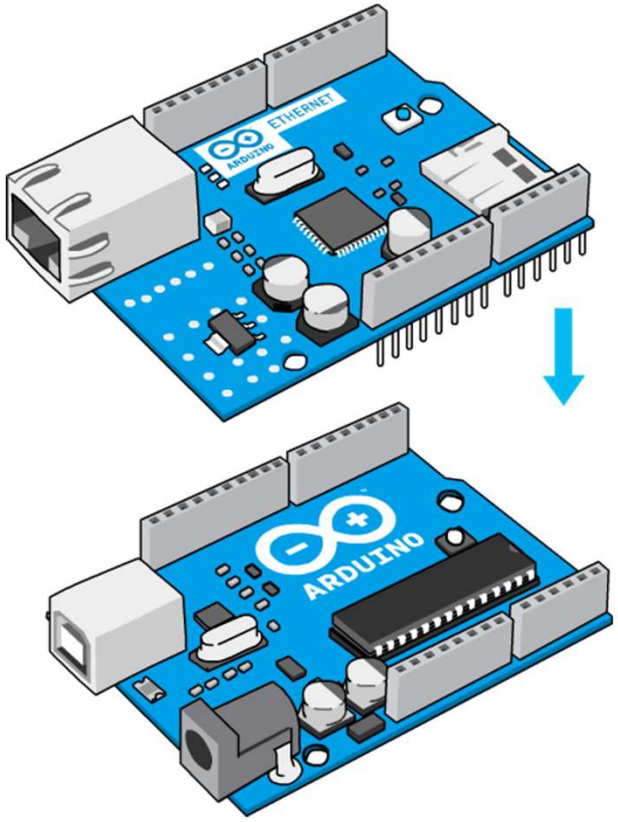
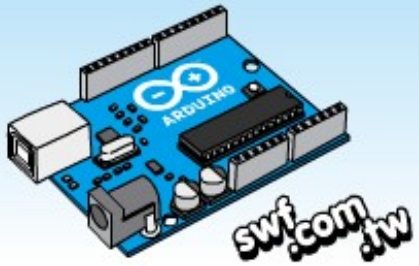
# 第八章 LED矩陣控制

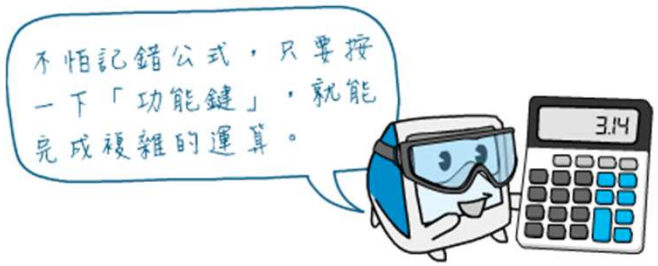


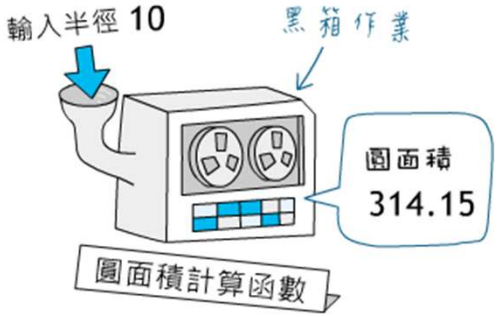

- 建立自訂函數
- 認識變數的有效範圍
- LED矩陣簡介
- 顯示靜態與動態矩陣圖像
- 指標與程式記憶體




# 建立自訂函數

具有特定功能並且能被重複使用的程式碼，叫做「函數」。

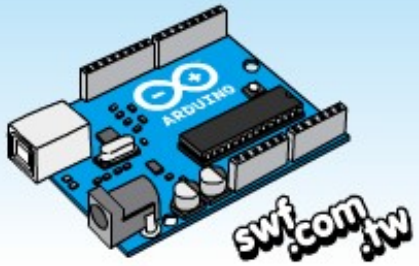



自訂函數的語法範例

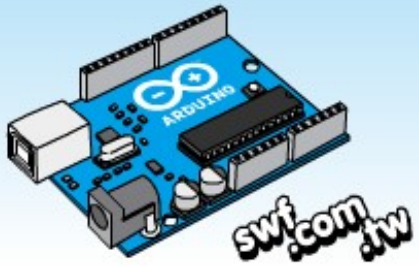
```
傳回值類型  函數名稱 (參數1, 參數2, ...) {
    運算式1;
    運算式2;
        :
    return 運算結果 ;
}
```

```
float cirArea() {
  int r = 5;
  float area = 3.14 * r * r;
  Serial.println(area);
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  cirArea();
  delay(2000);
}
```



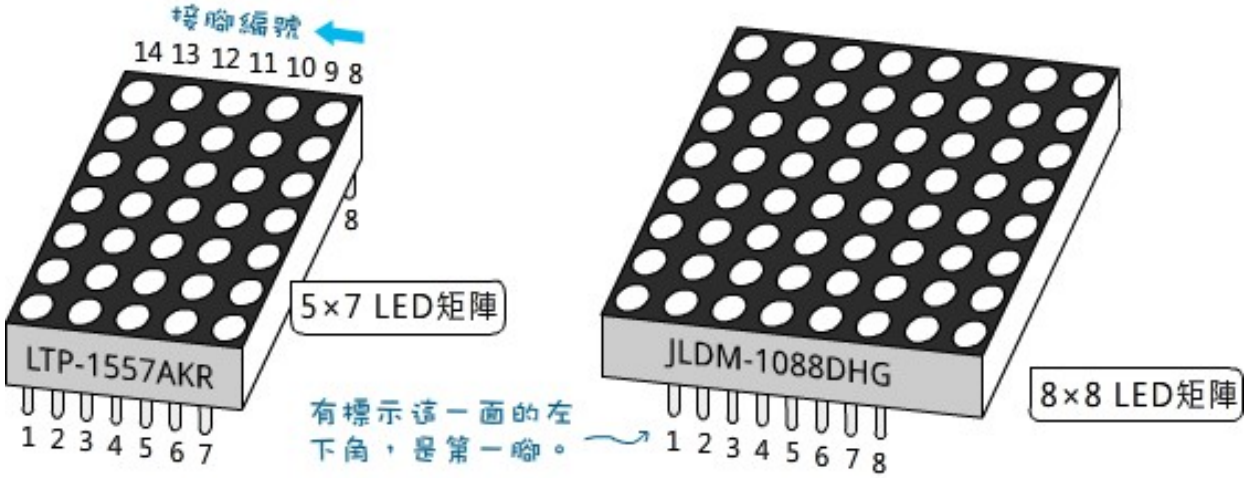

# 變數的有效範圍

```
int age = 20;                    ← 在室外宣告的變數：全域變數

void check() {
  int age = 10;                  ← 在室內宣告的變數：區域變數
  Serial.print("function: ");
  Serial.println(age);
}

void setup() {
  Serial.begin(9600);
  check();
  Serial.print("setup: ");
  Serial.println(age);
}

void loop() {
}
```

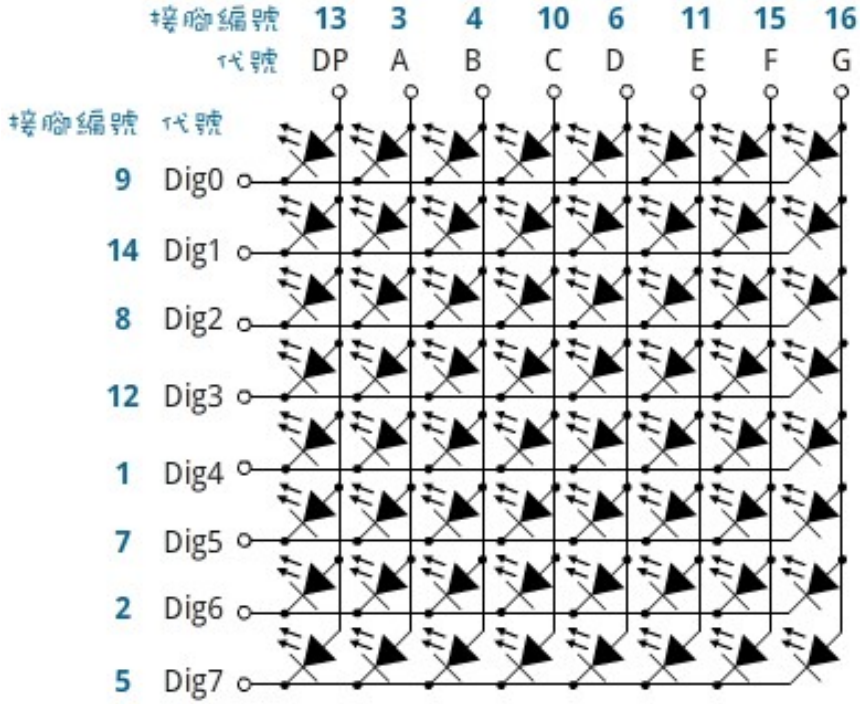


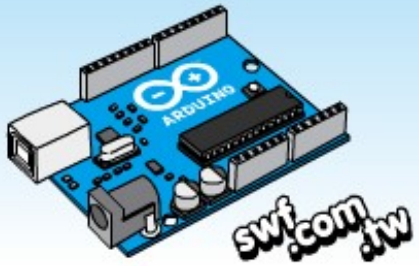

# LED矩陣簡介

LED矩陣（LED Matrix）是一種把數十個LED排列封裝在一個方形元件的顯示單元



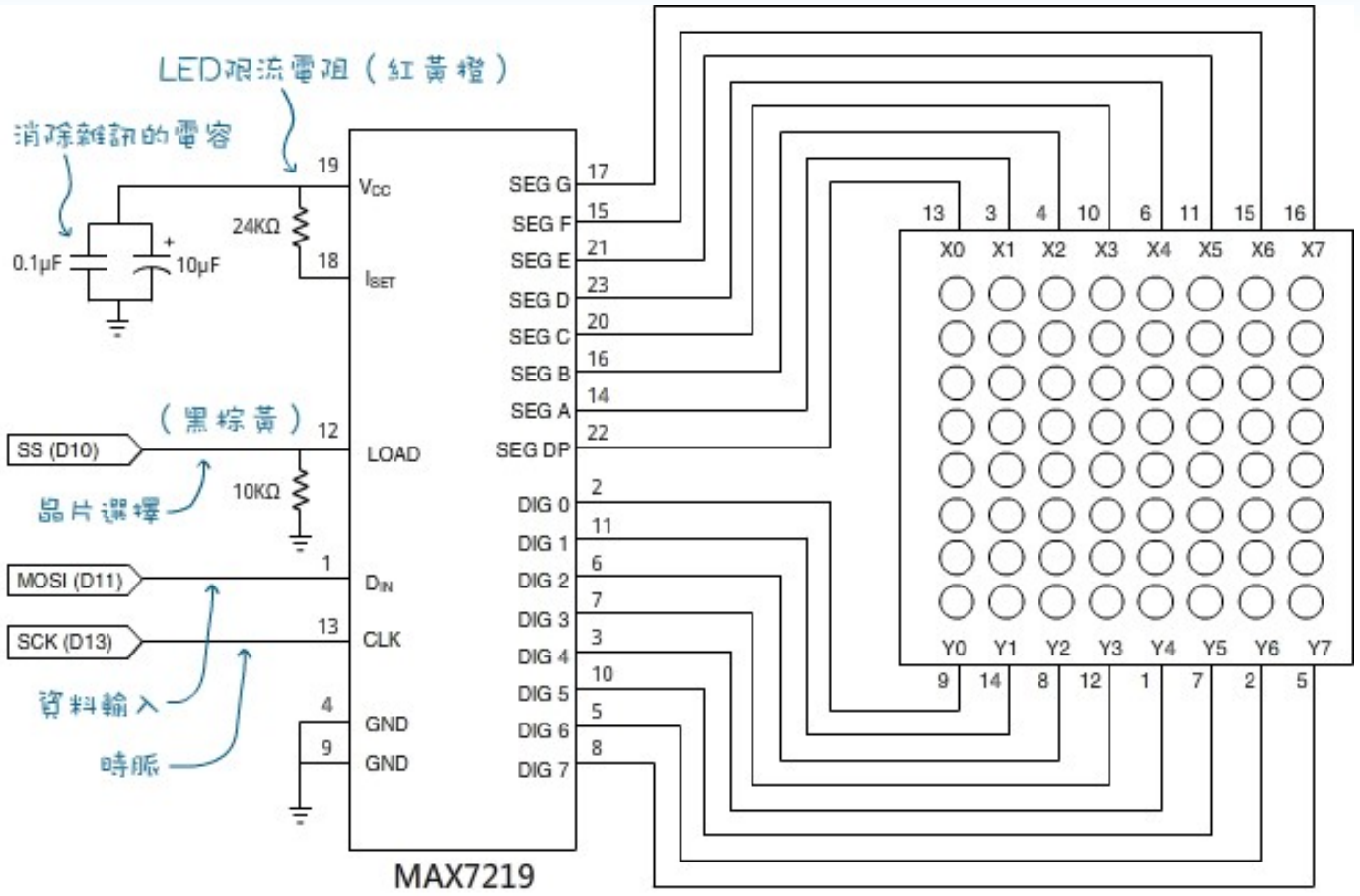

LED矩陣的內部結構



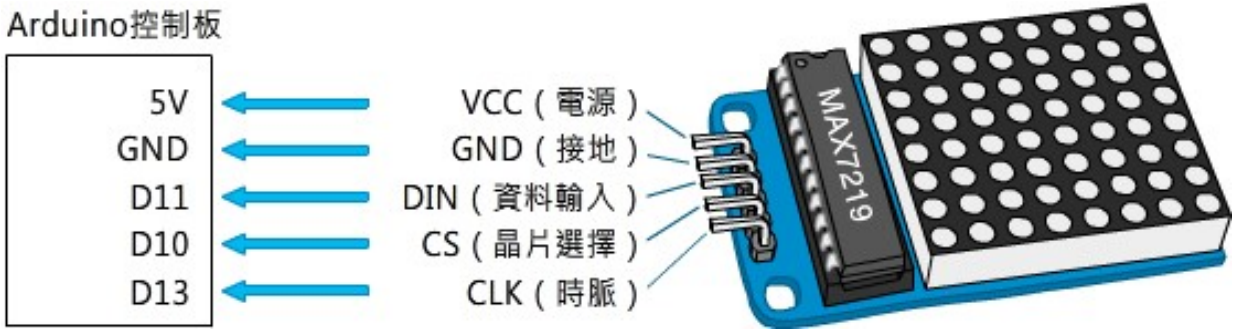


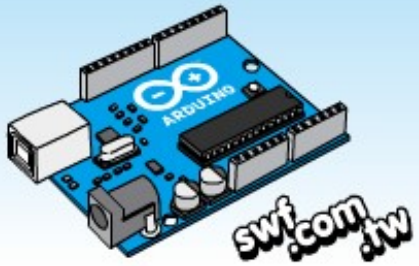

# LED矩陣電路

驅動一個8×8單色
LED矩陣的電路圖



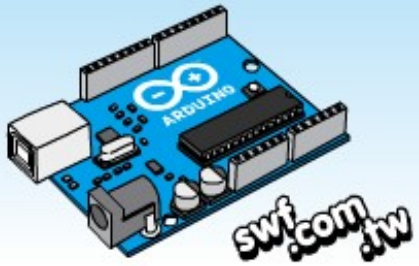

可以採用市售的MAX7219
與8×8rm LED矩陣的套件






# LED矩陣 單個字顯示

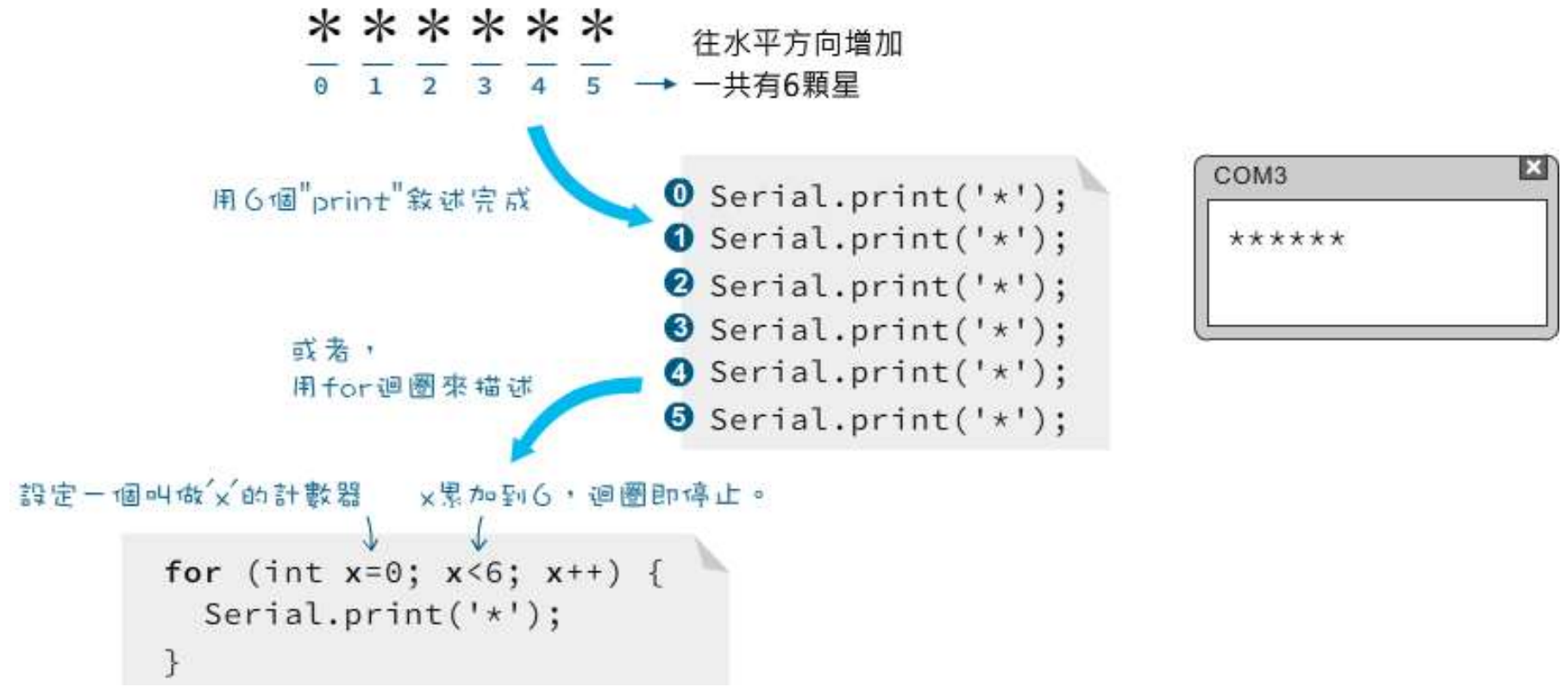
```
#include <SPI.h>
byte sym[8]={0xC0,0x78,0x1E,0x13,0x13,
0x1E,0x78,0xC0};  //英文字母A
const byte NOOP=0x0;  //從此起為固定輸入，啟動LED矩陣
const byte DECODEMODE=0x9;
const byte INTENSITY=0xA;
const byte SCANLIMIT=0xB;
const byte SHUTDOWN=0xC;
const byte DISPLAYTEST=0xF;
void max7219(byte reg, byte data) {
  digitalWrite(SS,0);
  SPI.transfer(reg);
  SPI.transfer(data);
  digitalWrite(SS,1);   }
void setup() {
pinMode(SS,OUTPUT);
digitalWrite(SS,1);
SPI.begin();
max7219(SCANLIMIT,7);
max7219(DECODEMODE,0);
max7219(INTENSITY,8);
max7219(DISPLAYTEST,0);
max7219(SHUTDOWN,1);   //到此結束
for (byte i=0; i<8; i++)  // 清除LED顯示為空白
max7219(i+1, 0); }
void loop() {
for (byte i=0; i<8; i++)
max7219(i+1,sym[i]);   }
```

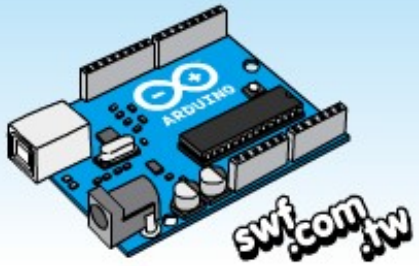



# 在序列埠監控視窗輸出矩形排列的星號（一）

如何呈現6個水平排列的星號？

- 用6個"print()"函數顯示星號。
- 用一個for迴圈完成。



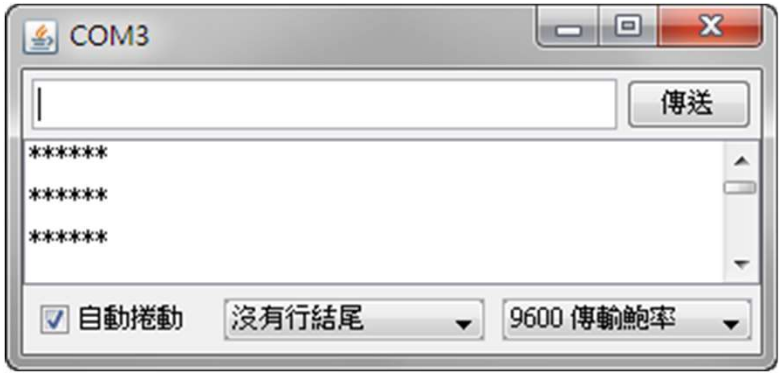



# 在序列埠監控視窗輸出矩形排列的星號（二）

```
0 * * * * * *
1 * * * * * *
2 * * * * * *
```

往垂直方向增加，
一共有三組，
每一組有6個星號。

分成3段for迴圈來完成




完成6×3排列顯示效果的雙重迴圈程式

```
❶ for (int x=0; x<6; x++) {
    Serial.print('*');
  }
  Serial.print('\n');
❷ for (int x=0; x<6; x++) {
    Serial.print('*');
  }
  Serial.print('\n');
❸ for (int x=0; x<6; x++) {
    Serial.print('*');
  }
  Serial.print('\n');
```

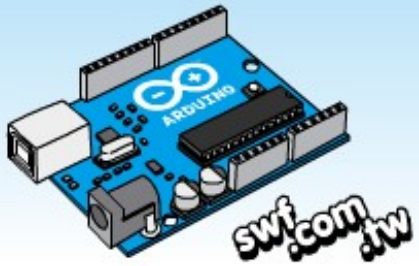
顯示6個星號，
加上1個「新行」結尾。

外層的計數器叫做'y'

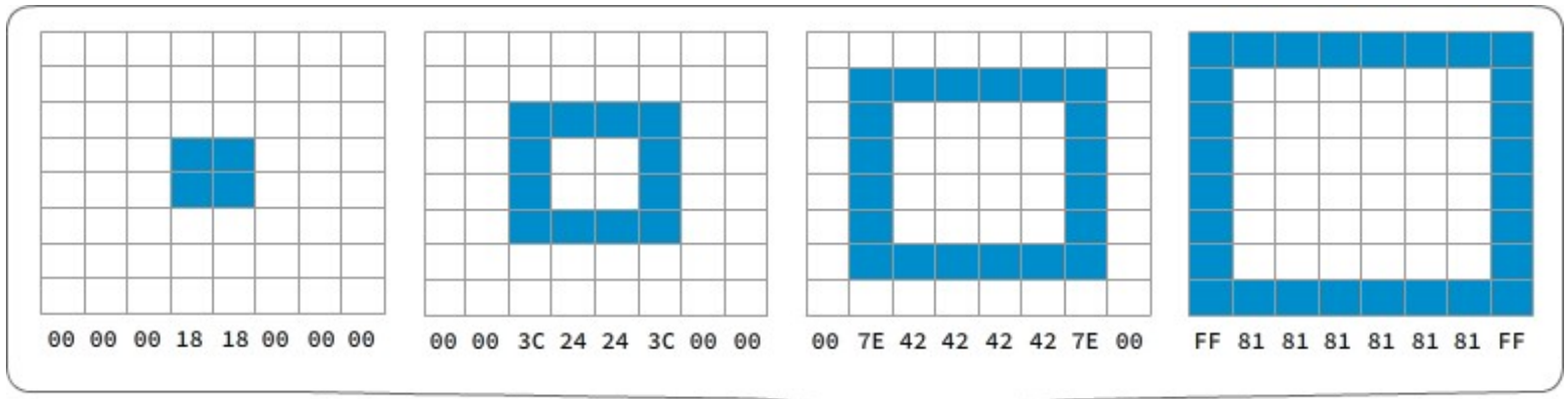
```
for (int y=0; y<3; y++) {
  for (int x=0; x<6; x++) {
    Serial.print('*');
  }
  Serial.print('\n');
}
```

這些敘述將被執行3次，
每次顯示6個星號。

使用雙重for
迴圈描述




# 顯示動態LED矩陣圖像

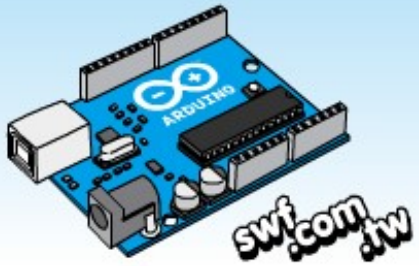



二維陣列定義LED矩陣圖像：

四組樣式　每組有八行

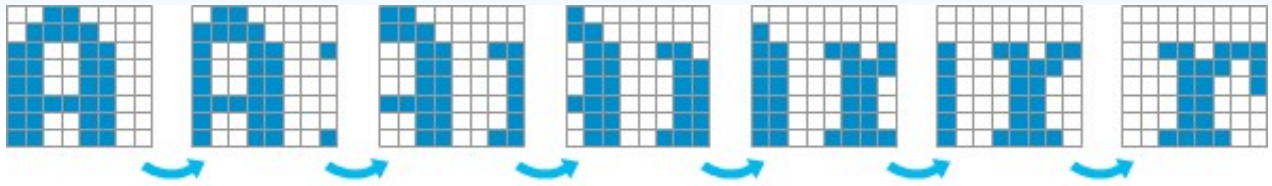
```
const byte sprite[4][8] = {
  { 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00 },
  { 0x00, 0x00, 0x3C, 0x24, 0x24, 0x3C, 0x00, 0x00 },
  { 0x00, 0x7E, 0x42, 0x42, 0x42, 0x42, 0x7E, 0x00 },
  { 0xFF, 0x81, 0x81, 0x81, 0x81, 0x81, 0x81, 0xFF }
};
```

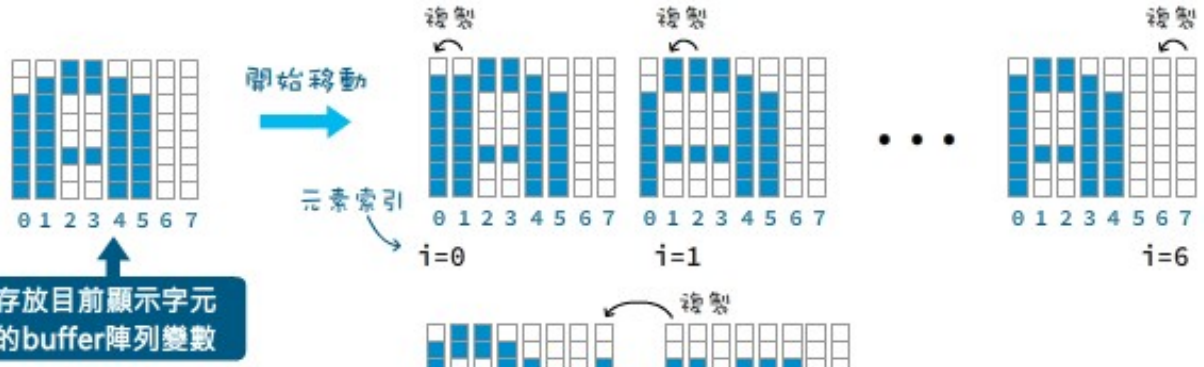
雙重迴圈程式先讀取第一張圖片裡的八行資料，再切換到下一張讀取：




# LED矩陣跑馬燈

先把動畫字元暫存在變數，再逐行移動（複製）其中的數據，就能產生捲動效果。

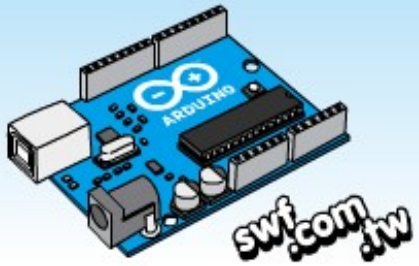





```
傳入下一個字元編號
void scroll(byte chr) {
  for (byte j=0; j<8; j++) {
    for (byte i=0; i<7; i++) {
      buffer[i] = buffer[i+1];
      max7219 (i + 1, buffer[i]);
    }
    buffer[7] = font[chr][j];
    max7219 (8, buffer[7]);
    delay(100);
  }
}
```

捲動目前的字元

逐行複製下個字元到目前的7行

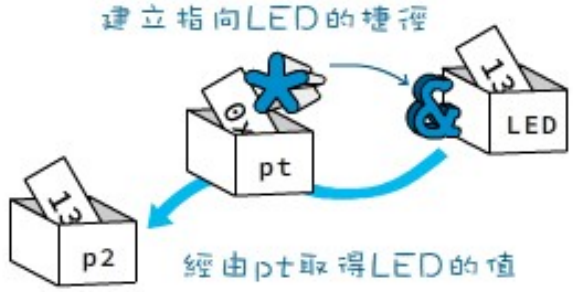



# 指標與「程式記憶體」

除了透過名稱，還可以透過記憶體位址存取變數資料。

```
byte LED = 13;
byte p1 = LED;
```


```
byte LED = 13;
byte *pt = &LED;
byte p2 = *pt;
```




在常數宣告敘述中加入PROGMEM關鍵字，可避免資料被複製到主記憶體。

```
byte fonts [256] [8] PROGMEM = {
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  { 0x7E, 0x81, 0x95, 0xB1, 0xB1, 0x95, 0x81, 0x7E },
      :
```

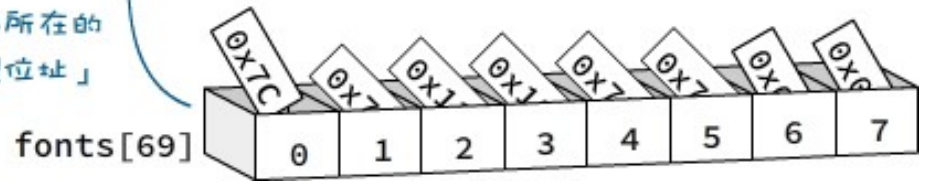
保存在程式記憶體

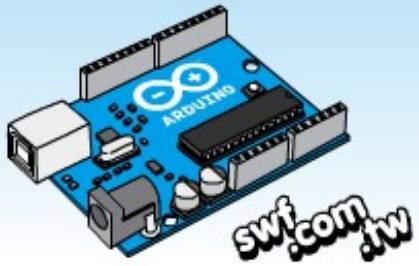
讀取程式記憶體的值，必須透過「指標」。

```
byte chr;
chr = pgm_read_byte (&fonts [69] [0]));
```

整個敘述代表「提取第69個字的第0」

提取此資料所在的「記憶體位址」






- 一分鐘回饋:
- https://goo.gl/forms/0C6jWOW5MTX9paos1