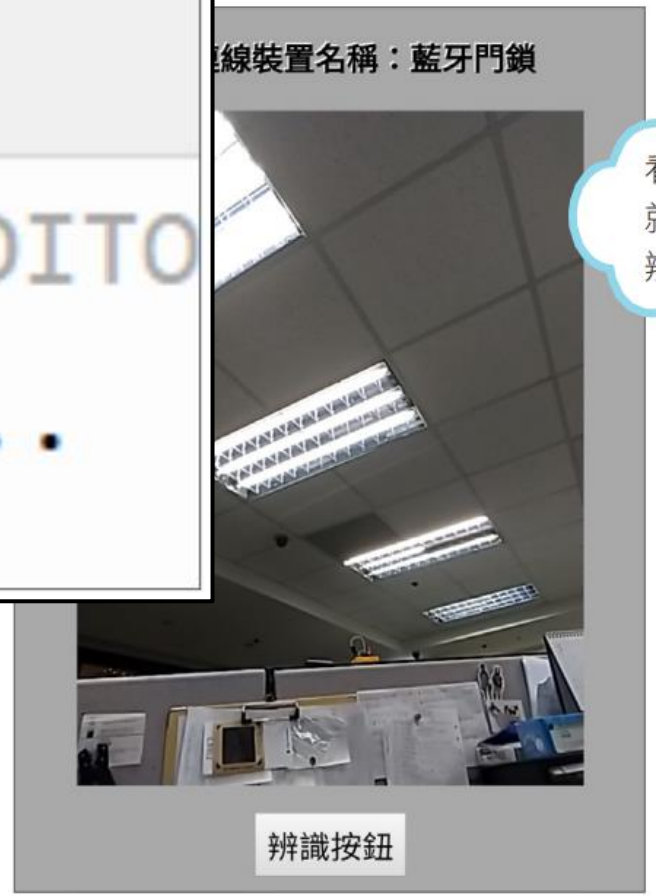
NEXT

LAB15 智慧門鎖





4 按**配對**

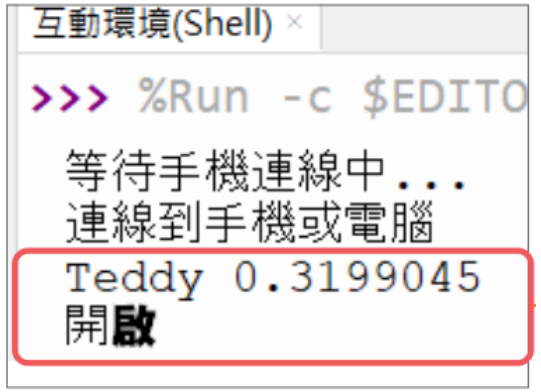


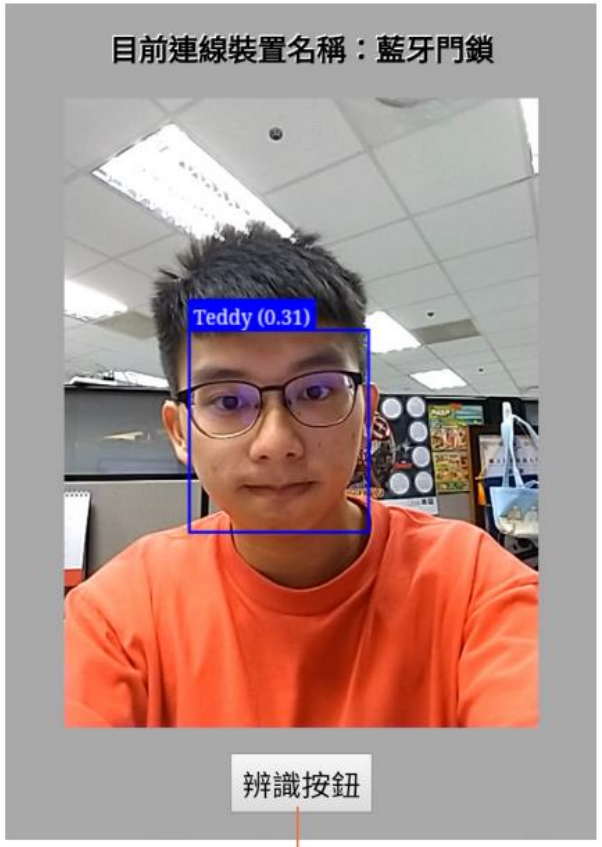
看到此畫面
就可以開始
辨識了

NEXT

LAB15 智慧門鎖

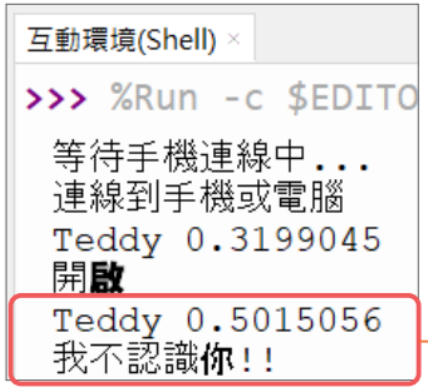
辨識成功





將攝影頭對準臉，按**辨識按紐**，如果成功辨識到人臉，顯示**藍框**、**名稱**和**差距**

辨識失敗

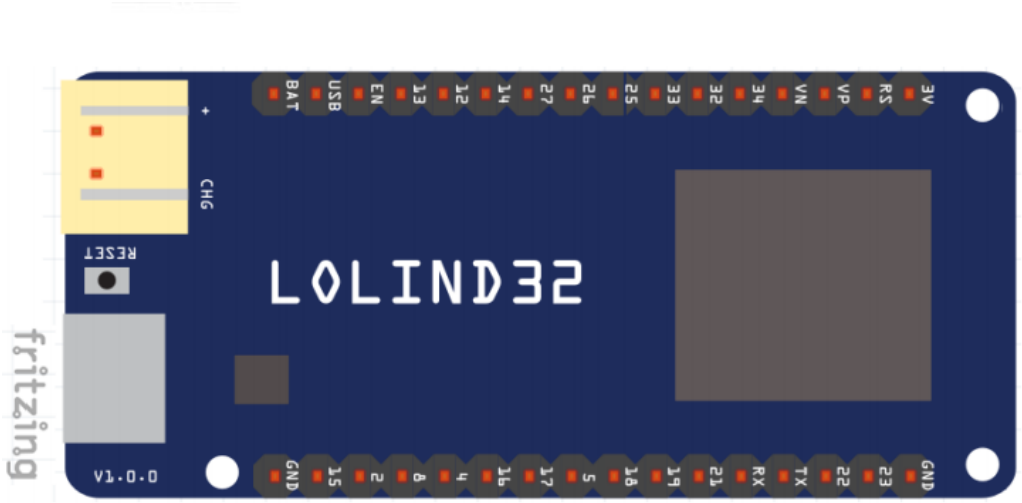


將攝影頭對準臉，
按**辨識按紐**

目錄

⚠ 由於 MicroPython 目前在 BLE 的功能並不完整，**本章實驗不適用於 iPhone 及 Mac 電腦**，請改用 Android 手機或是 Windows/Linux 筆記型電腦測試。

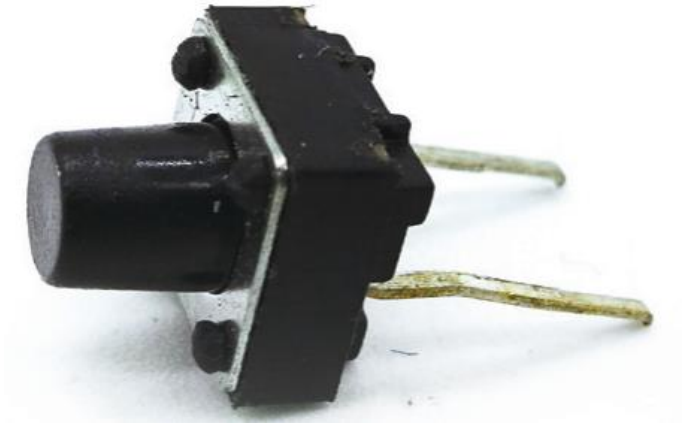
9-1 HID



鍵盤。HID 裝置

電腦。HID 主機

9-2 按鈕



沒有按下時不導通

按下時導通

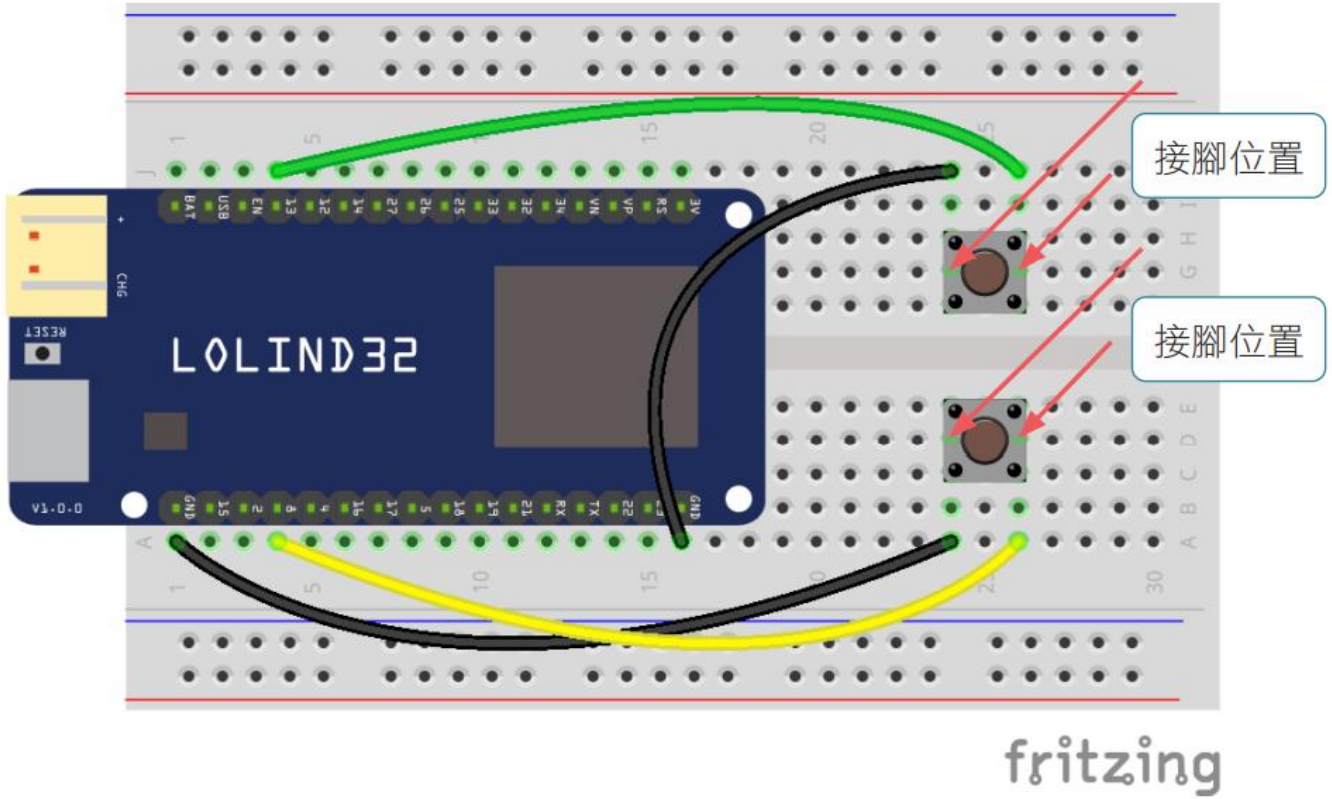
LAB16 按鈕開關測試

實驗目的	使用 ESP32 的輸入腳位讀取按鈕
材料	• ESP32 • 按鈕 × 2 • 麵包板 • 杜邦線若干條 • 排針

NEXT

LAB16 按鈕開關測試

線路圖



ESP32	上按鈕	下按鈕
13	右接腳	×
0	×	右接腳
GND	左接腳	左接腳

⚠ 按鈕沒有極性。

NEXT

LAB16 按鈕開關測試

設計原理

匯入模組並建立物件：

設定成輸入模式

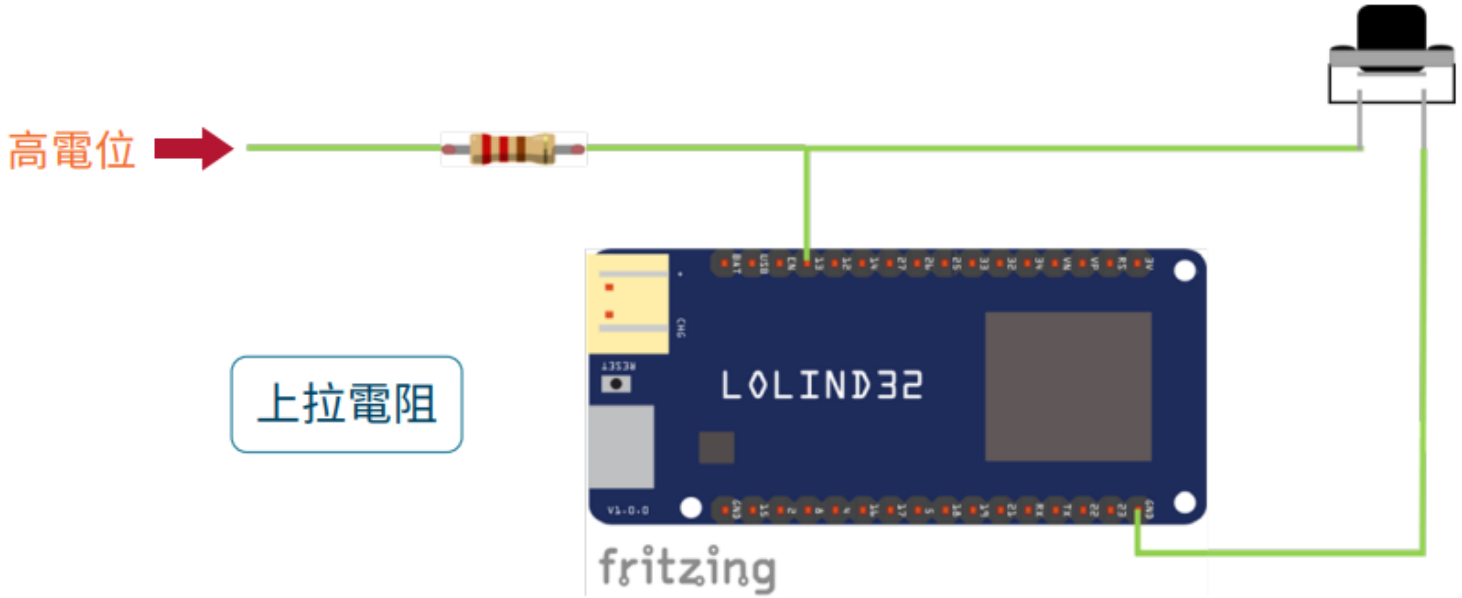
```
>>>  from machine import Pin
>>>  button_up = Pin(13, Pin.IN)
```

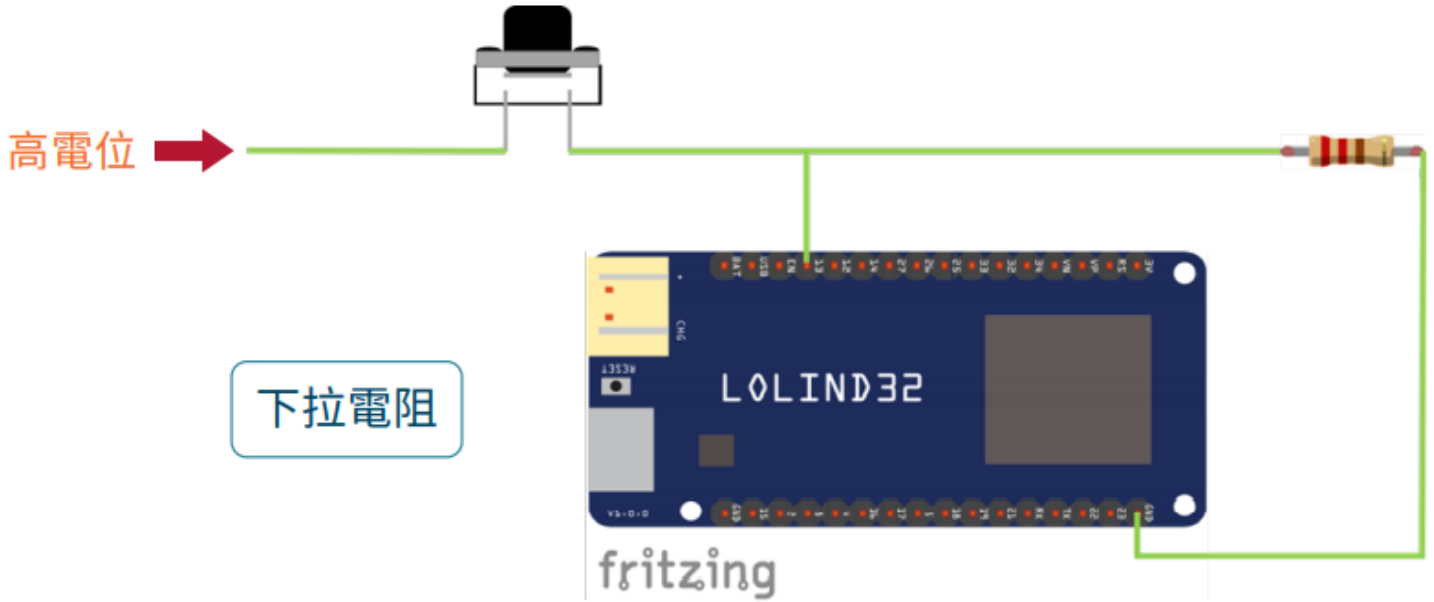
讀取電位高低：

```
>>>  button_up.value()
```

NEXT

LAB1





NEXT

LAB16 按鈕開關測試

啟用內建上拉電阻：

```
button_up = Pin(13, Pin.IN, Pin.PULL_UP)
```

按下按鈕 → 低電位(0)

沒按按鈕 → 高電位(1)

NEXT

LAB16 按鈕開關測試

程式設計

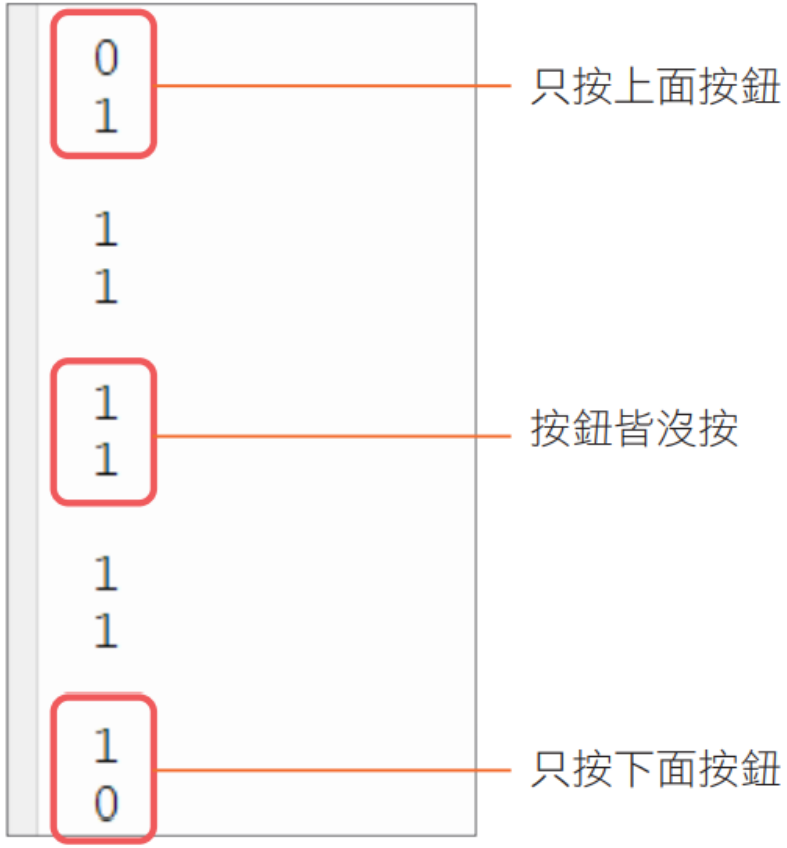
```
01  from machine import Pin
02  import time
03
04  # 上面按鈕
09  while True:
10      # 讀取上面按鈕的值
11      print(button_up.value())
12      # 讀取下面按鈕的值
13      print(button_down.value())
14      print()
15      time.sleep(0.1)
```

開啟範例程式 LAB16

NEXT

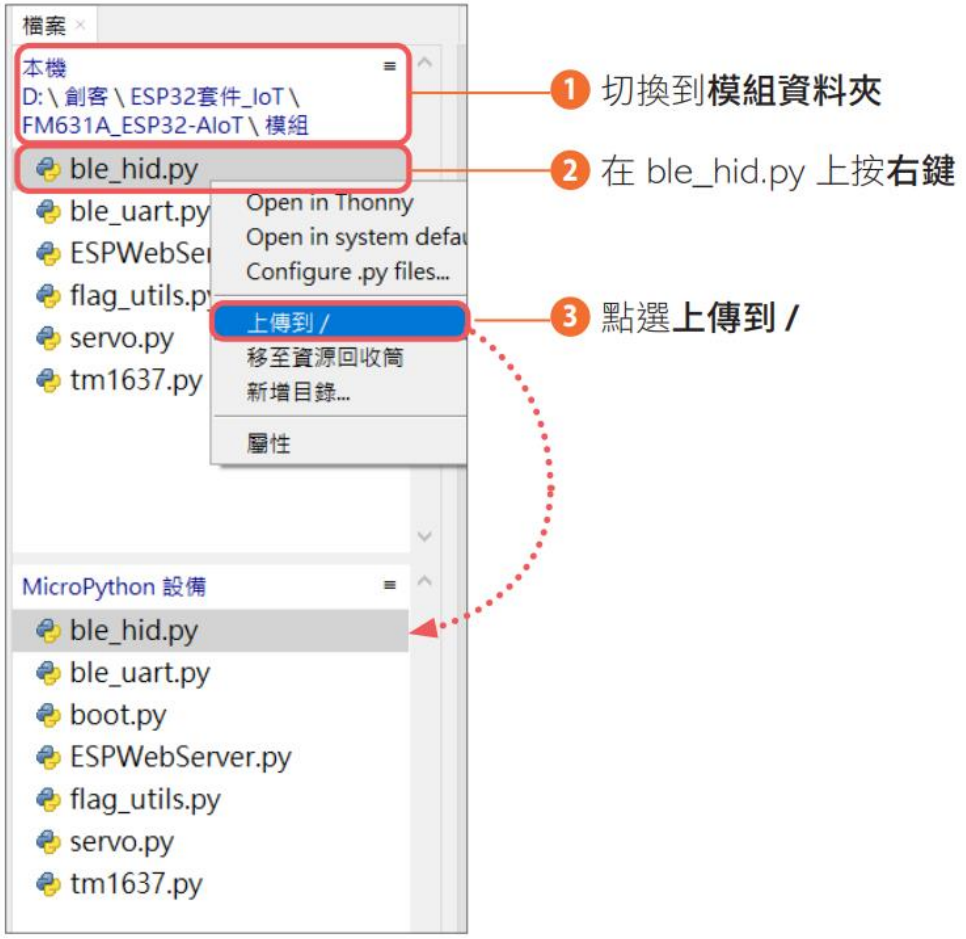
LAB16 按鈕開關測試

實測



9-3 藍牙鍵盤

上傳模組：


檔案
本機
D:\創客\ESP32套件_IoT\
FM631A_ESP32-AIoT\模組
ble_hid.py
ble_uart.py
ESPWebSer
flag_utils.py
servo.py
tm1637.py
Open in Thonny
Open in system defau
Configure .py files...
上傳到 /
移至資源回收筒
新增目錄...
屬性
MicroPython 設備
ble_hid.py
ble_uart.py
boot.py
ESPWebServer.py
flag_utils.py
servo.py
tm1637.py
1 切換到模組資料夾
2 在 ble_hid.py 上按右鍵
3 點選上傳到 /

9-3 藍牙鍵盤

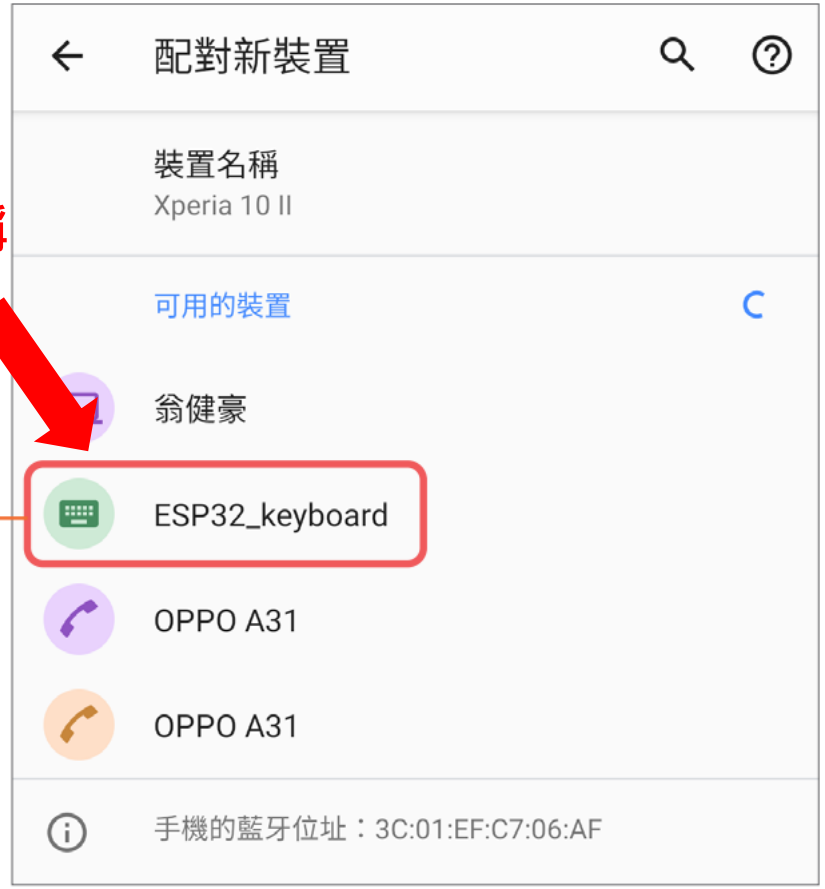
匯入模組並建立物件：

```
>>> from ble_hid import BLE_HID
>>> keyboard= BLE_HID ("ESP32_keyboard")
```

自行更改名稱, 注意不要超過 18 個字元

9-3 藍牙鍵盤

尋找自己改的名稱



ESP32 裝置

點選**配對**

⚠ 如果手機顯示**無法與裝置配對**，可以重新啟動藍牙或手機。

NEXT

9-3 藍牙鍵盤

9-3 藍牙鍵盤

```
>>>  keyboard.send_char('a')    ← 傳送單一字元。等同按下鍵盤的 a
>>>  keyboard.send_str('aBc')   ← 傳送多個字元。
>>>  keyboard.changeLanguage() ← 手機切換語言。等同按下鍵盤的
                                  Shift + Space
>>>  keyboard.screenShot()     ← 截圖。等同按下鍵盤的 Print Screen
>>>  keyboard.photograph()     ← 手機照相。等同按下鍵盤的 Enter
```

LAB17 截圖神器

實驗目的	將 ESP32 變成藍牙鍵盤，並模仿按下鍵盤的 Print Screen 來截圖。
材料	同 LAB16

接線圖

同 LAB16

NEXT

LAB17 截圖神器

設計原理

螢幕截圖

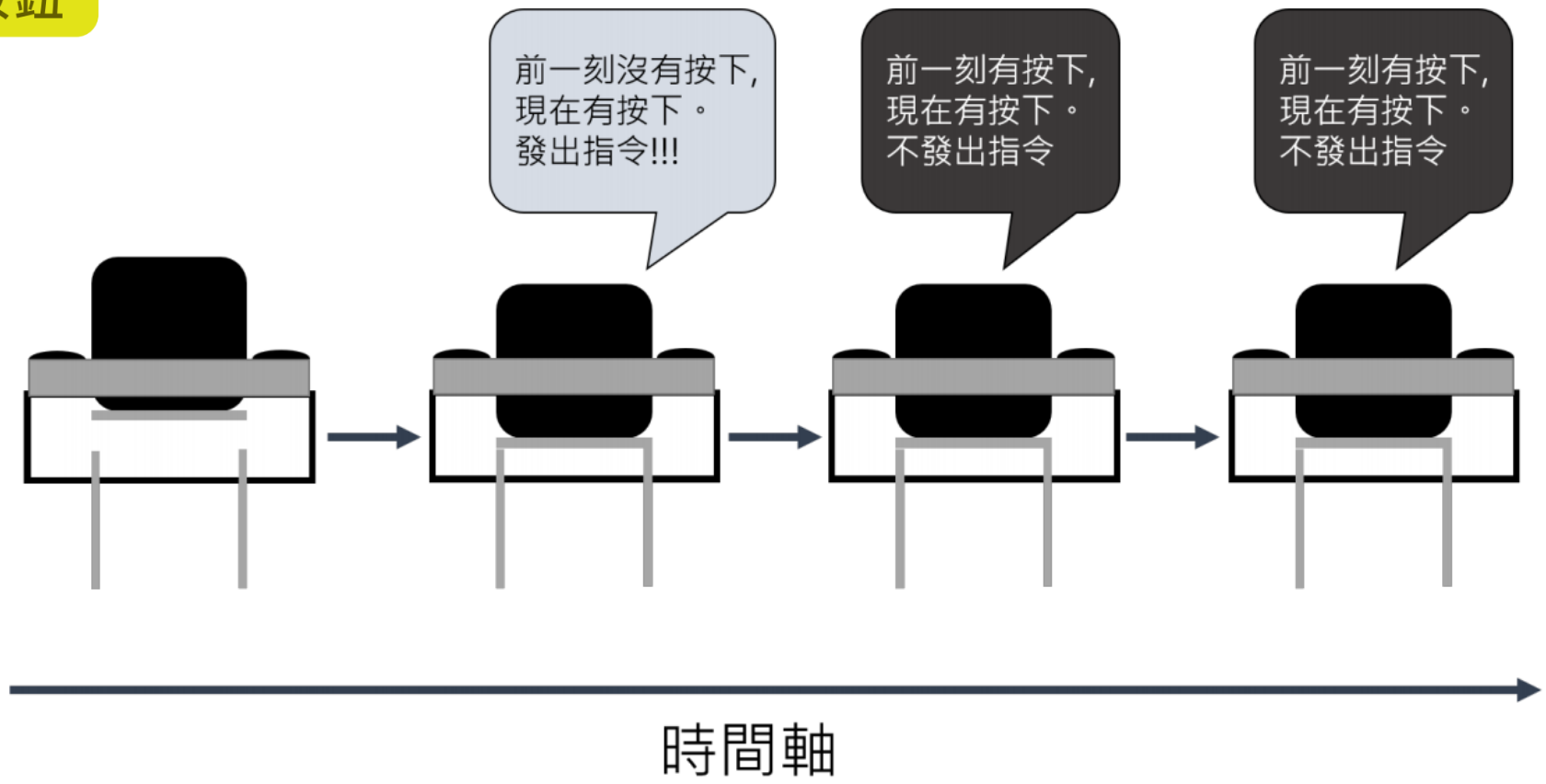
```
>>>  keyboard.screenShot()
```

等同按下 Print Screen 鍵

NEXT

LAB17 截圖神器

截圖按鈕



NEXT

LAB17 截圖神器

程式設計

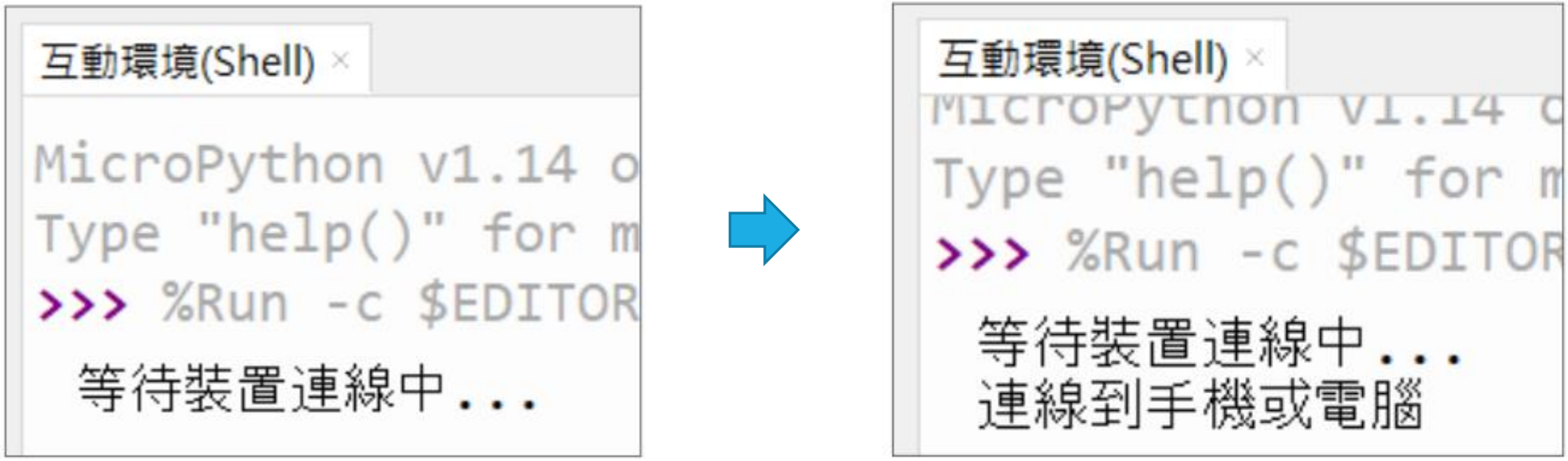
```
04
05  keyboard = BLE_HID("ESP32 keyboard")
14      staUp = button_up.value()
15      # 前一次沒按 且 這次有按
16      if(last_staUp == 1 and staUp == 0):
17          keyboard.screenShot()
18          print("截圖")
19      # 紀錄前一次狀態
20      last_staUp = staUp
21      time.sleep(0.05)
```

開啟範例程式 LAB17
並更改第 5 行的藍牙名稱

NEXT

LAB17 截圖神器

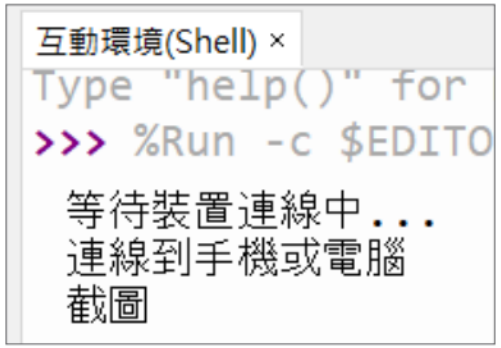
實測



NEXT

LAB17 截圖神器

按下上面按鈕



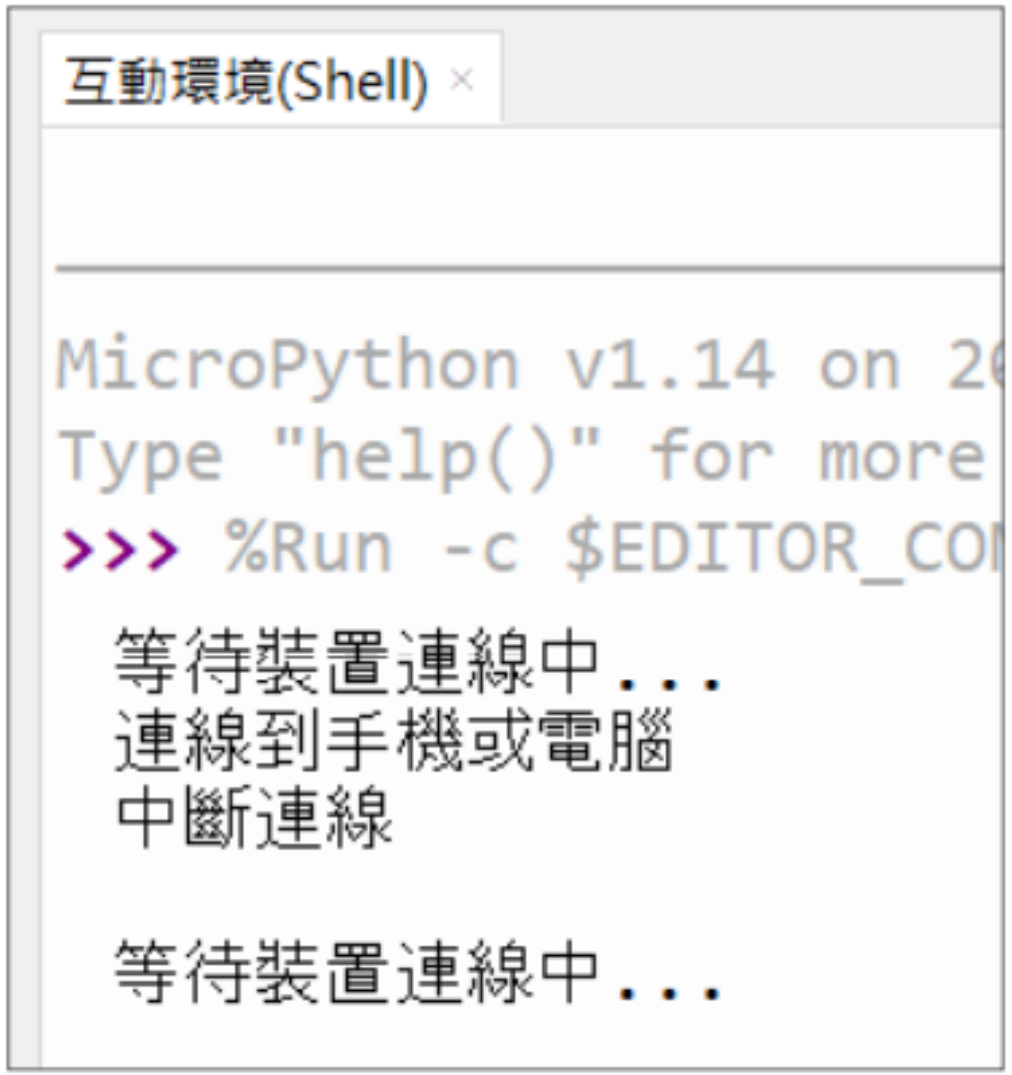
提醒已截圖成功

NEXT

LAB17 截

取消配對

按下**清除**



9-4 多媒體鍵控制

- 控制音量大小
- 上下首
- 撥放/暫停

匯入模組並建立物件：

自行更改名稱, 注意不要超過 18 個字元

```
>>>   from ble_hid import BLE_HID
>>>   mult = BLE_HID("ESP32_Multimedia")
```

LAB18 音量控制器

實驗目的	將 ESP32 變成多媒體控制器，控制手機音量大小。
材料	同 LAB16

線路圖

同 LAB16

NEXT

LAB18 音量控制器

設計原理 音量控制

音量增強：

```
mult.volumeIncrement()
```

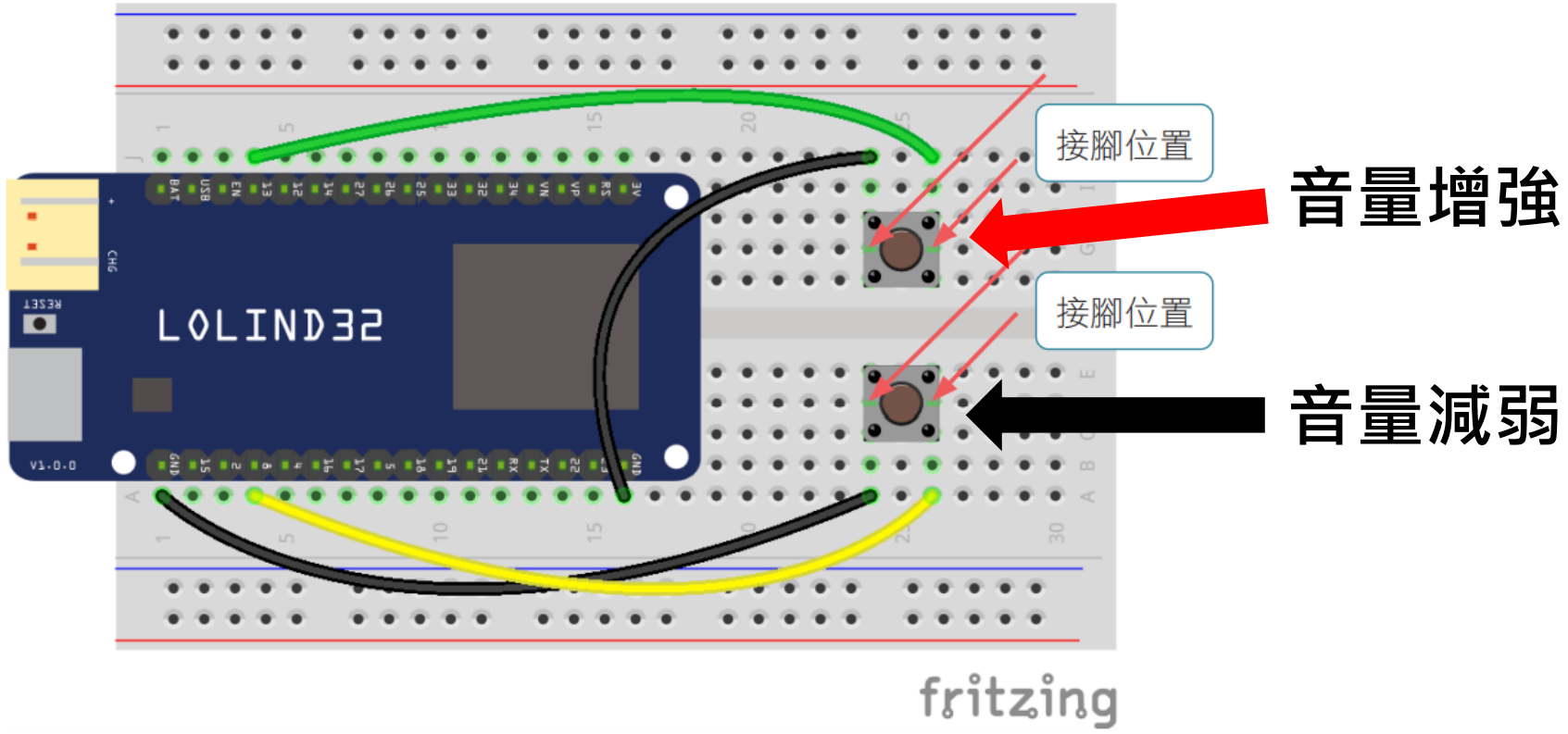
音量減弱：

```
mult.volumeDecrement()
```

NEXT

LAB18 音量控制器

調整音量按鈕



NEXT

LAB18 音量控制器

實測

程式設計

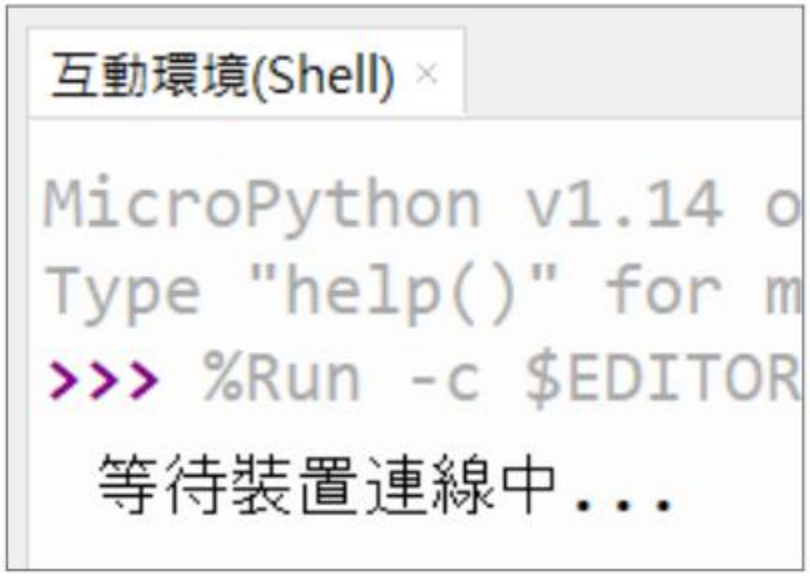
開啟範例程式 LAB18
並更改第 5 行的藍牙名稱

```
01  from machine import Pin
07  # 上面按鈕
08  button_up=Pin(13, Pin.IN, Pin.PULL_UP)
09  # 下面按鈕
10  button_down=Pin(0, Pin.IN, Pin.PULL_UP)
11
```

NEXT

LAB18 音量控制器

實測

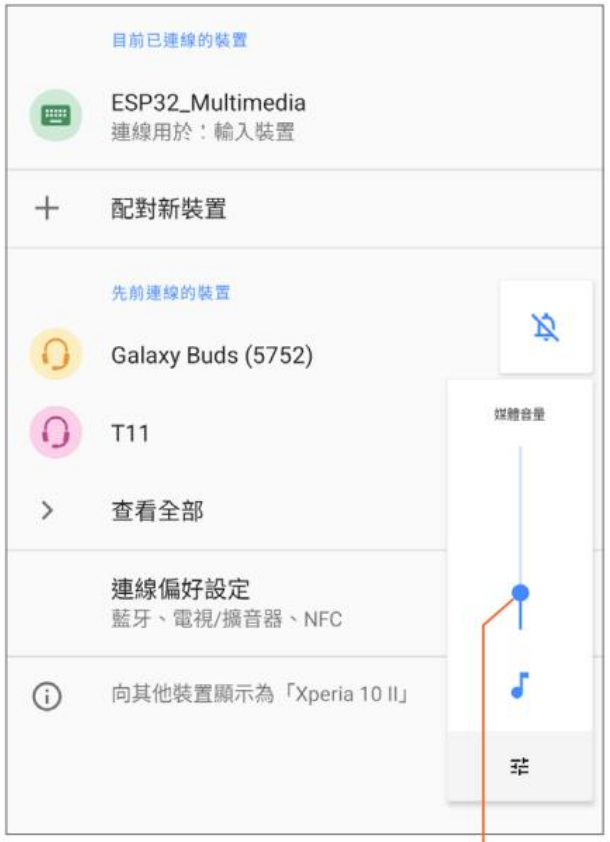


尋找自己改的名稱

NEXT

LAB18 音量控制器

音量增強



音量減弱