Java Tutorial

Chapter 3　圖形介面

3.1 圖形介面(Graphical User Interface, GUI)基礎

以往我們在寫程式的時候，我們在看結果或是輸入資訊都是藉由終端機(console)來進行。然而，這種指令式的互動方式對使用者來說並不是非常友善。使用者希望可以使用滑鼠來進行工作，並且進行較為視覺性的互動以及較多的資訊提供；因此我們就要藉由圖形介面來讓使用者與電腦程式有較為簡易的互動模式。圖形介面現在被廣泛運用，不管是手機或電腦，我們所看到的每個顯現在使用者面前的都是一種圖形介面的設計

然而，圖形介面與以往有很大的不同，在設計時的思考模式必須更為全面以及細心，因為有很多物件都會變成互相關聯，若是沒有設定好就很容易造成整個程式的錯誤出現。另一個圖形介面的特色就是會有很多的”事件”，舉例來說，在滑鼠變成一個很重要的操作媒介之後，就必須要設計很多滑鼠按下去之後會發生什麼事，又或是滑鼠掃過去的時候會發生什麼事，這些東西都是在以往與終端機互動的時候不需要去注意的事情。

3.2 Java AWT & Java Swing

在正式介紹 Processing 本身所提供的圖形介面函式之前，我們先介紹 JAVA 本身提供的圖形介面的 Tookit：AWT 與 Swing。

AWT 是 JAVA 官方第一個提供的套件，而 Swing 則是之後官方採納成為第二個官方 GUI 程式庫。兩者的差別在於， AWT 採用作業系統提供的視窗元件，所以利用 AWT 所設計的圖形介面程式會跟作業系統原本的視窗風格一致，Swing 則是利用 Java 原生程式碼重新繪出視窗元件，因此利用 Swing 設計的程式在各平台間並無外觀上的差異。在這邊我們元件都使用 Swing 作為主要的呼叫來源，然而有些功能我們也是要使用到 AWT 的方法。因此，要使用 AWT 要匯入 java.awt.*；另一方面，Swing 這個 Tookit 必須要先匯入 javax.swing.*的這個 package，它提供許多圖形介面元素，並且支援透過使用者編寫達到指定邊框、顏色、背景等等屬性的客製化元件。

基本上每個 Swing 裡的元件所呼叫的名字前面都會加一個”J”，例如我要叫一個 Frame，則我要呼叫的資料型態就是 JFrame；我需要一個 Button 就是宣告一個 JButton 的物件。這邊建立一個 frame 來做舉例。在創建 GUI 物件的時候於 constructor 中寫上字串會是該元件預設顯示的訊息；而後我們需要對這個 frame 做基本的大小設定；最後再讓它可以被看到。最後我們要呼叫 setDefaultCloseOperation 方法，這個方法是設定視窗標題列的關閉按鈕結束程式執行，提供 JFrame 類別的 static 屬性 EXIT_ON_CLOSE 當參數。

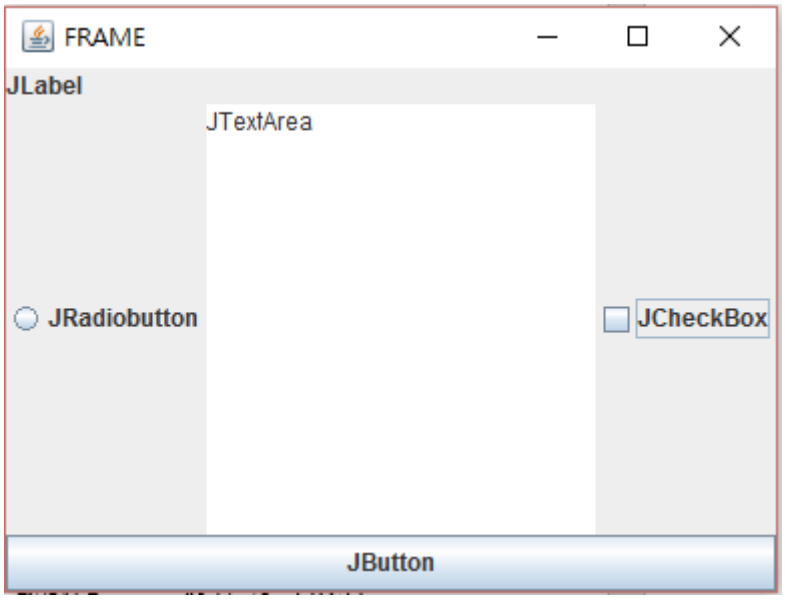
```
import javax.swing.*;

void setup(){
 JFrame frame = new JFrame("FRAME"); //make a frame
 frame.setSize(400,300);  // set the frame size
 frame.setVisible(true);  // show the frame
}
```

Frame 這個元件算是一個比較特別的元件，它不同於其他元件的地方在於它是一種容器(container)，可以用來容納其他元件。

元件名稱	類別名稱
視窗	JFrame
面板	JPanel
標籤	JLabel
按鈕	JButton
核取方塊	JCheckBox
選取方塊	JRadioButton
下拉式列表	JComboBox
列表	JList
捲動軸	JScrollBar
滑動軸	JSlider
文字方塊	JTextField
文字區域	JTextArea

上表展現了較常用的一些元件，這些元件一般來說除了 frame 之外都需要放在 container 裡面才能夠被顯示。

```
import java.awt.*;
import javax.swing.*;

void setup(){
 JFrame frame = new JFrame("FRAME");
 JCheckBox checkbox = new JCheckBox("JCheckBox");
 JRadioButton radiobutton = new JRadioButton("JRadiobutton");
 JButton button = new JButton("JButton");
 JLabel label = new JLabel("JLabel");
 JTextArea textarea = new JTextArea("JTextArea");

 frame.setSize(400,300);  // set the frame size
 frame.setVisible(true);  // show the frame
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 //put all the elments on the frame
 frame.getContentPane().add(BorderLayout.EAST, checkbox);
 frame.getContentPane().add(BorderLayout.WEST, radiobutton);
 frame.getContentPane().add(BorderLayout.SOUTH, button);
 frame.getContentPane().add(BorderLayout.NORTH , label);
 frame.getContentPane().add(BorderLayout.CENTER, textarea);
}
```



從上面的程式碼與執行結果可以看的出來我們需要一個 Frame 來承載這些不同的元件；然而值得注意的一點就是程式碼後面我們在增加這些元件的時候我們寫了”BorderLayout”，看到這個就必須要提到排版了。

3.3 排版(Layout)

在設計 GUI 程式的時候，有一些東西是特別需要去注意的，分別是視窗元件(widget)、排版(layout)以及事件(event)。我們在前面一節已經提到了一些元件的概念，然而，這些元件放進來之後到底要被放在哪邊呢？這時候排版就變得很重要了，一個 GUI 要能夠對使用者友善，因此每個元件被擺放的位置就變得相對重要許多了。

區域式版面配置	BorderLayout
流動式版面配置	FlowLayout
卡片式版面配置	CardLayout
格子式版面配置	GridLayout
帶狀式版面配置	GridBagLayout
盒子式版面配置	BoxLayout

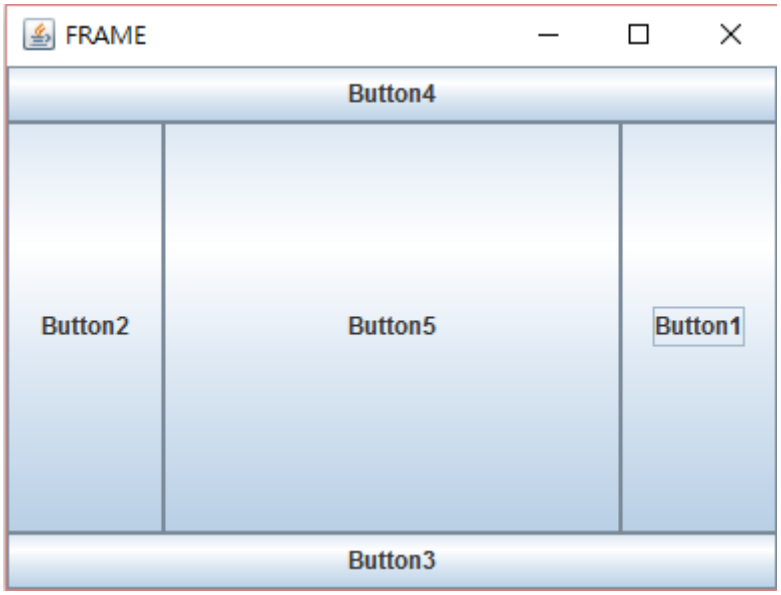
常用的排版有以上這幾種，我們並不會一一解釋每種排版，我們只舉幾個較常用的排版，其他可以再自行去做測試。在 JFrame 中，BorderLayout 是預設的排版配置，因此若當我們想要改變版面配置的時候，就必須要呼叫以下：

```
frame.setSize(400,300);  // set the frame size
frame.setVisible(true);  // show the frame
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new FlowLayout()); // set layout type
```

先呼叫完 setLayout 之後才開始放入各種元件並且同時進行該種配置的排版。

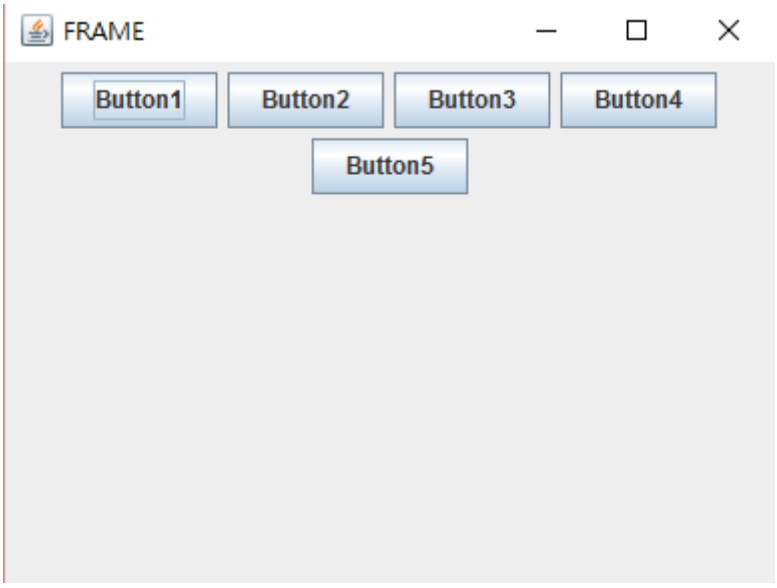
我們先從預設的 BorderLayout 講起，這種排版將整個視窗分成五個部分，分別是 North, South, East, West, Center，如下面舉例所顯示：

```
frame.getContentPane().add(BorderLayout.EAST, button1);
frame.getContentPane().add(BorderLayout.WEST, button2);
frame.getContentPane().add(BorderLayout.SOUTH, button3);
frame.getContentPane().add(BorderLayout.NORTH , button4);
frame.getContentPane().add(BorderLayout.CENTER, button5);
```



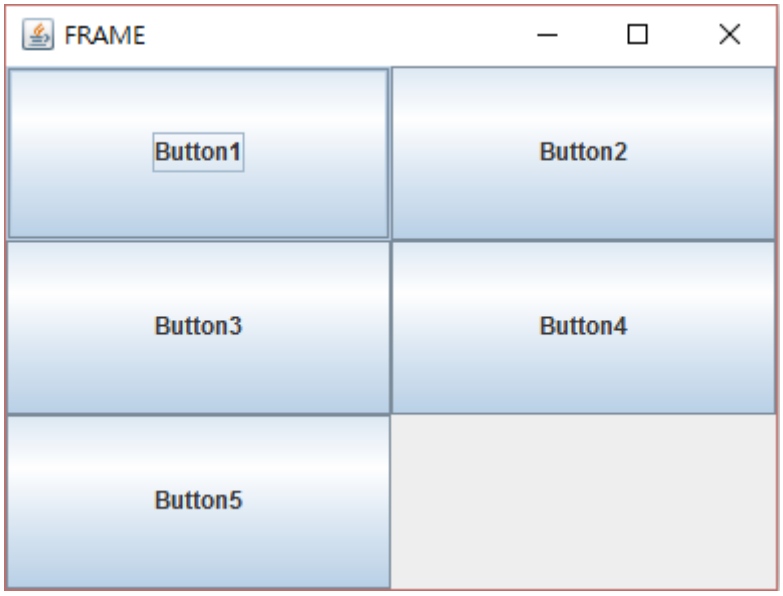
再來就是 FlowLayout，這是最簡單的一種排版方式，它就是照著元件放進去的順序開始由左至右由上而下開始排，並且可以設定每一行的對齊方式。這個方式因為是自動排列的所以不用像 BorderLayout 一樣分別在每個元件裡面寫明要放在哪邊。

```
frame.setSize(400,300);  // set the frame size
frame.setVisible(true);  // show the frame
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new FlowLayout(FlowLayout.CENTER));
```



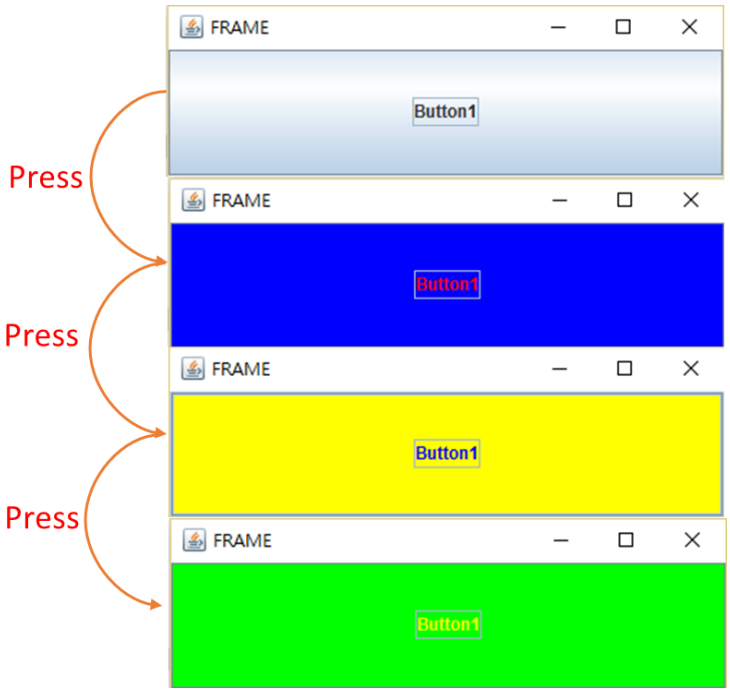
最後我們要介紹的是 GridLayout，它會將一個視窗分成數個網格，因此在一開始設置的時候必須先設定好要分成幾行幾列的網格，而後才放元件進去。

```
frame.setSize(400,300);  // set the frame size
frame.setVisible(true);  // show the frame
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLayout(new GridLayout(3,3));
```



3.4 事件(Event)

圖形介面裡面有很多元件都會產生事件，例如按鈕就是一個最常見的例子，按下該按鈕要做什麼樣的事情就是一種事件的反應。因此我們會在元件裡面建立 event listener，在裡面所寫的方法就會在該事件出現的時候啟動執行。

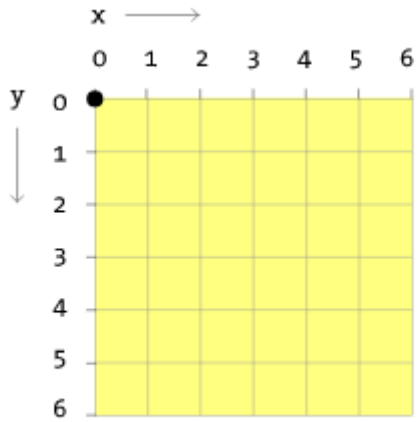


```
 frame.setSize(400,300);  // set the frame size
 frame.setVisible(true);  // show the frame
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setLayout(new GridLayout(3,3));
 //put all the elments on the frame
 frame.getContentPane().add(button1);

 button1.addActionListener(new MyListener()); //add action listener

}

class MyListener implements ActionListener{  // perform the action when the button is pressed
  private Color[] theColors = { Color.RED, Color.BLUE, Color.YELLOW, Color.GREEN };
  private int index = 0;
  public void actionPerformed(ActionEvent e){
  Component theEventer = (Component) e.getSource();
  theEventer.setForeground(theColors[index]);
  index = (index + 1) % theColors.length;
  theEventer.setBackground(theColors[index]);
  }
}
```

值得注意的是，要使用各種 action event 的時候必須匯入 java.awt.event.*的 package 才行。

3.5 Processing 作圖

相比使用 Swing 與 AWT 的方法，Processing 本身就已經提供了一個作圖的環境，而呼叫的方法就是加上一個 draw()的方法。Draw()本身呼叫出來的就是一個類似 frame 的東西，而其中擺放元件的方式是靠著座標來進行的。座標的原點在左上角而 x 與 y 的值分別向右與向下遞增。



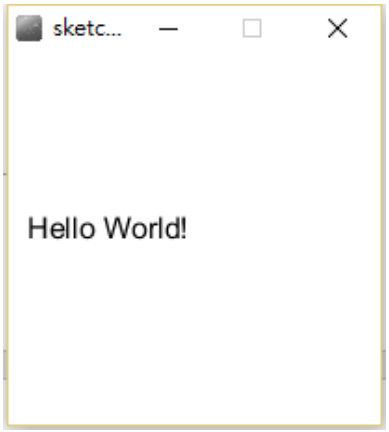
Computer

然而我們首先要先在 setup()裡面做好 frame 的初始設定，因為在 Processing 中，setup()這個 function 只會執行一次，而 draw()則是會一直執行。因此我們要來做一個舉例，如果我們想要在這個畫布中寫上 Hello World，則我們需要先在 setup()裡面先設定好畫布的大小，並且我們可以先加上設定好字體的樣式。

```
PFont f;
void setup() {
  size(200,200); // set frame size
  f = createFont("Arial",16,true); // setup word font
}
```

而後我們才會在 draw()的方法裡面寫上我們想要顯示的東西。

```
void draw() {
  background(255);
  textFont(f,16);                  // Specify font to be used
  fill(0);                         // Specify font color
  text("Hello World!",10,100);    // Display Text
}
```



最後我們使用一個範例來看看如何在 Processing 裡面建立 button 以及使用滑鼠事件。我們拿官網的一個與按鈕有關的範例來做講解，連結如下：https://processing.org/examples/button.html。此程式做出了兩個按鈕並且在按下按鈕之後會改變背景顏色，另外還有在滑鼠滑過去的時候會讓兩個按鈕變顏色。

```
int rectX, rectY;      // Position of square button
int circleX, circleY;  // Position of circle button
int rectSize = 90;     // Diameter of rect
int circleSize = 93;   // Diameter of circle
color rectColor, circleColor, baseColor;
color rectHighlight, circleHighlight;
color currentColor;
boolean rectOver = false;
boolean circleOver = false;
```

```
void setup() {
  size(640, 360); // frame size
  rectColor = color(0);  //rectangle color
  rectHighlight = color(51);
  circleColor = color(255);
  circleHighlight = color(204);
  baseColor = color(102);
  currentColor = baseColor;
  circleX = width/2+circleSize/2+10;
  circleY = height/2;
  rectX = width/2-rectSize-10;
  rectY = height/2-rectSize/2;
  ellipseMode(CENTER);
}
```

首先就是前置的變數宣告作業，這些先設定好我們需要的方形與圓形的按鈕以及顏色。

接下來則是寫了一個 update()的方法，這個方法是 draw()每次開始跑的時候都會先確認現在滑鼠的位置是在圓上或是方形上或是其餘的位置。而這個方法呼叫了 overCircle()與 overRect()兩個 bool 形態的方法來進行判斷。

```
void update(int x, int y) {  // check the mouse position
  if ( overCircle(circleX, circleY, circleSize) ) {
    circleOver = true;
    rectOver = false;
  } else if ( overRect(rectX, rectY, rectSize, rectSize) ) {
    rectOver = true;
    circleOver = false;
  } else {
    circleOver = rectOver = false;
  }
}
```

```
boolean overRect(int x, int y, int width, int height)  {// whether the mouse is in the range of the rectangle
  if (mouseX >= x && mouseX <= x+width &&
      mouseY >= y && mouseY <= y+height) {
    return true;
  } else {
    return false;
  }
}

boolean overCircle(int x, int y, int diameter) {// whether the mouse is in the range of the circle
  float disX = x - mouseX;
  float disY = y - mouseY;
  if (sqrt(sq(disX) + sq(disY)) < diameter/2 ) {
    return true;
  } else {
    return false;
  }
}
```

overRect()與 overCircle()兩個方法會用滑鼠位置的 mouseX, mouseY 變數與兩個圖形的中心做距離的比較，只要計算出距離就可以知道滑鼠的位置是否在圖形上面。

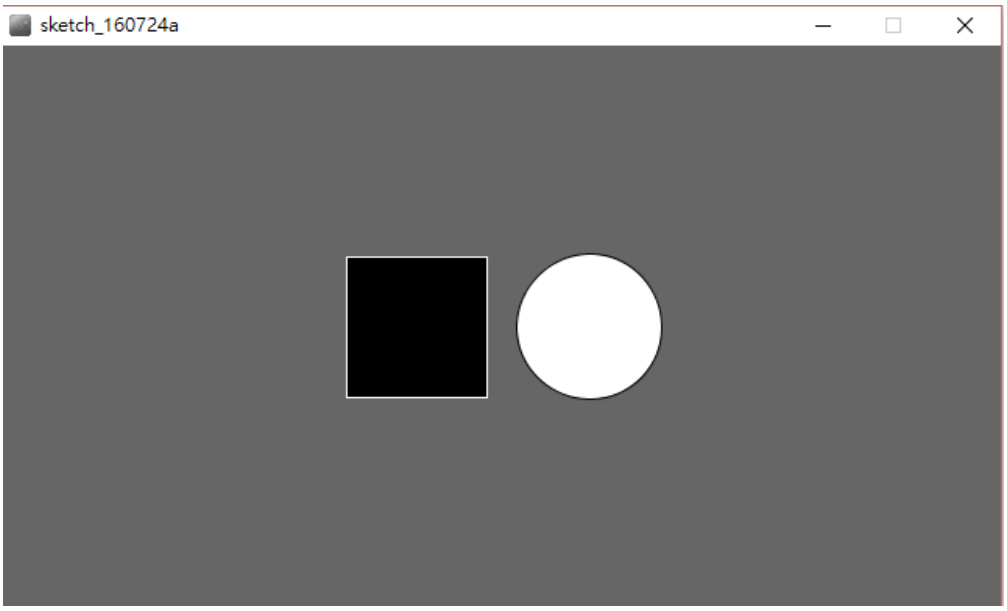
```
void draw() {
  update(mouseX, mouseY);
  background(currentColor);

  if (rectOver) {
    fill(rectHighlight);
  } else {
    fill(rectColor);
  }
  stroke(255);
  rect(rectX, rectY, rectSize, rectSize);

  if (circleOver) {
    fill(circleHighlight);
  } else {
    fill(circleColor);
  }
  stroke(0);
  ellipse(circleX, circleY, circleSize, circleSize);
}
```

在判斷完滑鼠位置之後，緊接著在最主要的 draw()方法裡面就要寫上當滑鼠經過的時候會發生什麼事。在這邊只要當滑鼠的位置在圖形裡面的時候，該圖形就會進行變色，而移開之後又會再度的變回原本的顏色。

最後則是要判斷滑鼠按下時的動作。這邊的作法與前面所敘述的直接增加 Button 之後再加上 actionListener 有一些不同。在 Processing 裡所提供的 mousePressed()方法，原則上是只要滑鼠在整個 frame 的範圍內按下去都會執行該方法，因此我們在裡面加上了 if 的判斷式來確認滑鼠是在什麼地方按下去的。只要在圖形內按下滑鼠就會執行裡面的程式，也就是讓背景換上相應的顏色。

```
void mousePressed() {
  if (circleOver) {
    currentColor = circleColor;
  }
  if (rectOver) {
    currentColor = rectColor;
  }
}
```

完成之後就可以來嘗試執行這個程式，而我們也成功地做出兩個按鈕並且加上了事件的互動反應。



到這邊我們講解了 GUI 的一些基礎做法，但是因為 Processing 本身擁有自己獨樹一格的 function，因此會跟平常 Java 自己支援的所使用的 function 有些許不同，不過大方向上的概念都是相同的。正因為如此，我們並沒有辦法詳細介紹每個 function 的寫法與全部的用法，因此在需要使用到的時候參考並研讀 library 裡的詳細介紹也是一種學習的好方法。

在下一個章節裡，我們會開始介紹一些圖片的應用處理，並且還有視覺化的概念設計。