Chapter 3

變數與常數

本章是介紹程式設計的一些基本觀念，例如：程式語言的保留字、識別字、資料分類與資料型態、變數與常數的宣告、變數資料型態的轉換與資料的數位化。以上內容或許有點瑣碎，也有點枯燥，卻又很重要，就像練功前的蹲馬步，所以請讀者們耐心讀完。

3-1 保留字與識別字

保留字（Keywords）

保留字（又稱關鍵字）是任一程式語言已事先賦予某一識別字（可識別的文字或字串，稱爲識別字）一個特別意義，所以程式設計者不得再重複賦予不同的用途，就像現實生活中，電視、冰箱已經有特別意義了，所以沒有人取名字爲電視或冰箱。Arduino 也是一樣，例如 if 已賦予決策敘述，程式設計者當然不得再定義 if 爲另外的用途，以下是 Arduino 語言的保留字，loop、setup、break 等，大部分擷取 C/C++語言精華。

STRUCTURE

The elements of Arduino (C++) code.

Sketch
loop()
setup()

Control Structure
break
continue
do...while
else
for
goto
if...else
return
switch...case
while

Further Syntax
#define (define)
#include (include)
/* */ (block comment)
// (single line comment)
; (semicolon)

Arithmetic Operators
% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

Comparison Operators
!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

Boolean Operators
! (logical not)
&& (logical and)
|| (logical or)

Pointer Access Operators
& (reference operator)
* (dereference operator)

Bitwise Operators
& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Compound Operators
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)

識別字（Identifiers）

眞實的世界裏，每個人、事及物都有一個名稱，程式設計亦不例外，於程式設計時我們必須爲每一個變數、常數、函式命名，以上所有變數、常數、函式等名稱，統稱爲程式語言的識別字。先以數學爲例，數學命名規則是，已知數用 a,b,c，未知數用 x,y,z；物理則習慣用 v 代表速度，t 代表時間；而 Arduino 語言的識別字命名規則如下：

1. 識別字必須是以字母（大小寫的 A 至 Z）或底線(_)開頭。例如，以下是一些合法的識別字。

   a
   i
   sum
   _sum
   Income

以下是一些非法的識別字。

```
7eleven          /*   不能由數字開頭   */
%as              /*   不能由符號開頭   */
```

2. 識別字由字母開頭後，僅可由字母、底線及數字組合而成，且不得包含空白與符號。例如，以下是一些合法的識別字。

```
a123
a123b
_a_b
```

以下是一些非法的識別字。

```
A=               /*   不能含有 = 號 */
sum!             /*   不能含有 ! 號 */
Age#3            /*   不能含有 # 號 */
a c              /*   不能含空白    */
c+3              /*   不得含有加號   */
```

3. 識別字的大小寫均視爲不同，例如 Score、score 及 SCORE 皆代表不同的識別字。
4. 識別字不得使用保留字，如 if、for 等。但是，if1、fora 等則可使用。
5. 識別字要用有意義的單字，例如，sum、avg、StudentNumber 或 AverageIncome。除非有效範圍很小（或稱生命週期極短）的變數才用 x、i 或 a 等當識別字，也千萬不要用 k23erp 等這種沒意義又難記的識別字。
6. 識別字有多個單字時，中間可以加上底線（_），例如上例的 StudentNumber 可寫成 student_Number，若擔心打字不靈光亦可寫成 Stu_Num、stu_num、stuNum 或 stunum，其中 stuNum 又稱駝峰表示法，因爲大寫字母看起來像駱駝駝峰一樣，這樣可以避免鍵入底線的困擾、且提昇閱讀效率，Arduino 的函數則採用此種命名習慣。

3-2 資料種類與資料型態

資料種類

程式設計的主要工作是處理人類數位化的資料，Arduino 語言所能處理的資料種類分別有數值（含整數及浮點數）、字元、字串和布林值等資料。

➲ 整數常數（Integer Constants）

Arduino 語言可以處理的整數常數有四種進位方式，分別是十進位（Decimal）、二進位（Binary）、八進位（Octal）及十六進位（Hexadecimal）。其中十進位則以我們平常書寫數字的方式即可，例如 12；二進位則以 B 開頭，例如，B11 則代表十進位的 3；八進位則應以 0 開頭，例如 011 代表十進位的 9(1*8+1)；十六進位應以 0x 開頭，例如，0x11 代表十進位的 17(1*16+1)。請鍵入以下程式，並觀察執行結果。

```
void setup() {
    Serial.begin(9600);
    int a=B11;
    int b=011;//零，不是字母 O
    int c=0x11;//零 x，不是字母 OX
    Serial.println(a);
    Serial.println(b);
    Serial.println(c);
}void loop() {}
```

➲ 浮點常數（Floating Point Constants）

數字中含有小數點或指數的稱爲浮點數。以指數爲例，E 或 e 表示 10 的次方，例如 0.0023、2.3E-3 及 2.3e-3 都是表示相同的浮點數；又例如 2.3E+2 則代表 230。浮點數可使用標準寫法或科學符號法表示，例如 321.123 即爲標準寫法，1.23e+4 即爲科學符號表示法。

➲ 字元常數（Character Constants）

使用單引號(')圍住的單一字元，稱爲字元常數，例如 'A' 或 'a' 等。Arduino 語言使用 8 bits 儲存字元，所以可表示 256 個字元，這樣即可包含

大小寫英文字母、數字、標點符號及其它特殊符號的所有 ASCII 碼。

➲ 字串

Arduino 語言中，字串以雙引號(")所圍住，例如，"Gwosheng", "台灣" 等。

➲ 布林值

Arduino 語言使用 1 代表布林的 true，0 代表布林的 false。

資料型態（Data Type）

前面已介紹人類經常使用的一些資料常數，每一種資料常數均需要不同大小的記憶體儲存，電腦爲了有效率的處理這些資料，所以有資料型態的規劃，也就是大的資料用大盒子裝，小的資料用小盒子裝，如此才可節省記憶體，並加快處理效率。反過來說，若不分資料大小，通通用大盒子裝資料，那將會非常浪費記憶體，也拖垮執行效率。下表是 Arduino 語言的常用資料型態。unsigned 表無號整數，即非負數整數：

資料型態	中文名稱	佔用記憶體的大小（位元）	所能代表的數值的範圍	備註
byte	位元組	8	0～255	
int	整數	16	-32768～32767	
long	長整數	32	-2147483648～2147483647	
float	浮點數	32	+/-3.4E+-38	
double	倍精度浮點數	32	+/-3.4E+-38	Arduino 的 double 同 float
unsigned char	正字元	8	0~255	
unsigned int	正整數	16	0~65535	
unsigned long	正長整數	32	0~42949667295	
char	字元	8	-128~127	
bool	布林	8	true or false	boolean 也可，但不鼓勵
String	字串			

3-3 變數和常數的宣告

變數宣告

變數的功能是用來輸入、處理及儲存外界的資料，而變數在使用以前均要事先宣告才可使用。在一些舊式的 Basic 語言中，變數使用前並不需要事先宣告，卻也帶來極大的困擾。例如，以下敘述即爲變數未宣告的結果，編譯器便無法回應使用者在拼字上的錯誤，而造成除錯上的困難。

```
student=studend+1;
```

上式若事先宣告 student 如下：

```
int student;
```

則編譯器遇到 studend 時，便會出現 studend 未宣告的錯誤訊息，提醒使用者補宣告或注意拼字錯誤。其次，變數宣告的優點是可配置恰當的記憶體而提高資料的處理效率。例如，有些變數的值域僅爲整數，則不用宣告爲 float。此即爲小東西用小箱子裝，大東西用大箱子裝，才能有效運用空間與提升搬運效率。Arduino 語言的變數宣告語法如下：

```
資料型態 變數名稱[=初值];
```

例如，

```
byte a;
```

即是宣告變數 a 爲 byte 型態。又例如，

```
char b,c;
```

則是宣告變數 b,c 爲 char 型態。變數的宣告亦可連同初值一起設定，如下所示：

```
float d = 30.2;
```

宣告 d 是 float 型態，並設其初值是 30.2。以下程式，宣告 e 是 char 型態，且其初值是字元'a'，其敘述如下。

```
char e='a';
```

以下程式同時宣告兩個變數，且設定其初值。

```
int f1=3,f2=3;
```

以下程式可宣告布林型態：

```
bool g=true;
```

C/C++/Arduino 都可用字元陣列表示字串，以下程式可宣告 a 爲字串型態，請留意字元是單引號，字串是雙引號，關於陣列的使用，請看第七章。

```
char a[]="ABC";//字串用雙引號
```

C++/Arduino 才有字串型態 String，以下程式可宣告變數 h 爲字串型態：

```
String h="123";
```

變數經過宣告之後，編譯器即會根據該變數的資料型態配置適當的記憶體儲存此變數，所以若要提高程式的執行效率，則應儘量依照資料性質，選擇佔用記憶體較小的資料型態。

➲ 自我練習

請鍵入以下程式，編譯、除錯與執行程式。

題號	程式	結果

1	void loop() {}
	  int 7eleven;
	  int %as;
	  int _a;
	  int A=;
	  int sum! ;
	  int Age#3;
	  int a c ;
	  int c+3 ;
	  int Score;
	  score=4;
	  int if;
	}	

2	void setup() {
	  Serial.begin(9600);
	  byte a=266;
	  int b=32768;
	  unsigned byte c=-4;
	  long d=5;
	  float e=6;
	  double f=7;
	  char g='A';
	  g=66;
	  Serial.println(g);
	}void loop() {}	

3	void setup() {
	  // put your setup code here, to run once:
	  Serial.begin(9600);
	  int student=0;
	  student=studend+1;
	  Serial.println(student);
	  int a=3.4;
	  char b,c;
	  b="aa";
	  c='a';
	  Serial.println(a);
	  Serial.println(b);
	  Serial.println(c);
	  String d;
	  d=5;
	}
	void loop(){}
	void setup() {	

```
    Serial.begin(9600);
    long a=24*60*60;
    Serial.println(a);
    a=24*60*60L;
    Serial.println(a);
}void loop(){}
```

常數符號的宣告

跟變數一樣，常數符號亦需要記憶體儲存位置，與變數不同的是，常數符號正如名稱所示，常數符號在整個程式中都不會，也不能改變其值。程式設計有兩種表示常數的方式，一種是文字式（Literal），例如直接以 15 或 3.14159 表示某一常數；另一種是本單元的常數符號式（Symbolic）。因爲有些數字在程式中會不斷的重複出現，爲了增加程式的可讀性及減少程式鍵入的麻煩，此時即可用一個有意義的符號代替。我們可以利用 const 來定義常數，則該符號的值將永遠保持在你所宣告的符號，程式中任何位置均不可能改變其值，此稱爲常數符號，簡稱常數。const 宣告常數，語法如下：

```
const 型態 常數名稱=常數值;
```

例如，以下程式令 PI=3.14，常數符號通常均用大寫表示。

```
const float PI=3.14;
```

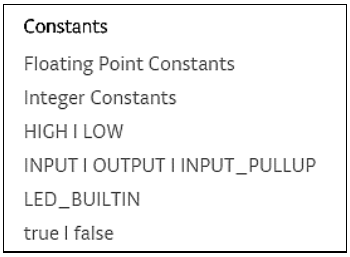
則每次要使用 3.14 時，只要填入 PI 即可。例如，以下程式可計算圓面積。

```
a=3*3*PI;
```

又例如，於銀行利息的計算，利率亦應以常數符號表示，程式中任何位置若需使用此利率時，均應以此常數符號代替，當要調整此利率時，只要在程式的最前面修改此常數符號的值即可。若未使用常數符號統一此值，則因此常數散落在程式中各處，而必須到處修改，萬一有些地方漏掉而未修改，則無法確保此值的一致性。（備註：#define 雖然也可以定義常數，但 Arduino 已經淘汰這種用法）

➲ 自我練習

1. 以下是 Arduino 已經定義的一些常數，請研讀其內容。



例如，您的程式可以這樣寫，其中 LOW、HIGH 都會變綠色，都是常數。

```
   b=digitalRead(pb);
   if (b==LOW)
      digitalWrite(led,HIGH);
   else
      digitalWrite(led,LOW);
```

2. 請鍵入以下程式，並觀察執行結果，請留意宣告爲常數 const，就不能再改變其值。

```
void setup() {
   Serial.begin(9600);
   const byte  a=3;
   Serial.println(a);
   a++;
   Serial.println(a);
}void loop(){}
```

變數與常數的有效範圍

在一個大型程式裏，一個程式是由一或數個函式組合而成。這些函式通常是由許多人合力完成，爲了避免彼此變數互相干擾，例如張三設 stunum=15，李四又設 stunum=20，則必然造成無法預期的錯誤，所以爲了防止變數互相干擾，才有變數的有效範圍（又稱爲變數的生命週期），茲將變數的有效範圍敘述如下：

任一變數的宣告，若無特殊聲明，均屬於區域變數，其有效範圍僅止於

該變數所在的程式區塊。例如，宣告在敘述區塊，則它的有效範圍僅止於該敘述區塊，別的敘述區塊是無法取得該區域變數的值；若宣告於函式中，則它的有效範圍僅止於該函式，其它的函式均無法取得該區域變數的值。例如，以下程式，變數 a 稱爲全域變數，其有效範圍是 setup( )及 loop( )全部函式，變數 b 的有效範圍則是僅在 setup()內，變數 c 的有效範圍則是僅在 loop()內，變數 i 的有效範圍僅在 for(){}迴圈內的程式區塊。

```
int a;
void setup() {
  int b;
}
void loop() {
  int c;
  for (int i=1;i<=5;i++){
    Serial.println(i);
    //i 的使用範圍僅在此 for 程式區塊內
  }
}
```

以上 for()後面接一個大括號{}，loop()後面也接一個大括號{}，這都稱爲一個程式區塊，各自都有一個獨立的區域變數範圍。以下程式，變數 b 也是獨立各自佔用一個記憶體，互不干擾。

```
void setup() {
  int b=2;
}
void loop() {
  char b='d';
}
```

以下程式，第一個變數 a，稱爲全域變數，第二個變數 a 稱爲 setup()區域變數，彼此獨立，各自佔用一個記憶體，互不干擾。但是 loop(){}內的變數 a，沒有宣告，則是使用全域變數的 a，所以還是 3。

```
int a=3;  //全域
void setup() {
  int a=4;  //值於 setup()內有效
```

```
  Serial.begin(9600);
  Serial.println(a);//4
}
void loop() {
  Serial.println(a);  //3
  delay(10000);
}
```

以下程式就不行，因爲重複宣告變數 b。

```
void setup() {
  int b=3;
  char b='a';
}
void loop() {
}
```

以下程式就可以，因爲 3 個變數 b 的有效範圍不同，各自獨立，沒有互相干擾。

```
int b=1;
void setup() {
    Serial.begin(9600);
    Serial.println(b);//1
}
void loop() {
    char b='4';
    Serial.println(b);//4
    for(byte b=0;b<=5;b++)
        Serial.print(b);//012345
    Serial.println();//跳列
    Serial.println(b);//4
    delay(10000);
}
```

3-4 資料型態轉換

變數的資料型態轉換

每一個變數宣告之後，即有屬於自己的型態，往後此變數均只能指派給相同型態的變數儲存。例如，以下程式，a 爲 int，b 也爲 int，兩者型態相同，所以 b 可指派給 a。

```
int a=3;
int b=8;
a=b;
```

以下變數 b 型態爲 String，兩者型態不相同，就不可指派給 a。

```
int a=3;
String b=”Horng”;
a=b;//錯
b=a;//錯
```

但是，資料型態相近的就可以互轉。例如，數值型態有 byte、int、long、float，這些型態都是數值，只是所能儲存的資料的範圍不同而已，這些資料型態執行階段可以互轉，此稱爲型態轉換。

在 Arduino 語言中型態轉換又可分爲隱含轉換（Implicit Conversion）與強制轉換（Explicit Conversion），分別說明如下：

➲ 隱含轉換（Implicit Conversion）

將值域小的型態轉爲值域大的型態，稱爲隱含轉換或轉型（Convert）。此種轉換，系統可自動處理並確保資料不會流失。例如，將 byte 轉爲 int，則因後者的值域比前者大，所以可順利的轉換。以下敘述可將型態爲 byte 的變數 a 指派給型態爲 int 的變數 b，且原值不會改變。

```
byte a=23;
int b;
b=a;
Serial.println(b);//23
```

結果是 23。

➲ 強制轉換（Emplicite Conversion）

將值域大的轉爲值域小的型態（如 int 轉爲 byte），則稱此爲強制轉換或稱爲鑄型（cast）。強制轉換的語法如下：

變數 1 = (變數 1 的型態) 變數 2;
變數 1 = 變數轉換函式 (變數 2);

例如，以下敘述可將型態是 int 的 a 變數指派給型態是 byte 的 b 變數。

```
void setup() {
   Serial.begin(9600);
   int a=23;
   byte b;
   b=a;
   Serial.println(b);//23
   b=(byte)a;
   Serial.println(b);//23
   b=byte(a); //byte() is function
}void loop() {}
```

結果是 23。強制轉換的風險比較大，有可能資料流失或溢位。例如，以下敘述將 float 型態強制轉換爲 byte 型態，將造成小數部份的流失。

```
void setup() {
   Serial.begin(9600);
   float a=3.4;
   byte b;
   b=a;
   Serial.println(b);//3
   b=byte(a);
   Serial.println(b);//3
   b=(byte) a;
   Serial.println(b);//3
}void loop() {}
```

以下敘述將 int 型態的 a 變數轉爲 byte，將造成溢位，輸出爲 1。

```
void setup() {
  Serial.begin(9600);
  int a=257;
  byte b;
  b=byte(a);
  Serial.println(b);//1
} void loop() {}
```

結果是 1，表示溢位。

➲ 自我練習

請於 Arduino 環境完成以下程式，並觀察執行結果。以下示範字串如何表示，字串與數值的不同、字串轉爲數值。（本題初學者先略）

```
void setup() {
  Serial.begin(9600);
  String a="123";//以字串型態表示字串
  String b="456";
  Serial.println(a+b);//字串相加
  Serial.println(a.toInt()+b.toInt());//先轉數值再相加
  char c[]="123";//以字元陣列表示字串
  char d[]="456";
  Serial.println(c);
  Serial.println(d);
  //Serial.println(c+d);//字元陣列無此運算子
  Serial.println((int)c+(int)d); //先轉數值再相加
}void loop() {}
```

3-5 資料數位化原理與方法

資料數位化的原理

電腦的最小記憶單元爲位元(bit)，每個位元僅能儲存 0 或 1，8 個位元稱爲一個位元組(Byte)，那資料如何儲存於電腦呢，答案就是要將資料數位化，此即爲資料數位化的原理。

資料數位化方法

日常生活可見的資料爲數值、字元、字串、聲音、圖片、影片，數值再分爲整數、與實數，以上資料數位化的方法各有不同，但最後都是將以上資料轉爲二進位的方式，才能儲存於電腦，以上資料的數位化方法，分別說明如下：

整數

所有整數都要先指定資料的長度，本例先假設是 8 位元，其餘 16 位元或 32 位元也是相同原理。其次，還要假設僅儲存正數，還是同時容納正負數。首先，以儲存正數爲例，那其二進位編碼如下，共可表示 0 到 255 的正數。

正數	二進位編碼
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
254	11111110
255	11111111

若要考慮正負數，那最高位元則用來表示正或負數，最高位元爲 0 時表示正數，所以可表示的範圍是 0 到 127 如下頁表；最高位元爲 1 時表示負數，那到底負多少，則要取 2 補數。所謂 2 補數是先取 1 補數再加 1，例如，

10000000

最高位元爲 1，表示負數，那到底負多少，先取 1 補數

01111111

再加 1，所以是

10000000

那就是代表

-128

又例如，

11111111

最高位元是 1，代表負數，先取 1 補數

00000000

再加 1

00000001

所以是代表 -1，如下表：

數字	二進位編碼
00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
00000101	5
00000110	6
00000111	7
01111110	126
01111111	127 最大正數
10000000	-128 最小負數
10000001	-127
11111110	-2
11111111	-1

由上表可知，若最高位元代表正負數，則 8 位元可表示的數值範圍爲-128～127。

▶範例 3-5a

示範正整數的數位化。請鍵入以下程式，並觀察 8 個 LED 的亮滅，請問與上表的正數是否相符。（電路同範例 2-3b）

➲ 輸出結果

請留意 8 個 LED 代表 8 位元，燈號的明就是 1，燈號的滅就是 0。

➲ 程式列印

```
void setup() {
  DDRA=B11111111;
  Serial.begin(9600);
}
unsigned char i=0;
void loop() {
  PORTA=i;
  Serial.println((int)i);
  delay(1000);
  i=(i+1)%256;
}
```

▶範例 3-5b

示範包含正負數整數的數位化。請鍵入以下程式，並觀察 8 個 LED 的亮滅，請問與上表的正負數是否相符。（電路同範例 2-3b）

➲ 輸出結果

1. 請留意數字 -128 的表示，燈號的明滅就是其二進位，本例會是 10000000。
2. 請留意數字 -1 的表示，燈號的明滅就是其二進位，本例會是 11111111。

```
void setup() {
  DDRA=B11111111;
  Serial.begin(9600);
}
char i=-128;
void loop() {
  PORTA=i;
```

```
  Serial.println((int)i);
  delay(1000);
  i=i+1;
}
```

➲ 自我練習

1. 請線上查詢 Arduino 資料型態 unsigned int 共使用多少 bit 代表 1 個整數，其所能代表的數字範圍爲何？其次，int 共使用多少 bit 代表 1 個整數，其所能代表的數字範圍爲何？

字元

鍵盤能用的字元有大小寫的 a,b,c、數字 0～9、還有一些控制字元，例如，跳列、歸位（回到該列最左邊）、跳頁、return 等，這些字元總共沒有超過 127 個，因此使用 1 個 byte 儲存就綽綽有餘，所以這些字元的編碼，在 1960 年就已經編碼完成，稱爲 ASCII 碼(American Standard Code for Information Interchange)，如下圖所示：

ASCII chart

The ASCII (American Standard Code for Information Interchange) encoding dates to the 1960's. It is the standard way that text is encoded numerically.

Note that the first 32 characters (0-31) are non-printing characters, often called control characters. The more useful characters have been labeled.

DEC Value	Character	DEC Value	Character	DEC Value	Character	DEC Value	Character
0	null	32	space	64	@	96	`
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8		40	(	72	H	104	h
9	tab	41	)	73	I	105	i
10	line feed	42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13	carriage return	45	-	77	M	109	m

由上圖可知共使用 7 位元，0 到 127，前面 32（0～31）爲控制字元，48～57 是 0～9 的數字，65～90 是大寫英文字元，97～122 是小寫字元。

▶範例 3-5c

示範輸出 ASCII 碼結果。請自行鍵入以下程式，並觀察燈號與輸出結果。（電路同範例 2-3b，請注意觀察，0～31 為控制字元，沒有輸出，32 是空白也沒有輸出）

```
void setup() {
  DDRA=B11111111;
  Serial.begin(9600);
}
char i=0;
void loop() {
  PORTA=i;
  Serial.print((int)i);Serial.print(":");Serial.println(i);
  delay(200);
  i=(i+1)%128;
}
```

▶範例 3-5d

示範字元的表示。請自行鍵入以下程式，並觀察燈號與輸出結果。（電路同範例 2-3b）

➲ 輸出結果

1. 請留意字元'a'的表示，且內部是以 97，燈號的明滅就是其二進位，本例會是 01100001。

```
void setup() {
  DDRA=B11111111;
  Serial.begin(9600);
}
char i;
void loop() {
  i='a';//字元用單引號
```

```
  Serial.print((int)i);Serial.print(":");Serial.println(i);
  PORTA=i;//97,0x61
  delay(1000);

  i='b';//字元用單引號
  Serial.print((int)i);Serial.print(":");Serial.println(i);
  PORTA=i;//98,0x62
  delay(1000);

  i='A';//字元用單引號
  Serial.print((int)i);Serial.print(":");Serial.println(i);
  PORTA=i;//65,0x41
  delay(2000);
}
```

➲ 自我練習

1. 請自行於 Arduino 線上手冊搜尋 ASCII。

字串

連續字元的集合稱爲字串，例如，

```
"ABC"
```

請留意前面字元用單引號，字串則用雙引號。C/C++/Arduino 都是使用字元陣列表示字串。例如，以下陣列 a 即可儲存以上字串。

```
   char a[]="ABC";//字串用雙引號
```

▶範例 3-5e

示範字串的表示。請自行鍵入以下程式，並觀察燈號與輸出結果。（電路同範例 2-3b）

➲ 輸出結果

1. 請留意 a[0]是字元 'A'，內部記憶體是 01000001，燈號會是 01000001，

a[1]是字元 'B'，a[2]是字元 'C'。

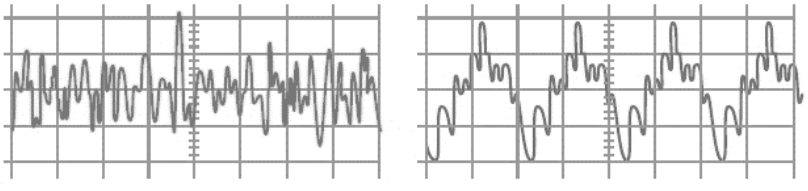
```
65:A
66:B
67:C
```

➲ 程式列印

```
void setup() {
  DDRA=B11111111;
  Serial.begin(9600);
}
void loop() {
  char a[]="ABC";//字串用雙引號

Serial.print((int)a[0]);Serial.print(":");Serial.println(a[0]);
  PORTA=a[0];//65,0x41
  delay(1000);


Serial.print((int)a[1]);Serial.print(":");Serial.println(a[1]);
  PORTA=a[1];//66,0x42
  delay(1000);


Serial.print((int)a[2]);Serial.print(":");Serial.println(a[2]);
  PORTA=a[2];//67,0x43
  delay(12000);
}
```

➲ 聲音

蚊子、蜜蜂飛舞、樹搖動等大自然的聲音、人類或樂器發出的聲音都是震動空氣所形成的聲波，如下圖所示，是一種類比信號，其頻率一直在變化。



聲音的數位化就是要將以上不斷變化的頻率擷取、儲存下來。要重新播放時，再使用電子元件震盪出這些頻率，那就可以將原音重現。要將以上不斷變化的頻率記錄下來，就要靠取樣與量化。分別說明如下：

取樣

取樣就是擷取當時的頻率，要接近原音，取樣頻率就要高，例如，CD 與 MP3 的取樣頻率是 44100Hz，DVD 音質的取樣頻率是 96000Hz（1 秒取樣 96000 次）。

量化

每一個取樣要使用多少 bit 記錄，稱爲量化。於上圖的波形，若只採用 1bit，那就只有將全部的頻率分爲 2 個等級，若只採用 2bits，那就只有將全部的頻率分爲 4 個等級，那聲音當然很粗糙，CD 可是採用 16bits，將聲音的頻率分爲 2^{16} 個等級，DVD 則用 24bits，也就是將聲音分爲 2^{24} 個等級，那當然較接近原音，下表是常見不同用途的聲音的取樣頻率、量化等級與音軌數。

聲音用途	取樣頻率（Hz）	樣本大小（bit）	聲道數量
電話通訊	11,025	8	1
收音機廣播	22,050	8	1
CD 音樂	44,100	16	2
DVD 音樂	96,000	24	2

聲音檔案大小

聲音檔案大小的公式是：

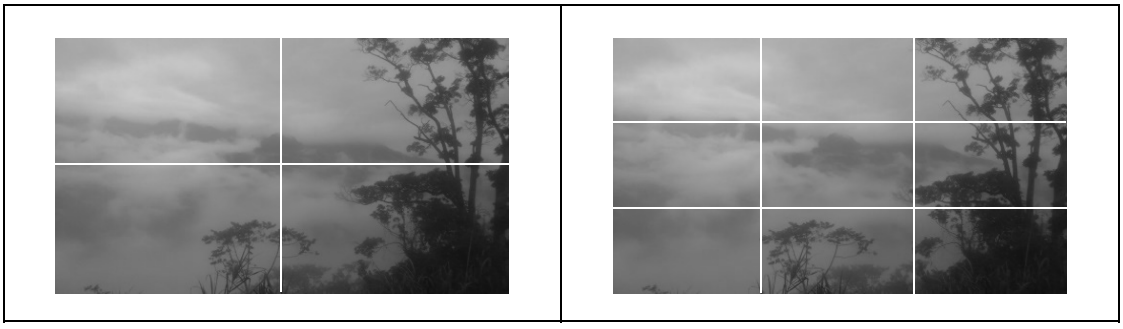
取樣頻率(Hz)*取樣大小(bit)*軌道數*秒數

例如，一個 2 秒 2 軌的 CD 檔案，其檔案大小是

44100*16*2*2 = 2,822,400 bit
2,822,400 bit /8 bit = 352,800 byte
352,800 byte /1024byte = 344k byte

圖片

圖片要數位化是先規劃取樣的大小，取樣的大小就是像素(pixel)。例如下圖左是將照片規劃爲 2*2 像素，下圖右是規劃爲 3*3 像素，像素越多，當然重繪時較能原圖重現，但是耗費記憶體也多，傳輸也費時，所以就要依照您圖片的用途，取一個平衡點。



請自行於檔案總管觀察圖片檔案所規劃的像素。其次，還要規劃每一個像素的顏色要多少 bit 儲存，若使用 1bit，那再漂亮細膩的色彩也只剩 2 個等級了。若使用 2bit，那全部的色彩只剩 4 個等級；我們目前所稱的『全彩』就使用 24bits，那就有 2^{24}=16,777,216 個色彩變化。

影像

因爲人類視覺有暫留現象，所以影像就是圖片的連續播放，其數位化原理也相同。