Genio Py.®

MicroPython SDK User Manual
MicroPython SDK 用户手册

V2.1 - FEB. 2, 2021

宣告

Mediawave 保留更改此文檔的權利，恕不另行通知，本司提供的文件內容被認為是準確且可靠的，但對本文檔中可能出現的任何錯誤不做任何保證。

下訂單之前，請與 Mediawave 聯繫以獲取最新版本的設備規格，並且 Mediawave 對使用此軟/硬體可能導致的任何專利或第三方其他權利的侵權不承擔任何責任。

此外，Mediawave 產品未經授權可在生命支持設備/系統或航空設備/系統中用作關鍵組件，在這種情況下，如果沒有明確的書面許可，可能會造成產品的故障且會對用戶產生傷害。

名威智慧通訊有限公司

Mediawave Intelligent Communication LTD.

目錄

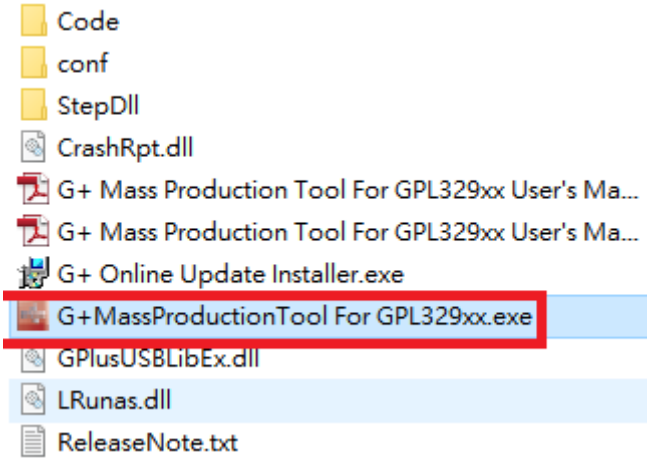
修訂歷史

版本	日期	建檔人員	備註
2.1	2021/02/01	Lynn Lin	更新文件內容
2.0	2020/11/10	Herman Yeh	將PCB更改為V2.0（GPC-2837）
1.0	2020/06/02	Herman Yeh	初版

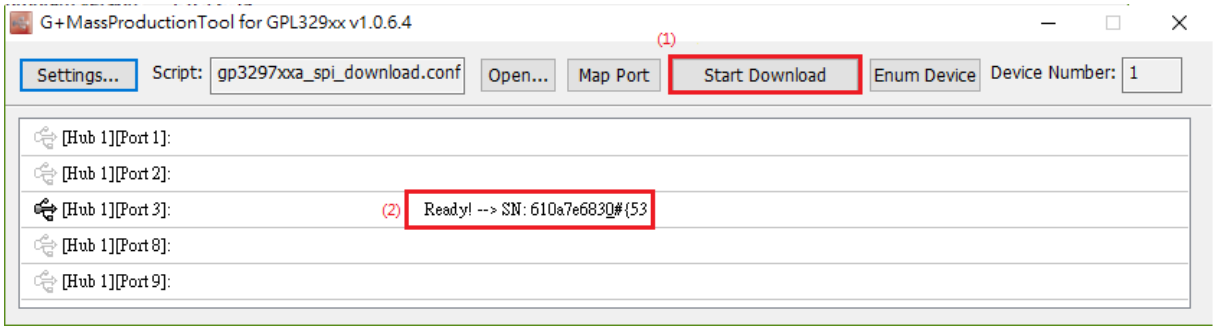
1. 開始使用 Genio py. Stem board

1.1 Firmware 更新

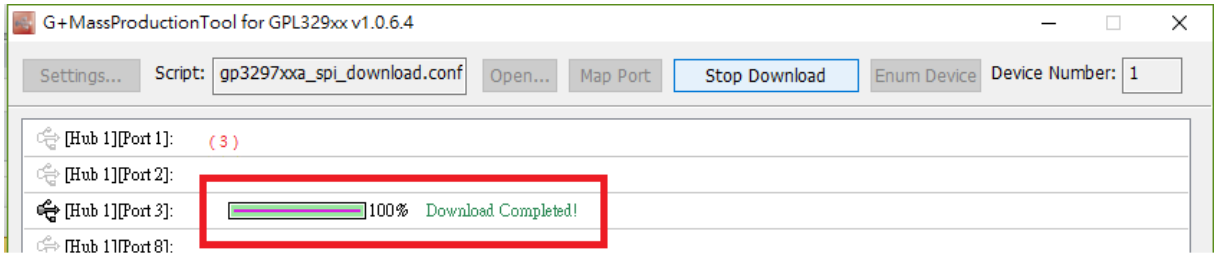
以 G+MassProductionTool 更新



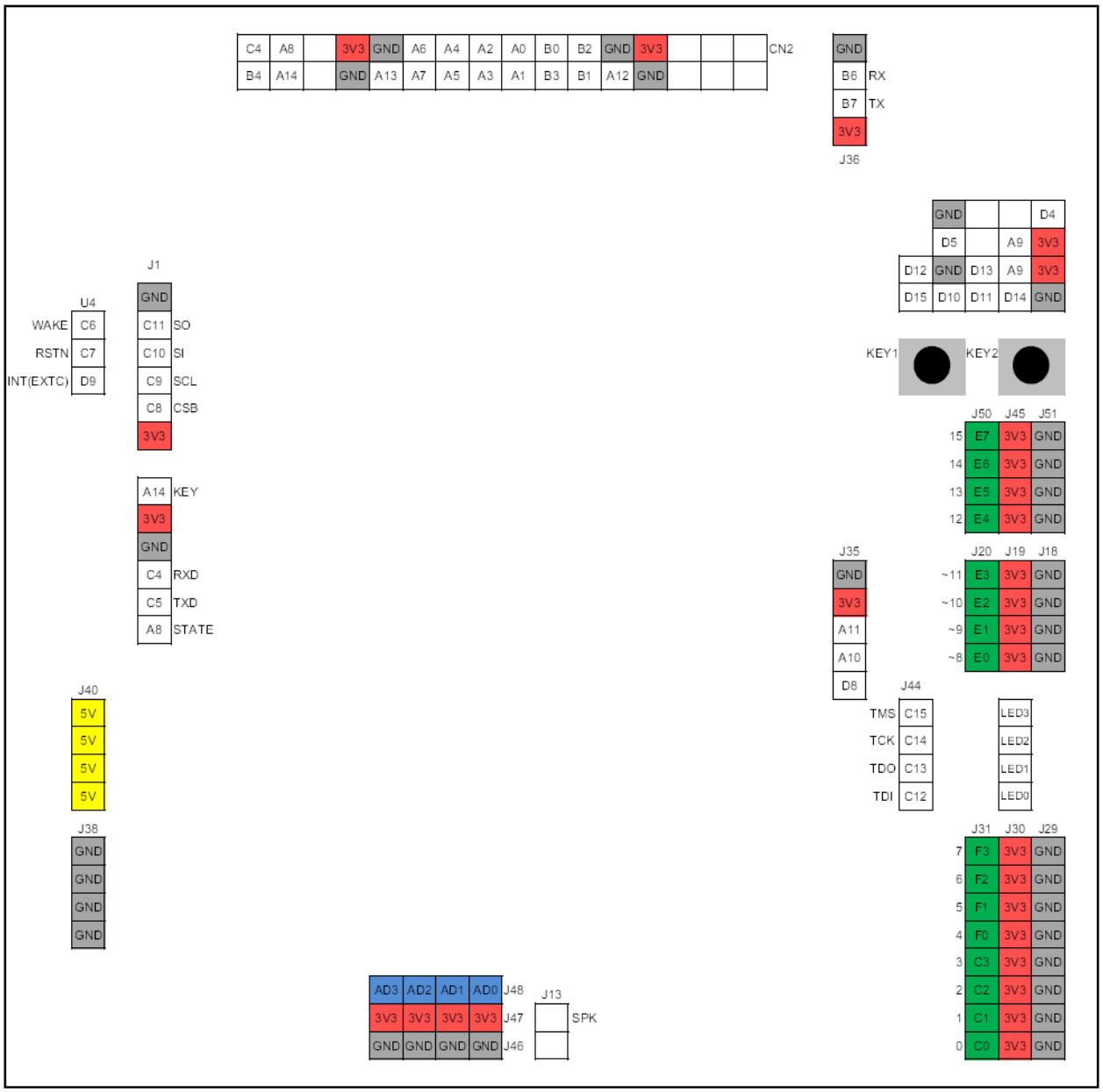
1. 開啟 G+MassProductionTool 後，會顯示認到板子 USB port.
2. 按下 Start Download.



3. Download 完成



1.2 V2 STEM Board Header_GPC-2837



1.3 啟動

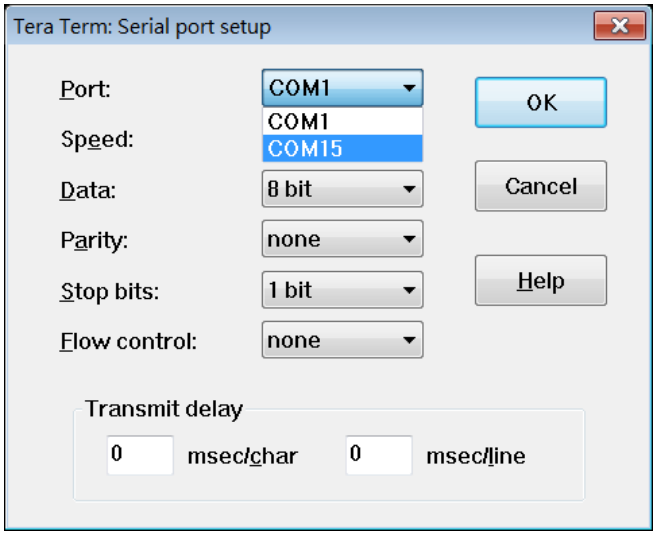
透過 Micro USB port 接上 PC 後, Genio Py. MicroPython Board(以下簡稱為 GPB)即上電, 此時進入 REPL 命令列, 使用者可打開 UART 軟體以輸入 REPL 命令或是使用 G+ Python Studio IDE 進行 python 語言的編寫或下載執行。

註： REPL 意為讀取-求值-列印-迴圈（Read Evaluate Print Loop），是互動式提示的名稱，您可在 GPB 上訪問此互動式終端。 目前測試代碼和運行指令的最簡便方法即使用 REPL。目前，使用 REPL 是檢驗代碼和運行指令的最簡單方式。

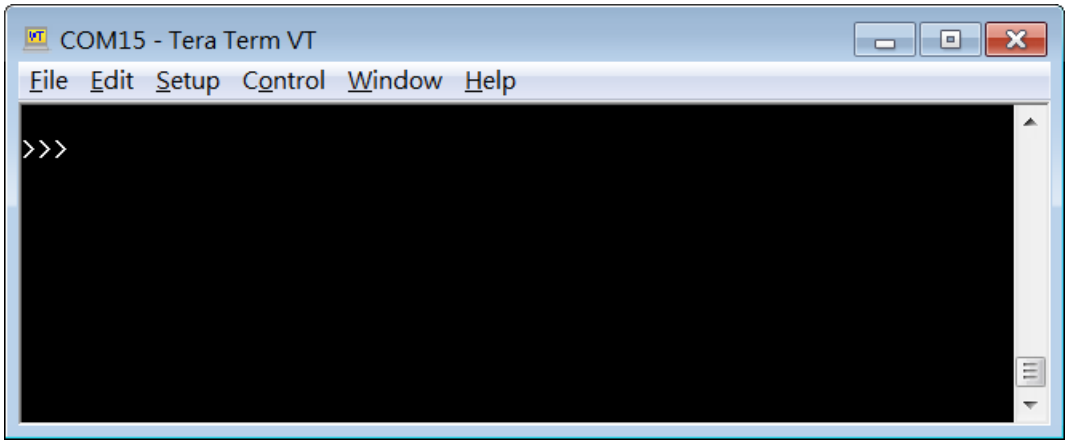
1.4 設置 Terminal 軟體

1. 將 GPB 接上 Windows PC, 待 Windows 安裝完成。
   如下圖例:COM PORT 為 COM15。

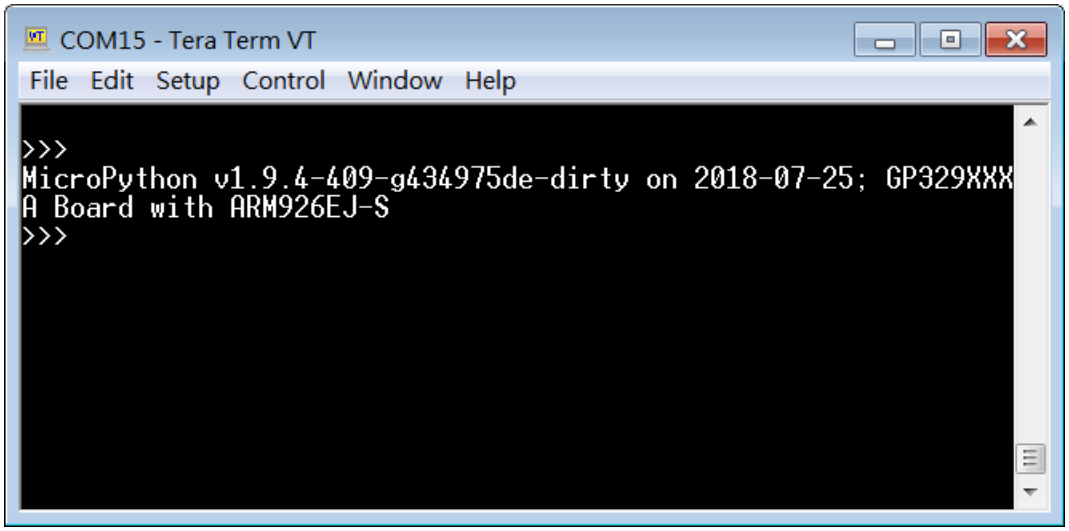
2. 於 Terminal 軟體選擇此 COM PORT(本例使用 “Tera Term” - COM15)



3. 此時在 UART 軟體上可直接輸入 REPL 命令，或按下 Enter，可以看見新的 REPL 提示符，如下圖:



4. 此時若按下 CTRL+B，可以顯示當前 FW 版本, HW 資訊及 REPL 提示等參數，如下圖:



5. 若 Windows 無法找到此 USB 裝置，請安裝驅動程式:

gp_usb_cdc.inf

1.5 使用 REPL

1. 嘗試直接在 gpb 上運行一些 MicroPython 命令，於 Terminal 軟體下，會看到 >>> 的提示符號，在提示符號後面輸入命令，按下 Enter 後會得到相對應的結果,如下圖:

```
>>> print("hello gpboard")
hello gpboard
>>>
```

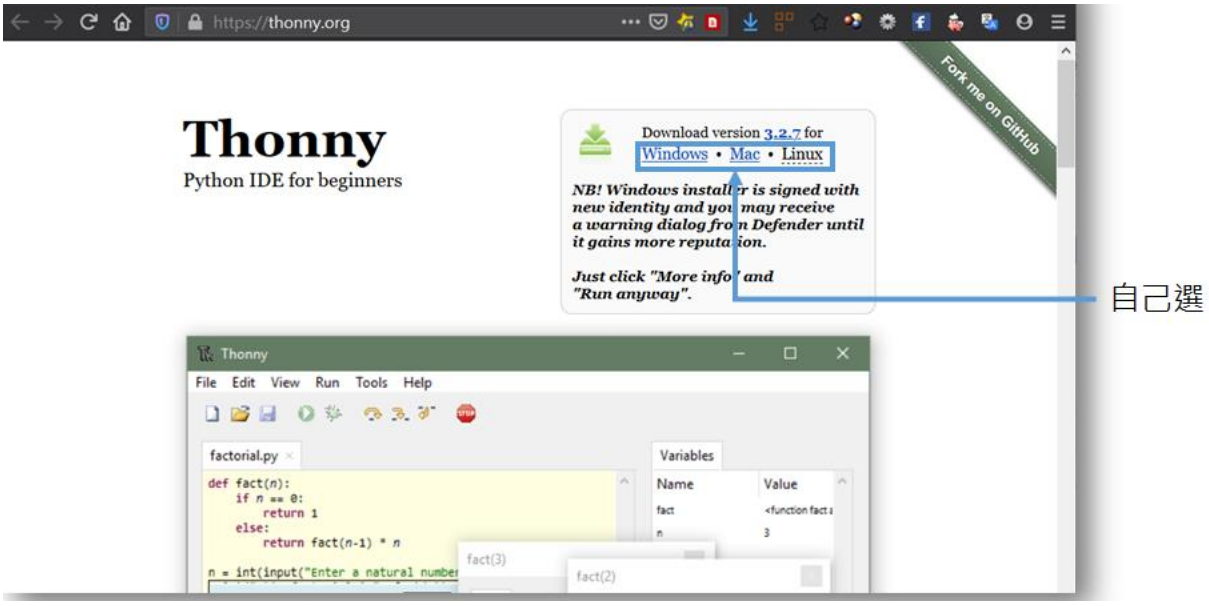
註:在上述過程中，您不應輸入 >>> 字元。此字元表示您應在其後的提示符中輸入文本，您輸入文本 print(“hello gpboard”)並按一下 Enter 鍵後，螢幕上的輸出應與上面顯示的相似。

2. 若您已對 python 有一定瞭解，現在即可嘗試一些基本指令。

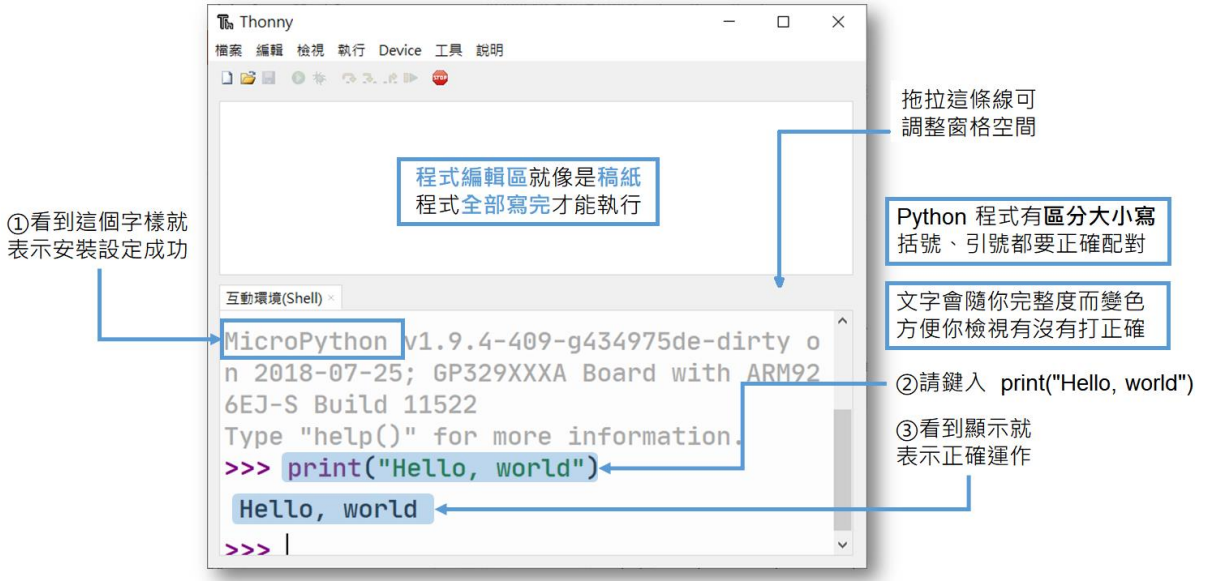
```
>>> 1+1
2
>>> 1/3
0.3333333
>>> 1/4
0.25
>>> 7*8
56
```

1.6 使用 Thonny 開發環境

1. 到 https://thonny.org 下載



2. 安裝完後，設定 Thonny.



1.7 開機自啟動腳本

當系統開機或重置時, gpb 會自動執行 SD 卡中的 boot.py，然後再執行 main.py，編輯這兩個腳本內容即可實現開機自啟。

2. 標準函式庫

2.1 Python 內建函數

由於 MicroPython 語法是基於 Python3, 因此很多模組與模組中的函數都是和 Python3 對應的, 用法也一樣。以下為 MicroPython 的全部內置函數列表:

abs()
all()
any()
bin()
class bool
class bytearray
class bytes
callable()
chr()
classmethod()
compile()
class complex
delattr(obj, name)
class dict
dir()
divmod()
enumerate()
eval()
exec()
filter()
class float
class frozenset
getattr()
globals()
hasattr()
hash()
hex()
id()
input()
class int

classmethod from_bytes(bytes, byteorder)
classmethod to_bytes(size, byteorder)
isinstance()
issubclass()
iter()
len()
class list
locals()
map()
max()
class memoryview
min()
next()
class object
oct()
open()
ord()
pow()
print()
property()
range()
repr()
reversed()
round()
class set
setattr()
class slice
sorted()
staticmethod()
class str
sum()
super()
class tuple
type()
zip()

2.1.18 utime: 時間相關的函數

- utime.localtime(*[secs]*)

  將一個以秒計的時間轉換為一個包含下列內容的 8 元組：(年、月、日、小時、分鐘、秒、一周中某日、一年中某日)。若未提供秒或為 None，則使用 RTC 時間。

  年 ： 包括世紀（例如 2014）

  月 ： 1-12

  日 ： 1-31

  時 ： 0-23

  分 ： 0-59

  秒 ： 0-59

  週中某日 : 0-6 for Mon-Sun.

  年中某日 : 1-366

  例 ：

```
>>> import utime
>>> utime.localtime()
(2000, 1, 1, 0, 0, 58, 5, 1)
>>> utime.localtime(0x3FFFFFFF)
(2034, 1, 9, 13, 37, 3, 0, 9)
>>> utime.localtime(0x40000000)
OverflowError: long int not supported in this build
>>>
```

- utime.mktime()

  此為本地時間的逆函數，其參數為一個表示本地時間的 8 元組。返回一個表示 2000 年 1 月 1 日以來的秒鐘的整數。

- utime.sleep(*seconds*)

  休眠給定秒數的時間。秒數可為一個表示休眠時間的浮點數。注意：其他板子可能不接受浮點參數，為滿足相容性，可使用 sleep_ms() 和 sleep_us() 函數。

- utime.sleep_ms(*ms*)

  延遲給定毫秒數，應為正值或 0。

- utime.sleep_us()

  延遲給定的微秒數，應為正值或 0。

- utime.ticks_ms()

  返回當前以 ms 為單位的 tick 數。

- utime.ticks_us()

  返回當前以 us 為單位的 tick 數。

- utime.time()

  返回整數形式的秒鐘數.

- utime.ticks_add(*ticks, delta*)

  將給定的 tick 數加上一偏移量，此偏移量可為正值或負值. 給定一個 ticks 值，則此函數會算出往前或往後偏移的 ticks 值. Ticks 參數幣需為直接調用 ticks_ms() 和 tick_us() 函數(或先前調用的 ticks_add()). 而 delta 可為任意整數. ticks_add() 對於計算事件/任務的截止時間非常有用。

- utime.ticks_diff(ticks1, ticks2)

  計算兩個經由調用 ticks_ms() 、ticks_us() 所得到的 ticks 參數的差值，此值為有號數且會折返。 由於 ticks 參數有折返的問題，因此若值接將 ticks 值相減可能會導致錯誤的結果.

  ticks_diff() 設計適用於各種使用模式，其中包括:

  i. 使用超時輪詢。在此種情況下，事件順序已知，因此只需處理 ticks_diff()的正結果:

```
# Wait for GPIO pin to be asserted, but at most 500us
start = time.ticks_us()
while pin.value() == 0:
    if time.ticks_diff(time.ticks_us(), start) > 500:
        raise TimeoutError
```

  ii. 安排事件。在此種情況下，若某一事件超期，則 ticks_diff() 的结果可能為負:

```
# This code snippet is not optimized  這行程式碼尚未進行優化
now = time.ticks_ms()
scheduled_time = task.scheduled_time()
if ticks_diff(now, scheduled_time) > 0:
    print("Too early, let's nap")
    sleep_ms(ticks_diff(now, scheduled_time))
    task.run()
elif ticks_diff(now, scheduled_time) == 0:
    print("Right at time!")
    task.run()
elif ticks_diff(now, scheduled_time) < 0:
```

```
    print("Oops, running late, tell task to run faster!")

    task.run(run_faster=true)
```

注意：請勿將 time()值傳遞給 ticks_diff()，您應在此使用正常的數學運算。但是請注意 time()可能（且將會）溢出。

2.1.19 uzlib: zlib 解壓縮

2.2 MicroPython 特別函式庫

2.2.1 Function

2.2.1.1 machine – 硬件相關函數

1. **電源相關函數**
   - machine.freq() : 返回 CPU 頻率
   - machine.idle() : Enter WFI.

2. **中斷相關函數**
   - machine.disable_irq() : 關閉中斷
   - machine.enable_irq() : 開啟中斷

2.2.2 Classes

2.2.2.1 class Pin control

目前 gpb 可提供的 Pin Name 如下, Pin 的 CPU Name 與 Board Name 相同.

A0 ~ A15,
B0 ~ B15,
C0 ~ C15,
D0 ~ D15,
E0 ~ E7,
F0 ~ F9..等等。

註: 請注意由於一些周邊占用固定 I/O，可以使用的 I/O 請參考 board header。

1. 構造函數

- Class machine.Pin(*id, mode*)
  - i. id: 目前 gpb 所提供的 Pin name.
  - ii. mode :
    - ✓ Pin.IN: input mode.
    - ✓ Pin.OUT: output mode.
    - ✓ Pin.FLOAT: floating mode.
    - ✓ Pin.OUT_INV: output inverse mode.

範例:

```
>>> import machine
>>> from machine import Pin
>>> pin0 = Pin('A0', Pin.IN)
>>> pin0
Pin(Pin.cpu.A0, mode=Pin.IN)
>>> pin1 = Pin('A1', Pin.OUT)
>>> pin1
Pin(Pin.cpu.A1, mode=Pin.OUT)
>>>
```

2. Methods

- Pin.init(*mode = pin_mode, value = pin_value, drive = pin_drive*)
  給定參數初始化 Pin 腳, 只設定指定的參數, 沒有設定的參數將不發生變化，其中:
  - i. pin_mode :
    - ✓ 0x00 : input
    - ✓ 0x01 : output
    - ✓ 0x02 : float
    - ✓ 0x03 : output inv
  - ii. pin_value :
    - ✓ 0 : Low
    - ✓ 1 : High
  - iii. pin_drive :
    - ✓ 0 : 8m
    - ✓ 1 : 16mA

範例 :

```
>>> import machine
>>> from machine import Pin
>>> a = Pin('B0')
>>> a
Pin(Pin.cpu.B0, mode=Pin.IN)
>>> a.init(mode = 1, value = 1, drive = 0)
>>> a
Pin(Pin.cpu.B0, mode=Pin.OUT)
>>>
```

- Pin.value(x)

  設置或讀取 Pin 的邏輯電位, 若不帶參數則為讀取 Pin 的邏輯電位. 1 為 High, 0 為 Low.

範例 :

```
>>> import machine
>>> from machine import Pin
>>> b = Pin('A1')
>>> b
Pin(Pin.cpu.A1, mode=Pin.IN)
>>> b.init(mode = 1, value = 0, drive = 0)
>>> b
Pin(Pin.cpu.A1, mode=Pin.OUT)
>>> b.value()
0
>>> b.value(1)
>>> b.value()
1
>>>
```

- Pin.off()

  設置 Pin 為邏輯 0。

- Pin.on()

  設置 Pin 為邏輯 1。

- Pin.low()

  設置 Pin 為邏輯 0。

- Pin.high()

  設置 Pin 為邏輯 1。

- Pin.name()

  顯示 Pin 名稱。

- Pin.names()

  顯示 cpu name 及 board name.

範例 :

```
>>> import machine
>>> from machine import Pin
>>> a = Pin('A0')
>>> a.name()
'A0'
>>> a.names()
['A0', 'A0']
>>>
```

- Pin.mode() : 顯示 Pin 模式
  i. Return:
     - ✓ IN → 0
     - ✓ OUT → 1
     - ✓ FLOAT → 2
     - ✓ OUT_INV → 3

範例 :

```
>>> import machine
>>> from machine import Pin
>>> a = Pin('A0', Pin.IN)
>>> a.mode()
0
>>> a = Pin('A0', Pin.OUT)
>>> a.mode()
1
>>> a = Pin('A0', Pin.FLOAT)
>>> a.mode()
2
>>> a = Pin('A0', Pin.OUT_INV)
>>> a.mode()
3
>>>
```

- Pin.irq(handler=none, trigger = (*Pin.IRQ_RISING*|*Pin.IRQ_FALLING*)
  設置一個中斷處理函數，當觸發源處於活動狀態時調用. 若腳位模式為 Pin.IN，則觸發源為外部值. 目前僅支持 IOD8(EXTB), IOD9(EXTC), IOF0(EXTF), IOF1(EXTE), IOF2(EXTD)可使用外部中斷，若使用在這 5 組外部中斷以外的腳位，則回傳錯誤。
  i. handler : 回調函數，當中斷觸發時被調用 。
  ii. trigger : 當腳位處於此狀態時則觸發中斷:
     - ✓ Pin.IRQ_RISING : 正緣觸發
     - ✓ Pin.IRQ_FALLING: 負緣處發

範例:

```
>>> import gpb
>>> from gpb import Pin
>>> a=Pin('B5', Pin.IN)
>>> a.irq(lambda p:gpb.LED(1).toggle())
```

2.2.2.2 class I2C 雙線串行協議

1. 構造函數

- class machine.I2C(*id, SCL, SDA, freq=400000*)
  - i. id :
    - ✓ 若 id=-1，代表使用軟體 I2C 模式，需要指定 SCL 及 SDA 參數，可以使用任意引腳。
    - ✓ 若 id >0，則代表硬體 I2C.
    - ✓ Id = 1，表示 I2C1 , id = 2 , I2C2、、、、
  - ii. SCL：軟體模式下 SCL 使用引腳。
  - iii. SDA: 軟體模式下 SDA 使用引腳。
  - iv. Freq：設置 SCL 上最大頻率值。

2. General Methods

- I2C.scan()

  掃描 I2C 總線上地址在 0x08~0x77 之間的設備，返回響應的地址列表。

```
>>> import machine
>>> from machine import I2C
>>> i2c=I2C(1,freq=400000)
>>> i2c.scan()
[80]
```

- init(*scl, sda, freq=400000*)

  初始化 I2C 總線

  - i. scl :軟體 I2C 模式下的 SCL 腳位，硬體 I2C 模式下這個參數無效。
  - ii. sda :軟體 I2C 模式下的 SDA 腳位，硬體 I2C 模式下這個參數無效。
  - iii. freq : SCL 上最大時脈。

3. 基本 I2C 操作

- I2C.start()

  於 BUS 上產生一 START 信號. 此函數於軟體 I2C 才可使用。

- I2C.stop()

  於 BUS 上產生一 STOP 信號. 此函數於軟體 I2C 才可使用。

- I2C.readinto(*buf, nack=True*)

  經由 bus 讀取 1 byte 資料到緩衝 buf. 此函數於軟體 I2C 才可使用。

範例 : 讀取總線位址 80 的 EEPROM 記憶體位址 0 的資料

```
>>> data=bytearray(2)
>>> data1=bytearray(1)
>>> data2=bytearray(1)
>>> data[0]=160
>>> data[1]=0
>>> data1[0]=161
>>> data2[0]=0
>>> i2c.start()
>>> i2c.write(data)    //////data[0]:device address &
data[1]:word address
2
>>> i2c.start()
>>> i2c.write(data1)    ////data1: device address
1
>>> i2c.readinto(data2)    ////data2: buffer address
>>> i2c.stop()
>>> data2
bytearray(b'\xa5')
```

- I2C.write(*buf*)

  將緩衝 buf 的資料經由 bus 寫入 1 byte. 此函數於軟體 I2C 才可使用。

範例 : 對總線位址 80 的 EEPROM 寫入數值為 8 的 1byte 資料到記憶體位址 8

```
>>> addr = bytearray(1)
>>> addr[0]=160  //EEPROM 的總線位址 80, 所以 addr 為
b'10100000'
>>> memaddr = bytearray(1)
>>> memaddr[0] = 8
>>> i2c.start()
>>> i2c.write(addr)       ###寫入 device addr
1
>>> i2c.write(memaddr)   ###寫入 word addr
1
>>> i2c.write(memaddr)   ###寫入 data
1
>>> i2c.stop()
>>> i2c.readfrom_mem(80, 8, 1)
b'\x08'
```

4. 標準總線操作

- I2C.readfrom(*addr, nbytes, stop=True*)

  從位址 addr 讀取 nbytes 數據, 如果 stop 是 true, 則在最後法送一個 STOP 信號. 返回值為 nbytes 數據。

範例 : 寫入 0x81 到 EEPROM 的記憶體位置 0, 再用 Random read 讀出

```
>>> i2c.writeto_mem(80,0,b'\x81')   //////寫 0x81 到 word
address 0
>>> i2c.readfrom_mem(80,0,1)
b'\x81'
>>> i2c.start()
>>> i2c.write(data)
2
>>> i2c.start()
>>> i2c.readfrom(80,1)
b'\x81'
```

- I2C.readfrom_into(*addr, buf stop=True*)

  從位址 addr 讀取數據到 buf, 讀取的數據量為 buf 長度, 如果 stop 是 true, 則在最後發送一個 STOP 信號. 此函數無返回值。

範例:先寫入 0x81 到 EEPROM 的記憶體位置 0, 再用 Random read 讀到緩衝 data3

```
>>> data3=bytearray(1)
>>> data3
bytearray(b'\x00')
>>> i2c.writeto_mem(80,0,b'\x81')        //////寫 0x81 到 word
address 0
>>> i2c.readfrom_mem(80,0,1)
b'\x81'
>>> i2c.start()
>>> i2c.write(data)
2
>>> i2c.start()
>>> i2c.readfrom_into(80,data3)
>>> data3
b'\x81'
```

- I2C.writeto(*addr, buf, stop=True*)

  寫入 buf 的數據到地址是 addr 的設備. 如果在寫操作後收到 NACK 信號, 剩餘的數據將不會被發送, 如果 stop 為 true, 則在最後發送一個 STOP 信號, 即使收到 NACK 也會發送 STOP 信號. 返回值是收到 ACK 的信號數。

範例 : 對 EEPROM 位址 80 的記憶體位址 2 寫入數值為 8 的資料

```
>>> data=bytearray(2)
>>> data[0]=2   //word address
>>> data[1]=8   //data
>>> i2c.writeto(80, data)
>>> i2c.readfrom_mem(80,2,1)
b'\x08'
```

5. 內存操作

- I2C.readfrom_mem(*addr, memaddr, nbytes, addrsize=8*)
  從指定設備位址 addr 的 memaddr 處開始讀取 nbytes 資料. addrsize 參數指定位址寬度，返回值是讀到的資料。
- readfrom_mem_into(*addr, memaddr, buf, addrsize=8*)
  從指定設備位址 addr 的 memaddr 處開始讀取 nbytes 資料到緩衝 buf. addrsize 參數指定位址寬度，返回值是讀到的資料。
- I2C.writeto_mem(*addr, memaddr, buf, addrsize=8*)
  寫入緩衝 buf 的資料到設備位址 addr 的內存地址 memaddr. addrsize 參數指定位址寬度。

```
>>> import machine
>>> from machine import I2C
>>> i2c=I2C(-1,'B4','B5',freq=400000)
>>> data=bytearray(4)
>>> data[0]=0
>>> data[1]=1
>>> data[2]=2
>>> data[3]=3
>>> i2c.scan()
[80]
>>> i2c.writeto_mem(80,0,data)
>>> i2c.readfrom_mem(80,0,4)
b'\x00\x01\x02\x03
>>> c=bytearray(4)
>>> i2c.readfrom_mem_into(80,0,c)
>>> c
b'\x00\x01\x02\x03
```

2.2.2.3 class WDT Watchdog timer

WDT 用於在應用程式崩潰且最終進入不可恢復狀態時重啟系統。一旦開始，就不可停止或重新配置。啟用後，應用需定期"餵養"看門狗，以防止其終止並重置系統。

1. 構造函數

- class machine.WDT(*timeout = to*)
  - i. to :
    - ✓ 125 : 125ms
    - ✓ 250 : 250ms
    - ✓ 500 : 500ms
    - ✓ 1000 : 1 second
    - ✓ 2000 : 2 second
    - ✓ 4000 : 4 second
    - ✓ 8000 : 8 second
    - ✓ 16000: 16 second
    - ✓ 其他數值將會返回錯誤.

2. Methods

- WDT.feed()

  需於設定 watchdog 的 timeout 時間內餵狗，否則會 reset.

範例 :

```
>>> import machine
>>> from machine import WDT
>>> a = WDT(timeout = 2000)
>>> a.feed()
```

2.2.2.4 class ADC Analog to Digital Converter

1. 構造函數

- class machine.ADC(Pin)
  - i. Pin : 選用的 ADC 通道 0,1,2,3 或輸入 Pin 名稱
    - ✓ 0 對應 IOE8
    - ✓ 1 對應 IOE9
    - ✓ 2 對應 IOE10
    - ✓ 3 對應 IOE11

2. Methods

- ADC.read()

  讀取目前 ADC 通道數值。

  範例 :

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM926
6EJ-S
>>> import machine
>>> from machine import ADC
>>> ad0 = machine.ADC(0)
>>> ad0.read()
699
>>> ad0.read()
2727
>>> ad3 = machine.ADC('E11')
>>> ad3.read()
1681
>>> ad3.read()
73
>>>
```

- ADC.key_init()

  初始化 AD 按鍵設置, 做任何按鍵操作前都必須先執行 key_init().

- ADC.key_uninit()

  反初始化 AD 按鍵設置。

- ADC.key_callback(*key_num, fun*)

  設置 Key 的 callback function

  i. key_num : 按鍵號

  ii. fun : callback function

- ADC.key_scan()

  返回目前按下的按鍵值。

  返回 0 則目前沒有按鍵被按下,

  返回 1 則目前按鍵 1 被按下,

  返回 2 則目前按鍵 2 被按下,

  、

  、

  、以此類推

範例 : 對按鍵 1~7 設置 callback 函數

```
>>> import machine
>>> from machine import ADC
>>> adkey = ADC(0)
>>> adkey.key_init()
>>> adkey.key_callback(1, lambda x: print("1"))     #print
"1" as key 1 is pressed
>>> adkey.key_callback(2, lambda x: print("2"))     #print
"2" as key 2 is pressed
>>> adkey.key_callback(3, lambda x: print("3"))     #print
"3" as key 3 is pressed
>>> adkey.key_callback(4, lambda x: print("4"))     #print
"4" as key 4 is pressed
>>> adkey.key_callback(5, lambda x: print("5"))     #print
"5" as key 5 is pressed
>>> adkey.key_callback(6, lambda x: print("6"))     #print
"6" as key 6 is pressed
>>> adkey.key_callback(7, lambda x: print("7"))    #print
"7" as key 7 is pressed
```

2.2.2.5 class PWM Pulse Width Modulation

1. 構造函數

- class machine.PWM(*CH, freq, duty*)
  - i. CH : 選用的 PWM 通道, 0~3 =>對應 IOE0, IOE1, IOE2, IOE3
  - ii. Freq : PWM 的頻率
  - iii. Duty : PWM 的工作週期
- PWM.duty(N)

  N 若無參數, 返回目前工作週期, N = 0~100, 設定 PWM 的工作週期。
- PWM.freq(N)

  N 若無參數, 返回目前頻率, N = 2~10000, 設定 PWM 的頻率。
- PWM.deinit()

  關閉 PWM.

範例 :

```
>>> pwm1 = PWM(0,duty=50,freq=2500)
>>> pwm1
PWM(0, freq=2500, duty=50)
>>> pwm2 = PWM(1,5000,56)
>>> pwm2
PWM(1, freq=5000, duty=56)
>>> pwm1.duty()
50
>>> pwm1.duty(67)
>>> pwm1.duty()
67
>>>
```

2.3 GPB 特別函式庫

2.3.1 Function

2.3.1.1 gpb (genio py board) 相關函數

1. 時間相關函數

- gpb.delay(ms)
  延時指定的時間, 單位為 ms.
- gpb.millis()
  返回 reset 後運行的時間(ms)
- gpb.micros()
  返回 reset 後運行的時間(us)
- gpb.elapsed_millis(start)
  返回從時間 start 到現在時刻的毫秒數。

範例 :

```
>>> import gpb
>>> gpb.delay(100)
>>> gpb.millis()
130680
>>> gpb.elapsed_millis(100000)
46219
>>> gpb.micros()
152570167
>>>
```

2. 中斷相關函數

- gpb.disable_irq()
  關閉中斷, 於 REPL 中請勿使用此功能, 會讓 Com Port 停止運作。
- gpb.enable_irq(*state=True*)
  如果 state 式 True(默認值), IRQs 將允許, 否則將禁止 IRQs 請求。

3. 功耗管理相關函數

- gpb.wfi()
  進入 wfi, 等待內部或外部中斷。

2.3.2 Classes

2.3.2.1 class Pin

同 machine module 的 Pin Class, 請參照 p17.

1. **構造函數**
   - Class gpb.Pin(*id, mode*)

2.3.2.2 Class LED

目前 GP STEM Board 的 LED1~LED4 是使用 IOC12~IOC15 控制腳位。

1. **構造函數**
   - class gpb.LED()
2. **Methods**
   - LED.toogle()
     Toggle LED
   - LED.on()
     開 LED
   - LED.off()
     關 LED

2.3.2.3 Class Motor

馬達 class 是設計給 PWM 與 IO 控制方向的馬達模組所使用。輸出 1 個 200Hz PWM 訊號, 方向控制分為 1Pin 模式和 2Pin 模式, 2Pin 模式為控制 H 電橋驅動電路所設計, 所指定的兩個 IO 互為反相。

1. **構造函數**
   - class gpb.Motor(*Pin_PWM, Pin_Direction1, Speed, Pin_mode, Pin_ Direction2*)
     i. Pin_PWM : 8~11 (對應 IOE0~IOE3).
     ii. Pin_Direction1 : 一般 IO。
     iii. Speed : PWM 工作週期。
     iv. Pin_mode:
     - ✓ 方向 Pin 為 1 Pin 模式或 2 Pin 模式。
     - ✓ 參數 Motor.DIR1P 或 Motor.DIR2P.

     v. Pin_Direction2 : 一般 IO

vi. 返回：

✓ 成功設定返回

```
Motor.E3, Direction(2 Pin)=CW(pin.A0,pin_b.A2),speed=25
```

✓ 失敗返回其他硬體暫用(PWM 與 Timer 共用硬體，若系統已使用，該阜無法使用 PWM)

```
>>> m0 = Motor('E3','E1')
Traceback (most recent call last)
  File "<stdin>", in <module>
ValueError: PWM timer occupied
```

✓ 失敗返回 Pin 不支援 PWM 功能

```
>>> m0 = Motor('A0','E1')
Traceback (most recent call last):
  File "<stdin>", in <module>
ValueError: Pin.A0 doesn't support Motor
```

2. Methods

- Motor.init(*pin, direction, speed*)

i. Pin ：方向 pin

ii. direction：順時鐘轉(CW)或逆時鐘轉(CCW)

iii. speed: 馬達速度,PWM 工作週期(0~100)

```
m0.init(pin='E3',dir=0,speed=99)
```

- Motor.on()

  開始輸出 PWM.

- Motor.off()

  停止輸出 PWM.

- Motor.CW()

  設定方向 IO 輸出 1(順時針)。

- Motor.CCW()

  設定方向 IO 輸出 0(逆時針)。

- Motor.speed(N)

  設定輸出 PWM 的工作週期(速度).

i. N=0~100

```
>>> m0.speed()
30
```

- Motor.name()
  顯示使用的 PWM 名稱

範例 ：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM92
6EJ-S
>>> import gpb
>>> from gpb import Motor
>>> m0 = Motor('E0','E1')
>>> m0
Motor(Motor.E0,mode=CW(pin.E1),speed=25)
>>> m0.speed(30)
>>> m0.speed()
30
>>> m0.CCW()
>>> m0
Motor(Motor.E0,mode=CCW(pin.E1),speed=30)
>>> m0.on()
>>> m0.off()
>>> m0.CW()
>>> m0.on()
>>> m0.off()
>>>
```

2.3.2.4 Class Servo

伺服馬達 class 是設計給伺服馬達模組, 以 PWM 控制位置, 或是轉動。

1. **構造函數**
   - class gpb.Servo(*servo_num*)
     i. servo_num : 1~4(GPC-2837 主板對應 IOE0,IOE1,IOE2,IOE3)
     ii. 返回:

```
>>> s1 = Servo(1)
>>> s1
<Servo 1 at 1500us>
```

2. **Methods**
   - Servo.pulse_width(*value*)
     輸出設定目前脈波位置(單位 usec), 未設定值即回傳目前脈波值。
   - Servo.angle(*angle, time=0*)
     i. Angle : 轉到設定角度脈波位置。
     ii. Time: 轉到指定角度的時間, 單位 msec, 未設定則以馬達最快的速度轉動。
   - Servo.speed(*speed, time=0*)
     i. Speed : 轉動設定速度, -90 度為一方向最高速, +90 度另一方向最高速, 0 即停止。
     ii. Time : 轉到的時間, 單位 msec, 未設定則一直轉動。

- Servo.close(*servo_num*)
  關閉 PWM.
  i. servo_num: 1~4.
- Servo.calibration(*pulse_min, pulse_max, pulse_centre[, pulse_angle_90, pulse_speed_100]*)
  i. pulse_min: 最小脈波寬度，取決於不同伺服馬達。
  ii. pulse_max: 最大脈波寬度，取決於不同伺服馬達。
  iii. pulse_centre: 脈波寬度對應於馬達中心點(0 度點位置)。
  iv. pulse_angle_90: 脈波寬度對應於角度 90 度之比例。

2.3.2.5 pulse_speed_100: 脈波寬度對應於速度 100 之比例。

2.3.2.6 Class SPI

1. 構造函數

- class gpb.SPI(*id, baudrate, mio*)
  - i. id : 0 or 1 , SPI device number
  - ii. baudrate : SPI clock speed
  - iii. mio :
    - ✓ 0 -> 1-bit mode
    - ✓ 1 -> 2-bit mode
    - ✓ 2 -> 4-bit mode
  - iv. 設定返回成功 spi id:

```
>>> spi=SPI(0,baudrate=12000000,mio=0)
IDX: 0 , mio: 0 , CLK: 12000000 , POL_PHA: 2
spi flash id = c2,20,17
```

2. Methods

- spi.spi_erase(*size*)

  清除 SPI 的資料，size 為 erase 的長度。

- spi.spi_write(*buf*)

  將緩衝 buf 的資料寫入 SPI.

- spi.spi_read(*size,buf*)

  讀取長度為 size 的資料到緩衝 buf

範例:

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM926EJ-S
>>> import gpb
>>> from gpb import SPI
>>> spi=SPI(0,baudrate=12000000,mio=0)
IDX: 0 , mio: 0 , CLK: 12000000 , POL_PHA: 2
spi flash id = c2,20,17
>>> spi.spi_erase(2)
SPI ERASE !!
erase len: 2
>>> spi.spi_read(2)
SPI READ !!
read len: 2
b'\xff\xff'
>>> data=bytearray(2)
>>> data[0]=1
>>> data[1]=2
>>> spi.spi_write(data)
SPI WRITE !!
io 0
>>> spi.spi_read(2)
SPI READ !!
read len: 2
b'\x01\x02'
```

2.3.2.7 Class UART

1. **構造函數**
   - class gpb.UART(*id, baudrate*)
     i. id : 0 , 1 or 2 ，UART device number.
     ii. baudrate : UART baudrate.
2. **Methods**
   - uart.uart_write(*buf*)
     寫入緩衝 buf 的資料。
   - uart.uart_read(*size*)
     讀取長度為 size 的資料。

2.3.2.8 Class SD card

1. **構造函數**
   - class gpb.sdcard()
     創建一個 sdcard 對象並初始化。
2. **Methods**
   - sdcard.present()
     偵測 sdcard 是否存在, 存在返回值 1, 不存在返回值 0。
   - sdcard.power(*state*)
     開啟或關閉 sdcard 功能。
     i. state : 0 or 1
   - sdcard. info()
     返回一 tuple 資訊,包含 sdcard 容量(bytes), sector 大小,及 card type.
   - sdcard.read(*lba*)
     返回讀取數據。
     i. lba : 讀取的 logic block address.
   - sdcard.write(*lba, data*)
     i. lba : 寫入的 logic block address.
     ii. Data : 欲寫入的數據。
   - sdcard.readblocks(*lba, buf*)
     返回 0 讀取成功。
     i. lba : 讀取的 logic block address.
     ii. Buf : 讀取數據的 buffer.

- sdcard.writeblocks(*lba, buf*)

  返回 0 寫入成功。

  i. lba : 寫入的 logic block address.

  ii. Buf : 寫入數據的 buffer

- sdcard.ioctl(*cmd, arg*)

  i. cmd : 操作命令

  - ✓ MP_BLOCKDEV_IOCTL_INIT : power on
  - ✓ MP_BLOCKDEV_IOCTL_DEINIT : power off
  - ✓ MP_BLOCKDEV_IOCTL_SYNC : do nothing
  - ✓ MP_BLOCKDEV_IOCTL_BLOCK_COUNT : 返回卡總 sector 數
  - ✓ MP_BLOCKDEV_IOCTL_BLOCK_SIZE : 返回卡 sector 大小

  ii. Arg : 0

2.3.2.9 Class Timer

1. 構造函數

- class gpb.timer(*id…*)

  通過指定參數新建一個 timer 對象。

  i. id : 指定 timer 編號, 數值為 0~7, 分別對應 TimerA~TimerH

  註:其中 TimerD(id=3)與 TimerF(id=5)已預設被使用, 請盡量避免使用這兩個 timer.

2. Methods

- timer.init(*freq, callback*)

  初始化 timer.

  i. Freq : 設置頻率。

  ii. callback: timer 中斷的回調函數。

  註: 由於此回調函數於中斷中調用, 所以請不要在回調函數中占用太長時間。

- timer.deinit()

  反初始化 timer.

- timer.callback(*fun*)

  i. fun : timer 中斷的回調函數。

```
>>> import gpb
>>> a=gpb.timer(1)
>>> a.init(freq=20)
>>> a.callback(lambda t:gpb.LED(1).toggle())
```

2.4 Sensor 特別函式庫

2.4.1 Function

2.4.1.1 Sensor module 相關函數

- sensor.start()
  啟動 CMOS 模組。
- sensor.stop ()
  停止 CMOS 模組。
- sensor.set_framesize(*N*)
  設定輸出影像大小。
  - ✓ sensor.QVGA (*default*)
  - ✓ sensor.VGA
  - ✓ sensor.HD(*720P*)
  - ✓ sensor.QQVGA
  - ✓ sensor.QQQVGA

  範例 ： `sensor.set_framesize(sensor.QVGA)`
- sensor.set_pixformat(*N*)
  設定影像格式。
  - ✓ sensor.YUYV or sensor.YUV422
  - ✓ sensor.RGB565 (default)
  - ✓ sensor.UYVY
  - ✓ sensor.GP420
- sensor.snapshot()
  返回 image class 影像截圖。
- sensor.pscaler_start(*N*)
  啟動 pscaler, 影像開始輸出到內部暫存(frame buffer).
  - ✓ N = 0 , 影像到 DRAM 路徑。
- sensor.pscaler_stop(*N*)
  停止 pscaler, 影像停止輸出到內部暫存(frame buffer).
  - ✓ N = 0, 影像到 DRAM 路徑。

- sensor.set_vflip(*N*)

  設定影像垂直翻轉。

  ✓ N = 0，影像不水平翻轉, N=1，影像水平翻轉。

- sensor.set_hmirror(*N*)

  設定影像水平翻轉。

  ✓ N = 0，影像不水平翻轉, N=1，影像水平翻轉。

範例 ：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM926
6EJ-S
>>> import sensor
>>> sensor.start()
>>> sensor.set_framesize(1)
>>> sensor.set_pixformat(0)
>>> sensor.pscaler_start(0)
>>> buf = sensor.snapshot()
>>> buf
2794048
>>> sensor.pscaler_stop(0)
>>> sensor.stop()
>>>
```

2.4.2 Classes

2.4.2.1 Class BH1750 luminance sensor

光感測器模組 BH1750 驅動 class.

1. 構造函數

- class sensor.BH1750(*scl, sda, addr, mode*)

  i. scl : I2C SCL Pin

  ii. sda : I2C SDA Pin

  iii. addr : 0x23 或 0x5C

  iv. mode :

  ✓ BH1750.CONT_LOWRES

  ✓ BH1750.CONT_HIRES_1

  ✓ BH1750.CONT_HIRES_2

  ✓ BH1750.ONCE_HIRES_2

  ✓ BH1750.ONCE_HIRES_1

  ✓ BH1750.ONCE_LOWRES

2. Methods

- BH1750.LUX(*mode*)

  讀取光感測值。

  i. None：使用預設 ONCE_HIRES_2 模式。

  ii. Mode：

  - ✓ BH1750.CONT_LOWRES
  - ✓ BH1750.CONT_HIRES_1
  - ✓ BH1750.CONT_HIRES_2
  - ✓ BH1750.ONCE_HIRES_2
  - ✓ BH1750.ONCE_HIRES_1
  - ✓ BH1750.ONCE_LOWRES

  iii. 返回：

  光照度值。

  範例：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP32
6EJ-S
>>> import sensor
>>> from sensor import BH1750
>>> lux = BH1750('C1','C0',0x23,BH1750.CONT_LOWRES)
>>> lux.LUX()
1023
>>> lux.LUX(lux.ONCE_HIRES_1)
1023
>>> lux.LUX(lux.ONCE_HIRES_2)
1791
>>>
```

2.4.2.2 Class DHT11 digital humidity temperature sensor

數位濕度與溫度感測器 DHT11 驅動 class.

1. 構造函數

- class sensor.DHT11(*pin*) or class dht.DHT11(*pin*)

  i. pin：模組 data Pin.

2. Methods

- DHT11.humidity ()

  讀取濕度值。

- DHT11. temperature ()

  讀取溫度值。

2.5 Bluetooth 特別函式庫

2.5.1 Function

2.5.1.1 Bluetooth Module 相關函數

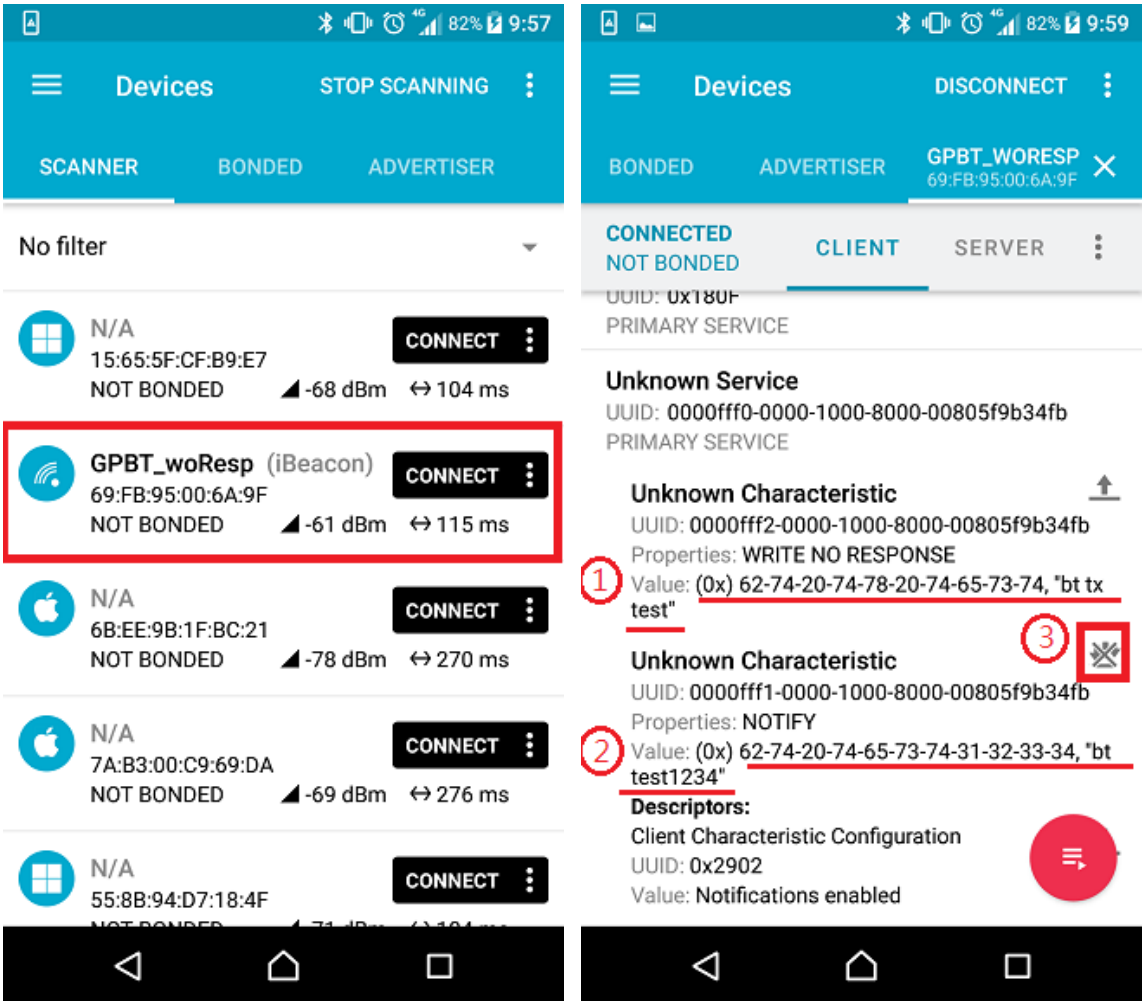
- bluetooth.active(*N*)
  i. N：啟動/停止藍牙模組。
- bluetooth.advertise(*n, name*)
  i. n：廣播週期，單位 msec.
  ii. name：藍牙裝置名稱。
- bluetooth.read()
  i. 返回：藍牙資料
- bluetooth.write(*N*)
  i. N：要傳送的藍牙字串
- bluetooth.avaliable()
  i. 返回：
     ✓ 1: 有藍牙資料。
     ✓ 0 無藍牙資料。

範例：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM92
6EJ-S
>>> import bluetooth
>>> bluetooth.active(1)
>>> bluetooth.advertise(100,'GPBT_woResp')
>>> bluetooth.read()
b'bt tx test'
>>> bluetooth.write('bt test1234')
>>> bluetooth.advertise(0)
>>> bluetooth.active(0)
>>>
```

2.5.2 Bluetooth Module 使用流程範例

1. 啟動藍牙並開始廣播 ID.
2. 手機(或是藍牙裝置) 掃描 BT 裝置與連線。
3. 手機(或是藍牙裝置) 傳送資料到 BT 裝置。 (紅色標記①)
4. 手機(或是藍牙裝置) 開啟 Notify 通知。(紅色標記③)
5. BT 裝置傳送資料到手機, 手機(或是藍牙裝置)接收到資料。 (紅色標記②)



2.6 Voice Recognition(VR) 特別函式庫

2.6.1 Function

2.6.1.1 VR 相關函數

- voice_recognition.start()
  啟動語音辨識模組。
- voice_recognition.stop ()
  停止語音辨識模組。
- voice_recognition.get_id()
  返回語音辨識結果。
- voice_recognition.load_database(*database*)
  i. database : vr database bin file.
- voice_recognition.close()
  關閉語音辨識模組。

範例 ：

```
>>> import voice_recognition
>>> voice_recognition.start()
>>> voice_recognition.get_id()
0
>>> voice_recognition.stop()
>>>
```

2.7 Face Recognition (Face ID)特別函式庫

2.7.1 Function

2.7.1.1 Face Recognition 相關函數

- face_recognition.start()
  啟動人臉偵測模組。
  - ✓ 返回 0：成功
  - ✓ 其他 ：失敗
- face_recognition.stop ()
  停止人臉偵測模組。
  - ✓ 返回 0：成功
  - ✓ 其他 ：失敗
- face_recognition.train()
  訓練人臉模型。(註: 30 秒內需要成功訓練 10 張人臉圖，超時則失敗。)
  i. 返回 Train ID 結果 ：
     - ✓ 1~5：成功
     - ✓ 其他 ：失敗
- face_recognition.recognize_start()
  開始人臉辨識。
  - ✓ 返回 0：成功
  - ✓ 其他 ：失敗
- face_recognition.recognize_stop()
  停止人臉辨識。
  - ✓ 返回 0：流程結束
  - ✓ 其他 ：失敗
- face_recognition.set_process (*N*)
  設定處理流程。
  i. N=0. 儲存辨識資料(default SD)
     - ✓ 返回 0：成功
     - ✓ 其他 ：失敗

  ii. N=1. 取得辨識資料(default SD)
     - ✓ 返回 0：成功
     - ✓ 其他 ：失敗

- face_recognition.get_face_id (*N*)

  取得人臉辨識結果，設定處理流程。

  i. N = 0 , 取到 face ID.
     - ✓ 返回 1~5：成功
     - ✓ 其他 ：失敗

  ii. N = 1, 取到 face ID 並且播聲音(default SD)。
     - ✓ 返回 1~5：成功
     - ✓ 其他 ：失敗

  範例 ：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM92
6EJ-S
>>>
>>> import face_recognition
>>> face_recognition.start()
0
>>> face_recognition.train()
1
>>> face_recognition.recognize_start()
0
>>> face_recognition.get_face_id(0)
1
>>> face_recognition.recognize_stop()
0
>>> face_recognition.set_process(0)
0
>>> face_recognition.set_process(1)
0
>>> face_recognition.stop()
0
>>>
```

2.8 RTK WI-FI 特別函式庫

2.8.1 Classes

2.8.1.1 Class network_WLAN

1. 構造函數

- class network_WLAN (*mode*)
  - i. mode : 0 為 AP mode
  - ii. 返回 :
    - ✓ mode 設定正確 : wifi init sdio / lwip doneinitial : ret 0.

```
>>> wlan=network_WLAN(0)
AP mode !!

wifi_init sdio/lwip doneinitial: ret 0
```

    - ✓ mode 設定失敗 : wrong mode.

```
>>> wlan=network_WLAN(1)
wrong mode !!
```

2. Method

- wlan.active(*is_active*)
  - i. 不帶參數時為查詢 Wi-Fi active 狀態。
  - ii. 帶有參數時，為是否 active Wi-Fi.
    - ✓ is_active = 1 : active Wi-Fi
    - ✓ is_active = 0 : 停止 Wi-Fi
- wlan.status ()

  返回連線狀態。
  - i. 回傳 0 代表沒有 client 連線。
  - ii. 回傳 1 則為 1 個 client 連線。
  - iii. ….以此類推。

- wlan. connect (*ssid,password*)

  連線到無線網路。

  i. 當 password 為空值時，加密方式為 open mode.

  ii. 當要使用 WPA2 加密方式時，password 必須至少 8 個字元。

- wlan.disconnect ()

  與無線網路斷線。

範例 ：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM926EJ-S
>>> import network
>>> from network import network_WLAN
>>> wlan=network_WLAN(0)
AP mode !!

wifi init sdio/lwip doneinitial: ret 0
>>> wlan.active(1)
ACTIVE AP mode!!!!
wifi_on ret: 0
>>> wlan.active()
1
>>> wlan.connect('GP','12345678')
WLAN_connect !!
channel = 13
Starting AP ...security_type: 400004, password_len:8
GP73cf97 started
rtk_wifi_module_connect wifi_start_ap end
[MEM] After WLAN Init, available heap 13272896
connect: ret 0
>>> wlan.disconnect()
WLAN_disconnect !!
rtk_wifi_module_disable wal_ap_mode_disable end
>>>
```

2.9 TCP Socket 特別函式庫

2.9.1 Function

2.9.1.1 TCP Socket 相關函式

- socket.setsockopt()
  設置 socket 連線。
- socket.bind()
  綁定 IP address 和 port.
- socket.listen()
  設定 TCP 連線為 listen mode.
- socket.accept()
  接受連線請求。
- socket.recvfrom(*size*)
  接收資料。
  - ✓ size : 接收的資料量。
- socket.sendto(buf)
  傳送資料。
  i. buf : 要傳送的資料。
- socket.close()
  關閉連線。

範例 :

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM926EJ-S
>>> import network
>>> from network import network_WLAN
>>> wlan=network_WLAN(0)
AP mode !!

wifi init sdio/lwip doneinitial: ret 0
>>> wlan.active(1)
ACTIVE AP mode!!!!
wifi_on ret: 0
>>> wlan.connect('GP')
WLAN_connect !!
channel = 13
Starting AP ...security_type: 0, password_len:0
GP73cf97 started
rtk_wifi_module_connect wifi_start_ap end
[MEM] After WLAN Init, available heap 13274496
connect: ret 0
>>> import socket
>>> socket.setsockopt()
Enter socket_cmd_start_server port 8081
socket_cmd_task: socket cmd server connection created(fe1fcc)
>>> socket.bind()
socket_cmd_task: socket cmd server bind to port 8081
>>> socket.listen()
socket_cmd_task: socket cmd server listen ok
gp cmd: server netconn 0xfe1fcc
>>> socket.accept()
gp cmd: client netconn 0xfe1f9c
>>> socket.recvfrom(2)
socket_cmd_service_receive
b'GP'
>>> socket.sendto('GP')
len: 2, pWrBuf 47 50
socket_cmd_service_receive
>>> socket.close()
TCP socket connection close!!
gp cmd: client netconn close, netconn 0xfe1f9c service 1
socket_close: server fe1f9c,  client fe1f9c
```

2.9.2 TCP Socket 測試步驟

1. 使用任一種 TCP 測試 APP.
   - ✓ IP 位址輸入 ：192.168.25.1
   - ✓ Port 輸入 ：8081

2. 當與 server 連接後，上方會顯示已連線狀態

3. 當與 server 連接後，輸入接收資料的 command “socket.recvfrom(*size*)”，並且在 APP 端發送要傳送的資料，terminal 會看到接收到的資料。

```
>>> socket.recvfrom(2)
socket_cmd_service_receive
b'GP'
```

4. 輸入傳送資料的 command “socket.sendto('GP')” ，可在 APP 的接收區看到傳送的資料

2.10 Image 特別函式庫

此模組支援部份 OpenMV Image 模組和類別的功能，在此簡述支援的功能函數，詳情請參照 OpenMV 關於 Image Module 和 Image Class 的說明文件。
http://docs.openmv.io/library/omv.image.html

註：以下兩節列出支援的功能，未列出的不保證可運行。

2.10.1 Function

2.10.1.1 Image 模組相關函數

- image.binary_to_grayscale(*binary_image_value*)
  將二元數值(0-1)轉換至灰階數值(0-255).
- image.binary_to_rgb(*binary_image_value*)
  將二元數值(0-1)轉換至 RGB888 三元組。
- image.binary_to_lab(*binary_image_value*)
  將二元數值(0-1)轉換至 LAB 三元組 L(0-100) A/B(-128 to 128).
- image.binary_to_yuv(*binary_image_value*)
  將二元數值(0-1)轉換至 YUV 三元組 Y/U/V(-128 to 128).

以下類推

- image.grayscale_to_binary(*grayscale_value*)
- image.grayscale_to_rgb(*grayscale_value*)
- image.grayscale_to_lab(*grayscale_value*)
- image.grayscale_to_yuv(*grayscale_value*)

- image.rgb_to_binary(rgb_tuple)
- image.rgb_to_grayscale(rgb_tuple)
- image.rgb_to_lab(rgb_tuple)
- image.rgb_to_yuv(rgb_tuple)

範例：

```
>>> import image
>>> image.rgb_to_yuv((128,255,80))
(69, -66, -48)
```

2.10.2 Classes

2.10.2.1 Class image

1. **建構函數**

- class image.Image(path [,copy_to_fb=False])
  class image.Image(w,h,pixformat [,copy_to_fb=False])
  支援 BINARY, GRAYSCALE, RGB565 格式 bmp/ppm 檔案格式，
  或是透過 sensor.snapshot()

  i. 參數 copy_to_fb
  - ✓ copy_to_fb = False :
    image 的數據會存在 MicroPython 的 heap 中。
  - ✓ copy_to_fb = True :
    image 的數據會載入另外配置的 frame buffer。
    - MicroPython heap 的大小有限，在操作尺吋較大的 image 時需要將 copy_to_fb 設為 True.
    - sensor.snapshot()所建構的 image 為例外，數據會存在 sensor 內部的暫存 buffer，隨著每次的 snapshot 不斷更新，可以透過 copy 相關的指令將數據複製到 heap 或 frame buffer.
    - 注意：Frame buffer 只能同時給一個 image 物件使用，當新的 image 物件使用了 frame buffer，原本使用 frame buffer 的物件其數據就會被覆蓋。

  ii. 本建構函數回傳值為新的 image 物件。

  初始化範例:

```
>>> import image, display, sensor
>>> sensor.start()
>>> sensor.pscaler_start(0)
>>> display.start()
>>> display.set_frame(image.Image("img.bmp")) //read
from file
>>> display.set_frame(image.Image(320, 240,
image.RGB565)) //create blank image
>>> display.set_frame(sensor.snapshot()) //create
from sensor snapshot
>>> img = image.Image(“img.bmp”, copy_to_fb=True)
>>> display.set_frame(img)
```

2. Methods

- image.width()
  回傳 image 寬的像素大小。
- image.height()
  回傳 image 高的像素大小。
- image.format()
  回傳 image 的格式：
  i. image.BINARY = 1
  ii. image.GRAYSCALE = 2
  iii. image.RGB565 = 3
  iv. image.BAYER = 5
  v. image.JPEG = 6
- image.size()
  回傳 image 的位元數(bytes).
- image.get_pixel(*x, y[, rgbtuple]*)
  回傳坐標點(x,y)的像素值，格式會因 image 而不同，不支持 JPEG.
- image.set_pixel(*x, y, pixel*)
  設定坐標點(x,y)的像素值，pixel 格式依 image 而不同，不支持 JPEG.

範例：

```
>>> img = sensor.snapshot()
>>> img.width()
320
>>> img.height()
240
>>> img.format()
3
>>> img.size()
153600
>>> img.get_pixel(20,20)
(82, 77, 90)
>>> img
{"w":320, "h":240, "type"="rgb565", "size":153600}
```

- image.mean_pool(*x_div, y_div*)
  image.mean_pooled(*x_div, y_div*)
  image.midpoint_pool(*x_div, y_div[, bias=0.5]*)
  image.midpoint_pooled(*x_div, y_div[, bias=0.5]*)

  將 image 以 x_div * y_div 大小的方格為單位快速縮小，每個方格取平均(mean)或中位(midpoint)為像素值。取中位時可以設定偏移量(bias)0.0~1.0，0.0 為最小值，1.0 為最大值。函數回傳縮小後的 image，*_pool()為修改原本的 image，*_pooled()為創建新的 image.

範例：

```
>>> img
{"w":320, "h":240, "type"="rgb565", "size":153600}
>>> img2 = img.mean_pooled(2,2)
>>> img2
{"w":160, "h":120, "type"="rgb565", "size":38400}
>>> img.mean_pool(4,4)
{"w":80, "h":60, "type"="rgb565", "size":9600}
```

- image.to_bitmap(*[copy=False[, rgb_channel=-1]]*)
  image.to_grayscale(*[copy=False[, rgb_channel=-1]]*)
  image.to_rgb565(*[copy=False[, rgb_channel=-1]]*)
  image.to_rainbow(*[copy=False[, rgb_channel=-1[, color_palette=sensor.PALETTE_RAINBOW]]]*)

  image 物件的格式轉換，參數 copy 為 True 表示創建新的 image，False 表示修改原始的 image，rgb_channel 設 0/1/2，表示從 R/G/B 三擇一取值(只適用從 RGB565 轉換).

- image.compress(*[quality=50]*)
  image.compressed(*[quality=50]*)

  將 image 壓製成 JPEG 格式，quality 為 JPEG 的壓縮品質(0-100)，compress()會破壞原始影像，compressed 會創建新的 image 物件。

- image.copy(*[roi[, x_scale[, y_scale [, copy_to_fb=False]]]]*)
  image.crop(*[roi[, x_scale[, y_scale [, copy_to_fb=False]]]]*)

  複製(copy)image 物件，可以指定 roi (選取的範圍(x,y,w,h))，x_scale, y_scale(x/y 軸縮放倍數，浮點數)，回傳值為新的物件。裁切(crop)和 copy 類似，差別在於會修改原始的 image 物件。這兩個函數不支持 JPEG 和 BAYER 格式的 image.
- copy_to_fb = False，image 的數據會存在 MicroPython 的 heap 中。
  copy_to_fb = True，image 的數據會載入另外配置的 frame buffer.

  MicroPython heap 的大小有限，在操作尺吋較大的 image 時需要將 copy_to_fb 設為 True.

  註： Frame buffer 只能同時給一個 image 物件使用，當新的 image 物件使用了 frame buffer，原本使用 frame buffer 的物件其數據就會被覆蓋。
- image.save(*path[, roi [,quality=50]*)
  將 image 存成 jpeg 檔案，path 為路徑，roi 為選取範圍(x,y,w,h)，quality 為 JPEG 的壓縮品質。

範例：

```
sensor.snapshot().to_grayscale().crop((10,10,100,100)
).compress().save("K:\\gray_crop.jpg")
```

- image.clear(*[mask]*)
  將 image 內容清為 0，mask 為另一個同大小的黑白 image，做為庶罩之用。

- image.draw_line(*x0, y0, x1, y1[, color[, thickness=1]]*)
  image.draw_rectangle(*x, y, w, h[, color[, thickness=1[, fill=False]]]*)
  image.draw_circle(*x, y, radius[, color[, thickness=1[, fill=False]]]*)
  image.draw_ellipse(*cx, cy, rx, ry, rotation[, color[, thickness=1[, fill=False]]]*)
  image.draw_cross(*x, y[, color[, size=5[, thickness=1]]]*)
  image.draw_arrow(*x0, y0, x1, y1[, color[, thickness=1]]*)

  繪圖指令：畫線(line) 矩形(rectangle) 圓(circle) 橢圓(ellipse) 十字(cross) 箭號(arrow)。除座標大小參數外的其他參數：color 指定顏色 thickness 指定線條粗線(pixel) fill 指定是否填滿，橢圓可設定 rotation(旋轉度數)。
- image.draw_string(*x, y, text[, color[, scale=1[, x_spacing=0[, y_spacing=0[, mono_space=True[, char_rotation=0[, char_hmirror=False[, char_vflip=False[, string_rotation=0[, string_hmirror=False[, string_vflip=False]]]]]]]]]]*)
  將字串畫在影像上，x,y 為座標，text 指定字串(只支援英文)，其他參數請看 openmv 的說明。
- image.draw_image(*image, x, y[, x_scale=1.0[, y_scale=1.0[, alpha=256[, mask=None]]]]*)
  將另一個 image 畫上去，可指定 scale、alpha、mask（另一個 image 物件)。

範例 ：

```
while (True):
    display.set_frame(sensor.snapshot().draw_circle(30,30,20))
```

2.11 uos 標準函式庫

uos 提供基本操作系統服務，此模組包含檔案系統訪問和掛載以及 uname 函數。

2.11.1 Function

2.11.1.1 uos 模組相關函數

- uos.uname()

  返回一個 tuple(可能是一個命名的 tuple)，其中包含關於底層機器和/或其作業系統的資訊。tuple 有五個欄位，順序如下:

  i. sysname ：底層系統名稱。
  ii. nodename ：網路名稱(可以與 sysname 相同)。
  iii. release ：底層系統的版本。
  iv. version : MicroPython 版本和構建日期。
  v. machine ：底層硬體(如主機板、CPU)的識別字。

範例 ：

```
>>> uos.uname()
(sysname='gpboard', nodename='gpboard', release='1.9.4', version='v1.9.4-409-g
434975de-dirty on 2018-07-25', machine='GP329XXXA Board with ARM926EJ-S')
```

- uos.sync()

  同步所有檔案系統。

- uos.chdir()

  改變目前的目錄。

- uos. getcwd()

  獲取目前的目錄。

範例 ：

```
>>> uos.getcwd()
'/'
>>> uos.chdir('/sd')
>>> print(uos.getcwd())
/sd
```

- uos. ilistdir([dir])

  該函數返回一個迭代器，該迭代器將生成與它所列出目錄中的條目相對應的 3 tuple。無參數情況下，列出目前的目錄，否則列出由 dir 指定的目錄。

  i. tuple 的形式為 (*name, type, inode[, size]*) :

  - ✓ name 為一個字串（若 dir 為一個位元組物件，則名稱為位元組）且為條目的名稱；
  - ✓ type 為一個指定條目類型的整數，其中目錄為 0x4000，常規檔為 0x8000；
  - ✓ inode 為一個與檔的索引節點相對應的整數，而對於沒有這種概念的檔案系統來說，可能為 0。
  - ✓ 一些平臺可能會返回一個 4 元組，其中包含條目的*size*。對於檔條目，size 是表示檔大小的整數，如果未知，則為-1。 對於目錄項，其含義目前尚未定義。

  範例 :

```
>>> import gpb
>>> from gpb import sdcard as sdc
>>> import uos
>>> uos.VfsFat.mkfs(sdc())
>>> vfs = uos.VfsFat(sdc())
>>> uos.mount(vfs, '/sd')
>>> uos.getcwd()
'/'
>>> uos.chdir('/sd')
>>> print(uos.getcwd())
/sd
>>> print(list(uos.ilistdir()))
[]
>>> uos.mkdir('foo')
>>> print(list(uos.ilistdir()))
[('foo', 16384, 0, 0)]
```

- uos. listdir([dir])

  若無參數，則列出目前的目錄；否則將列出給定目錄。

範例 ：

```
>>> print(list(uos.listdir()))
['foo']
```

- uos.mkdir(*path*)

  創建一個新目錄。

範例 ：

```
>>> uos.mkdir('foo')
>>> print(list(uos.ilistdir()))
[('foo', 16384, 0, 0)]
```

- uos.rmdir(*path*)

  刪除一個目錄。

範例 ：

```
>>> uos.mkdir('foo')
>>> uos.mkdir('bar')
>>> print(list(uos.listdir()))
['foo', 'bar']
>>> uos.rmdir('foo')
>>> print(list(uos.listdir()))
['bar']
```

- uos.remove(*path*)

  刪除一個檔案。

- uos.rename(*old_path, new_path*)

  重新命名檔案。

- uos.stat(*path*)

  獲取檔案或目錄的狀態。

- uos.statvfs(*path*)

  獲取檔案系統的狀態。

  按照以下順序返回一個具有檔案系統資訊的 tuple:

  i. f_bsize ：檔案系統塊大小。
  ii. f_frsize ：碎片大小。
  iii. f_blocks ： f_frsize 單元中 fs 的大小。
  iv. f_bfree ：空閒塊的數量。
  v. f_bavail ：非特權用戶的免費塊數。
  vi. f_files ：索引節點的數量。
  vii. f_ffree ：空閒索引節點的數量。
  viii. f_favail ：非特權使用者的免費空閒索引節點的數量。

ix. f_flag ：掛載標誌。

x. f_namemax ：最大檔案名長度。

與索引節點相關的參數：f_files、f_ffree、f_avail、f_flags參數可能會返回 0，因為它們在特定於埠的實現中不可用。

範例:

```
>>> uos.statvfs('/sd')
(32768, 32768, 477252, 477250, 477250, 0, 0, 0, 0, 255)
```

2.11.1.2 文件系統安裝

某些端口提供虛擬檔案系統（VFS），並能夠在此 VFS 中掛載多個“實際”檔案系統。 檔案系統物件可以安裝在 VFS 的根目錄下，也可以安裝在根目錄下的子目錄中。 這樣可以動態靈活地配置 Python 程式可以看到的檔案系統。 具有此功能的端口提供 mount()和 umount() 函數，以及由 VFS 類表示的各種檔案系統實現。

- uos.mount(*fsobj, mount_point, *, readonly*)

  將檔案系統物件 fsobj 掛載在 VFS 中 mount_point 字串給出的位置。 fsobj 可以是一個具有 mount() 方法的 VFS 物件，也可以是一個塊設備。

  i. 如果是塊設備，則自動檢測檔案系統類型(如果沒有識別出檔案系統，則會引發異常)。 mount_point 可以是 ‘/’ 以便將 fsobj 掛載在根目錄下， 或者是 ‘/<name>’ 可以將其掛載在根目錄下的子目錄中。

  ii. 如果 readonly 是 True，那麼檔案系統是唯讀安裝的，在掛載過程中，在檔案系統物件上調用 mount() 方法。

  iii. 如果 mount_point 已經掛載將引發 OSError(EPERM).

- uos.umount(*mount_point*)

  卸載檔案系統。mount_point 可以是命名掛載位置的字串，也可以是以前掛載的檔案系統物件。 在卸載過程中，在檔案系統物件上調用 umount() 方法。

  i. 如果沒有找到 mount_point ，將引發 OSError(EINVAL).

2.11.2 Classes

2.11.2.1 VfsFat 類別相關函數

1. 建構函數

- class uos.VfsFat(*block_dev*)

  創建一個使用 FAT 檔案系統格式的檔案系統物件。FAT 檔案系統的存儲由 block_dev 提供。 這個構造函數創建的物件可以使用 mount() 來掛載。

2. Methods

- VfsFat.mkfs(*block_dev*)

  在 block_dev 上建構一個 FAT 檔案系統。

範例:

```
>>> import gpb
>>> from gpb import sdcard as sdc
>>> import uos
>>> uos.VfsFat.mkfs(sdc())
```

- VfsFat. open(*path, attrubute*)

  i. path : 檔案路徑。

  ii. attribute :

  - ✓ “r”：以只讀方式開啟檔案，該檔案必須存在。
  - ✓ “r+”：以讀/寫方式開啟檔案，該檔案必須存在。
  - ✓ “w”：開啟只寫檔案，若檔案存在則長度清為 0，即該檔案內容消失，若不存在則建立該檔案。
  - ✓ “w+”：開啟可讀/寫檔案，若檔案存在則檔案長度清為零，即該檔案內容會消失。若檔案不存在則建立該檔案。

  iii. 其餘請參考 C 標準函式庫說明。

  範例:

```
f = vfs.open("footest1.txt", "w+")
len = f.write("  Hello World! I'm GP board")
print("create footest1.txt, length =", len)
f.close()
f = vfs.open("footest1.txt", "r+")
print(f.read())
f.close()
```

3. 下列 method 功能同 uos 模組下的對應函數 :

- VfsFat. ilistdir()
- VfsFat.mkdir()
- VfsFat.rmdir()
- VfsFat.chdir()
- VfsFat.getcwd()
- VfsFat.remove()
- VfsFat.rename()
- VfsFat.state()
- VfsFat.statvfs()
- VfsFat.mount()
- VfsFat.unmount()

2.12 Audio encode 特別函式庫

2.12.1 Function

2.12.1.1 audio encode 相關函數

- audio_encode.init()
  啟動錄音模組。
- audio_encode.start(*path*)
  開始錄音。
  i. path：檔案名稱。
- audio_encode.stop()
  停止錄音。
- audio_encode.uninit()
  關閉錄音模組。

範例：

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP3
6EJ-S
>>> import audio_encode
>>> audio_encode.init()
>>> audio_encode.start("test2.wav")
>>> audio_encode.stop()
>>> audio_encode.uninit()
>>>
```

2.13 Audio decode 特別函式庫

2.13.1 Function

2.13.1.1 audio decode 相關函數

- audio_decode.init()
  啟用播放模組。
- audio_decode.start(*path*)
  開始播放。
  i. path : 檔案名稱。
  ii. 支援格式 : mp3 / wav
- audio_decode.stop()
  停止播放。
- audio_decode.uninit()
  關閉播放模組。
- audio_decode.mute()
  靜音播放模組。
- audio_decode.unmute()
  取消靜音播放模組。
- audio_decode.pause()
  暫停播放模組。
- audio_decode.resume()
  繼續播放模組。
- audio_decode.volume_up(*step*)
  增加音量。
  i. step : 每次增加音量，音量最大值 63，超過會固定值為 63.
- audio_decode.volume_down(*step*)
  降低音量
  i. step : 每次增加音量，音量最大值 0，超過會固定值為 0.

- audio_decode.beep(*time_ms, freq, volume*)
  i. time_ms : 持續時間,單位 msec.
  ii. freq : 頻率 0 ~ 12
  iii. volume : 音量 0 ~ 63

範例 :

```
>>> import audio_decode,audio_encode,gpb
>>> audio_decode.init()
>>> audio_decode.start("test1.wav")
>>> audio_decode.mute()
>>> audio_decode.unmute()
>>> audio_decode.pause()
>>> audio_decode.start("test1.wav")
>>> audio_decode.pause()
>>> audio_decode.resume()
>>> audio_decode.volume_up(10)
>>> audio_decode.volume_up(10)
>>> audio_decode.volume_down(10)
>>> audio_decode.start("test1.wav")
>>> audio_decode.stop()
>>> audio_decode.uninit()
```

2.14 Object Detect 特別函式庫

2.14.1 Function

2.14.1.1 Object Detect 相關函數

```
typedef struct
{
    INT16S x;
    INT16S y;
    INT16S width;
    INT16S height;
} gpRect;

typedef struct {
    int color_result_idx; // -1, 0, 1, 2, ..., MAX_COLOR_COUNT-1
    float x; //x:0~1 y:0~1
    float y;
    float cb_size; // 0~1
} cb_python_output;

typedef struct {
    INT32U cb_blob_plot_state[OBJ_RECT_MAX];
    gpRect gp_output[OBJ_RECT_MAX];
    cb_python_output cb_output[OBJ_RECT_MAX];
} gpObjectRect;

typedef struct image {
    int w;
    int h;
    int bpp;
    union {
        uint8_t *pixels;
        uint8_t *data;
    };
    gpObjectRect object_detect_result;
            } image_t;
```

- object_detect.start()
  啟動物件偵測模式。
  i. 返回 0：成功。
  ii. 其他 ：失敗。
- object_detect.stop ()
  停止物件偵測模組。
  i. 返回 0：成功。
  ii. 其他 ：失敗。
- object_detect. set_face (*image*)
  i. image: 根據畫面，執行人臉偵測模型。
    - ✓ 返回 0：無偵測結果。
    - ✓ 返回 1：偵測成功。
    - ✓ 其他: 偵測失敗。

  ii. 偵測結果放在 ：Image-> object_detect_result. gp_output[0].
    - ✓ gp_output[0]. Width
      - 為 0：沒有偵測到物件。
      - 非 0 值 ：有偵測到物件。
- object_detect. set_face_train (*image, face_train_id,train_image_cnt*)
  i. image: 根據畫面，訓練人臉模型。
  ii. face_train_id：訓練人臉 ID，範圍 1~5.
  iii. train_image_cnt: 訓練人臉影像張數，範圍 10 張。
  iv. 10 張影像訓練尚未完成時 ：
    - ✓ 返回 100000：訓練成功。
    - ✓ 返回 0：無偵測到人臉。
    - ✓ 其他 ：訓練失敗。

  v. 10 張影像訓練完成 ：
    - ✓ 返回 1~5(Train ID)：訓練成功。
    - ✓ 其他 ：失敗。
- object_detect.set_face_recognition (*image*)
  i. image: 根據畫面，開始人臉辨識。
    - ✓ 返回 1~5(Recognition ID)：成功。
    - ✓ 其他 ：失敗。
- object_detect. set_ball (*image*)
  i. image：根據畫面，執行球體偵測模型。
    - ✓ 返回 0：無偵測結果。
    - ✓ 返回 1：偵測成功。
    - ✓ 其他 ：偵測失敗。

ii. 偵測結果放在 : Image-> object_detect_result. gp_output[0].
- ✓ gp_output[0]. width :
  - 返回 0 :沒有偵測到物件。
  - 其他 : 有偵測到物件。

- object_detect. set_color_ball_train (*image, color_ball _train_id, train_image_cnt*)

i. color_ball _train_id: 訓練色球 ID，範圍 1~3。
ii. train_image_cnt: 訓練色球 ID 影像張數，範圍 10 張。
iii. 10 張影像訓練尚未完成時:
- ✓ 返回 100000 : 訓練成功。
- ✓ 其他 : 訓練失敗。

iv. 10 張影像訓練完成:
- ✓ 返回 1~3(Train ID):成功。
- ✓ 其他 : 失敗。

- object_detect.set_color_ball_track (*image*)

i. image: 根據畫面，執行手掌偵測模型。
ii. hand_mode:
- ✓ 0 : 手掌
- ✓ 1 : 左 45 度手掌
- ✓ 2 : 右 45 度手掌
  - 返回 0 : 無偵測結果。
  - 返回 1 : 偵測成功。
  - 其他 : 偵測失敗。

iii. 偵測結果放在 : Image-> object_detect_result. gp_output[0].
- ✓ gp_output[0]. Width:
  - 返回 0 : 返回 0 :沒有偵測到物件。
  - 其他 : 有偵測到物件。

- object_detect. set_car (*image, car_mode*)

i. image: 根據畫面，執行車子偵測模型.
ii. car_mode:
- ✓ 0: 白天車子。
- ✓ 1: 晚上車子。
  - 返回 0 : 無偵測結果。
  - 返回 1 : 偵測成功。
  - 其他 : 偵測失敗。

iii. 偵測結果放在:Image-> object_detect_result. gp_output[0].

- ✓ gp_output[0].width:
  - 返回 0: 沒有偵測到物件。
  - 其他 : 有偵測到物件。

♦ object_detect.set_process (*image,type,mode*)

設定處理流程.

i. Type:0,處理人臉辨識資料

- ✓ mode =0. 儲存辨識資料(default SD)
- ✓ mode =1. 取得辨識資料(default SD)

ii. Type:1,處理色球辨識資料

- ✓ mode =0. 儲存辨識資料(default SD)
- ✓ mode =1. 取得辨識資料(default SD)

iii. Type:2,將偵測到結果畫在圖上

- ✓ mode =0. 偵測到結果不畫在圖上
- ✓ mode =1. 偵測到結果畫在圖上
  - 返回 0: 成功。
  - 其他: 失敗。

範例 :

```
MicroPython v1.9.4-409-g434975de-dirty on 2018-07-25; GP329XXXA Board with ARM92
6EJ-S
>>>
>>> import sensor,display,object_detect
>>> sensor.start()
>>> sensor.pscaler_start(0)
>>> display.start()
>>> object_detect.start()
0
>>> img = sensor.s
start           stop            set_framesize   snapshot
set_pixformat   set_hmirror     set_vflip
>>> img = sensor.snapshot()
>>> object_detect.s
start           stop            set_face        set_face_train
set_face_recognition            set_ball
set_color_ball_train            set_color_ball_track
set_hand        set_fist        set_car
>>> object_detect.set_f
set_face        set_face_train  set_face_recognition
set_fist
>>> object_detect.set_face(img)
0
>>> display.set_frame(img)
>>>
```