第1章

绘图基础

本章将带领读者进入 Processing 的世界，学习如何通过编程绘制基本的几何图形，如点、线、矩形、椭圆等。同时，还将介绍颜色的使用和 Processing 的工作流程。本章的目的是为读者奠定视觉编程的基础，帮助读者掌握编程创作的核心工具。学习这些基本绘图功能是后续创意编程的重要起点，因为所有的复杂作品都是由简单的图形和颜色组合而成的。

1.1 认识 Processing

Processing 是一个基于 Java 的开源编程语言和集成开发环境（Integrated Development Environment，IDE），专门为视觉艺术家、设计师以及初学者开发，帮助他们快速上手计算机编程。它最初由 Casey Reas 和 Ben Fry 于 2001 年创立，目的是让人们能够轻松地将编程与图形化创作相结合。Processing 的最大特点是简洁易学，提供了直观的图形输出，用户可以通过几行简单的代码生成复杂的视觉效果。

Processing 在计算机艺术、视觉设计、数据可视化以及交互设计领域得到了广泛应用。它不仅支持静态图形的生成，还可以实现动态效果、动画甚至交互式作品。与传统编程语言不同，Processing 注重创意编程的体验，使得非技术背景的创作者也能够轻松地实现自己的艺术想法。

通过学习 Processing，读者将能够掌握编程的基本概念，并在艺术创作中自由运用这些工具。本书将从最基础的绘图功能开始，逐步深入编程的核心内容，帮助读者掌握编程技术与艺术创作的结合技巧。

1.1.1 安装 Processing

首先，需要下载 Processing 软件，下载网址详见前言二维码。Processing 软件下载界面如图 1-1 所示。单击 Download Processing 4.3 for macOS 按钮，安装即可。

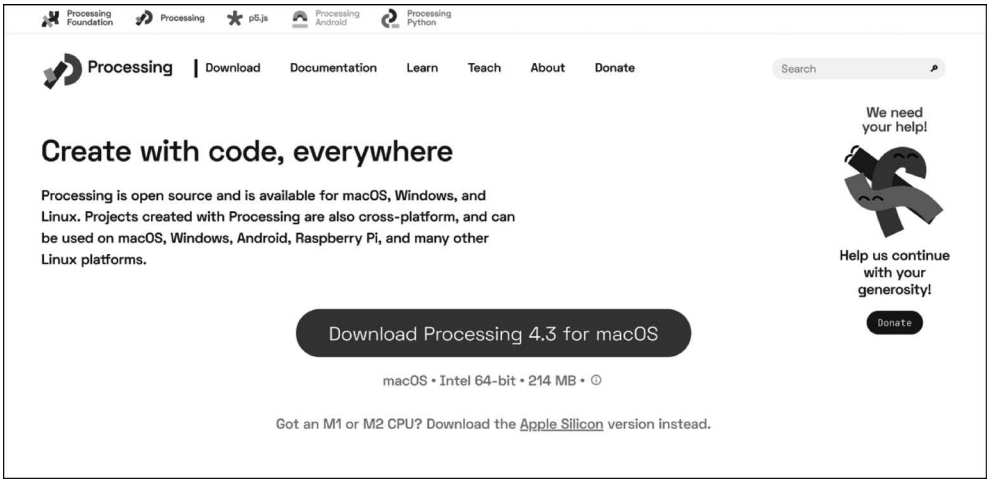


图 1-1　Processing 软件下载界面

双击打开 Processing 工作界面，如图 1-2 所示，可以在这个页面中输入代码。当输入代码后，需要单击左上角的三角形按钮 来运行代码。三角形按钮旁边的 按钮用来结束代码的运行。

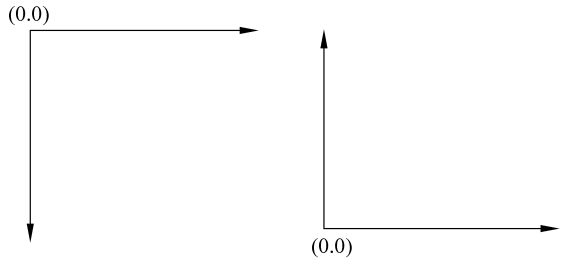
图 1-2　Processing 工作界面

1.1.2　绘制图形

本节将通过一个案例讲解如何用 Processing 来绘制最基本的图形。

如果想用中文来描述一条线，则通常会说：从坐标点(1,2)到坐标点(4,5)画一条线。Processing 的语法也是非常类似的，可以用命令 line(1,2,4,5)来绘制这条线。括号里的数字叫作参数。前两个参数是第一个点的坐标，后两个参数是第二个点的坐标。

在使用 Processing 进行图形绘制时，需要特别注意其坐标系与在中学所学的坐标系有所不同。在中学的坐标系中，X 轴是从左到右增加，Y 轴是从下到上增加。然而，在 Processing 中，虽然 X 轴仍然是从左到右增加，但 Y 轴是从上到下增加。Processing 坐标系如图 1-3 所示。图 1-3(a)为 Processing 定义的坐标系，图 1-3(b)为数学中的平面直角坐标系。



(a) Processing定义的坐标系　　(b) 数学中的平面直角坐标系

图 1-3　Processing 坐标系与数学中的平面直角坐标系对比

理解了 Processing 的坐标系后，就可以开始绘制各种图形了。

首先，使用 point()函数，在坐标(45,45)处绘制一个点。输入如下代码。

```
point(45, 45);
```

使用 rect()命令绘制矩形，默认模式下，需要提供 4 个参数：前两个参数是矩形左上角的坐标，后两个参数是矩形的宽度和高度，例如，输入如下代码。

```
rect(20,30,20,50);
```

绘制的图形如图 1-4 所示，左上角的坐标是(20,30)，宽度是 20，高度是 50。

如果需要，可以改变绘制模式。例如，使用 rectMode(CENTER)可以使前两个参数定义矩形的中心点，后两个参数定义矩形的宽度和高度。以下是代码示例。

```
rectMode(CENTER);
rect(20, 30, 20, 50);
```

绘制的图形如图 1-5 所示，矩形中心点的坐标是(20,30)，宽度是 20，高度是 50。

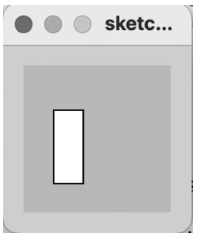


图 1-4　绘制矩形运行结果

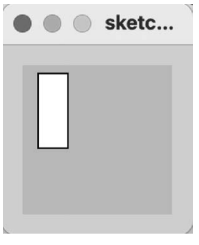


图 1-5　用 CENTER 模式绘制矩形运行结果

那么后面的矩形就都会使用CENTER模式来绘制，直到绘制模式被更改为止。另外，如果更习惯于定义矩形的左上角和右下角的坐标，也可以调整模式为rectMode(CORNERS)。

绘制椭圆的命令是ellipse()。椭圆的默认模式为CENTER，前两个参数是椭圆的中心点，后两个参数是椭圆外接矩形的宽度和高度。代码示例如下。

```
ellipse(50, 50, 100, 50);
```

椭圆的绘制模式也可以通过ellipseMode()来改变，这里不再赘述。

三角形可以用triangle()命令来绘制，如triangle(120,300,232,80,344,300)，这个命令共包含6个参数，第1、2个参数表示三角形第一个点的坐标，第3、4个参数表示第二个点的坐标，第5、6个参数表示第三个点的坐标。

Processing还有很多命令也可以用于绘制其他图形，这里暂时先学习这些。如果读者想要了解更多的图形命令，可以参考Processing官网，网址详见前言中的二维码。

掌握这些基本的图形绘制方法后，就可以进行更复杂的创作了。需要注意的是，练习这些基本图形的目的是熟悉Processing的坐标系和基本功能。Processing功能强大，在实际应用中，不会仅用代码生成简单的基本图形，而是利用这些基础进行更复杂的创作。

在开始第一个创作之前，还需要掌握一个基本命令——size函数，用于设置窗口的大小。例如，size(400,400)将创建一个长和宽均为400px的窗口。需要注意的是，Processing默认的窗口大小是100×100px。

练习1.1：

创建一个400×400px的窗口，在窗口中央绘制一个房子。

读者可以自己设计房子的形状，例如，如图1-6所示的房子包含矩形的主体、三角形的屋顶、圆形的窗户和直线绘制的门。

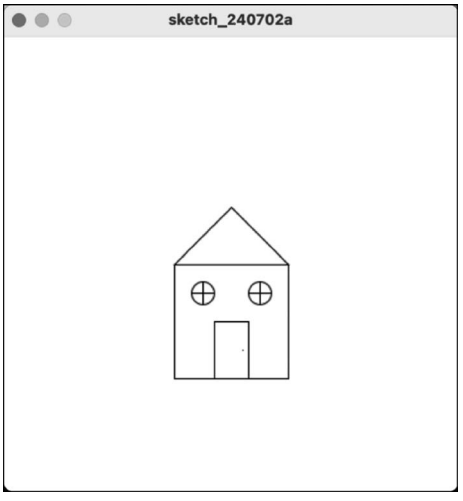


图1-6　练习1.1绘制的房子参考图

1.2　Processing 中的颜色

1.2.1　RGB 颜色模式

在 Processing 中，颜色默认用 RGB 模式表示。RGB 颜色模式由红色(Red)、绿色(Green)和蓝色(Blue)三种颜色组成，每种颜色的取值范围是 0～255。

此外，还有灰度值表示法。如果灰度值为 0，表示纯黑色；如果灰度值为 255，表示纯白色。

在 Processing 中，指定颜色时需要考虑边框颜色(stroke)和填充颜色(fill)。边框颜色使用 stroke 命令，填充颜色使用 fill 命令。

如果 stroke 命令使用三个参数，则分别代表 RGB 值。例如：

```
stroke(255, 0, 0);      //设置边框颜色为红色
stroke(0, 255, 0);      //设置边框颜色为绿色
```

如果使用 4 个参数，最后一个参数则表示透明度(alpha)。例如：

```
stroke(255, 0, 0, 128);      //设置半透明的红色边框
```

类似地，fill 命令的参数表示法与 stroke 命令相同。例如：

```
fill(0, 0, 255, 128);      //设置半透明的蓝色填充
```

对于 stroke 和 fill 命令，如果只使用一个参数，表示灰度值；如果使用两个参数，第一个参数是灰度值，第二个参数是透明度。例如：

```
fill(128);          //灰色填充
fill(128, 128);     //半透明的灰色填充
```

如果想改变窗口背景的颜色，可以使用 background 命令，例如：

```
background(255);      //设置背景颜色为白色
```

如果不需要某个图形的边框或填充颜色，可以使用 noStroke()或 noFill()命令取消边框或填充，运行结果如图 1-7 所示。

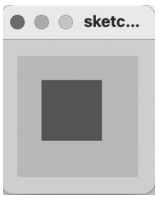


图 1-7　绘制红色填充的矩形

```
noStroke();              //取消边框
fill(255,0,0);
rect(20,20,50,50);       //绘制一个没有边框并且填充为红色的矩形
```

此时的图形没有边框。如果绘制的下一个图形需要边框，再使用 stroke()命令指定边框颜色即可。

在 Processing 中，所有颜色值的范围都是 0～255。如果不确定具体的 RGB 值，可以使用 Processing 的“工具”→“颜色选择器”选项来辅助选择颜色。颜色选择器提供 RGB 值和

HSB 值的参考，如图 1-8 所示。

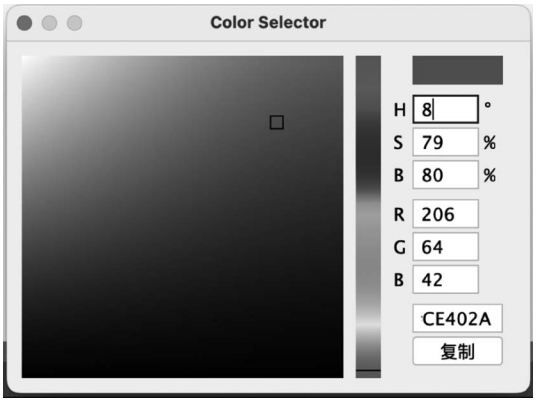


图 1-8　Processing 中的颜色选择器

1.2.2　HSB 颜色模式

HSB 颜色模式是另一种表示颜色的方式，其中，H 代表色相（Hue），S 代表饱和度（Saturation），B 代表亮度（Brightness）。要使用 HSB 模式，可以使用以下命令。

```
colorMode(HSB); //设置颜色模式为 HSB
```

在 HSB 模式下，色相（H）的取值范围是 0～255，对应色轮的 0°～360°。饱和度（S）和亮度（B）的取值范围也是 0～255。

如果要将颜色模式改回 RGB，可以使用：

```
colorMode(RGB); //设置颜色模式为 RGB
```

注意，Processing 中的 HSB 模式与某些美术或设计软件中的 HSB 模式可能略有不同。Processing 将所有值映射到 0～255。

颜色还可以用十六进制数来表示，但是这里不详细讲解十六进制，只需要知道在颜色选择器中，在选定颜色后，右下角会自动生成十六进制数，可以直接单击“复制”按钮，把这个数作为颜色使用，十六进制数的截图如图 1-9 所示。

图 1-9　颜色选择器中的十六进制数

练习 1.2：

将练习 1.1 中绘制的房子上色，为你梦想中的房子涂上颜色吧！

1.3　Processing 工作流程

在使用 Processing 进行动态交互图形创作之前，需要了解 Processing 的基本工作流程。通常，会使用两个主要的函数 setup 和 draw。其中，setup 函数用于初始化设置，内容只会

执行一次；而 draw 函数会在每一帧更新一次。

在 Processing 中，帧（Frame）的概念是指每秒钟更新多少次图像，在默认情况下，Processing 的帧率是每秒 60 帧（即每秒更新 60 次图像）。

可以通过使用 mouseX 和 mouseY 来获取鼠标的位置，并利用这些坐标实现简单的互动效果。

以下案例实现了一个简单的动态交互，代码如例 1-1 所示。

【例 1-1】 鼠标动态交互。

```
void setup() {
  size(400, 400);                    //定义一个 400×400px 的窗口
  background(255);                   //设置背景颜色为白色
  noStroke();                        //取消图形的边框
  fill(255, 0, 0);                   //使用红色填充
}

void draw() {
   background(255);                  //每帧都重置背景颜色
   ellipse(mouseX, mouseY, 40, 40); //绘制一个随鼠标移动的红色椭圆
}
```

运行结果如图 1-10 所示。

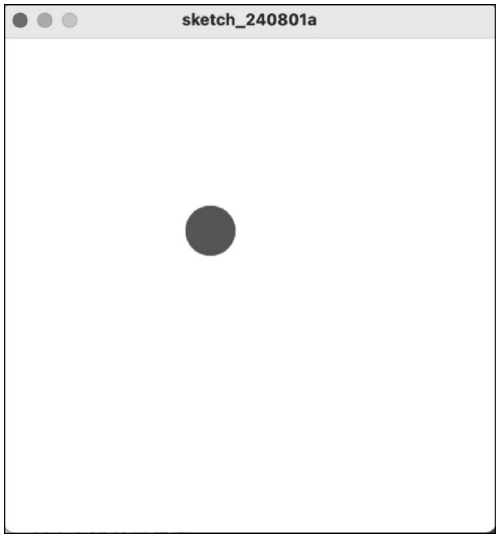


图 1-10　绘制红色小球运行结果

为了更深入地理解 Processing 的工作流程，可以尝试例 1-2 的代码。

【例 1-2】 鼠标绘画效果。

```
void setup() {
  size(400, 400);              //定义一个 400×400px 的窗口
  background(200);             //设置背景的灰度值为 200
}
```

```
void draw() {
//画一条从前一帧鼠标位置到当前帧鼠标位置的线
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

在窗口内随意移动鼠标时，将得到如图 1-11 所示的图案。

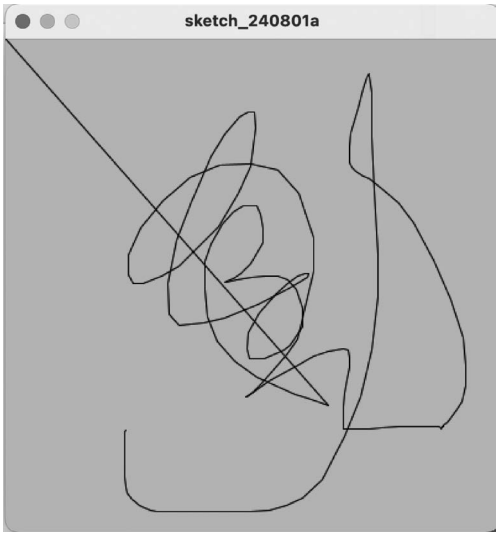


图 1-11　例 1-2 任意移动鼠标时的运行结果

在上述代码中，line(pmouseX，pmouseY，mouseX，mouseY)绘制了上一帧鼠标位置(pmouseX，pmouseY)与当前帧鼠标位置(mouseX，mouseY)之间的线条。这样，每次鼠标移动时，都会在窗口中留下轨迹。

需要注意的是，background 命令的使用位置会影响图形的绘制效果。

如果 background 函数放在 setup 函数中，背景颜色只会设置一次，图形在这张“纸”上不断绘制时，所有的痕迹都会被保留下来。

如果按照如下方式更改上面的代码，把 background 命令放在 draw 函数中，每帧都会重置背景颜色，相当于每帧重新铺一层“纸”，只有当前帧的图形会显示出来，代码如例 1-3 所示。

【例 1-3】　每帧刷新背景。

```
void setup() {
  size(400, 400);
}

void draw() {
  background(200);
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

Processing 提供了 random()函数来生成随机数。例如，random(0，255)会生成一个

0～255 的随机数，第一个参数如果是 0，则可以省略。更改上面的例子，使用 random(0，255)来随机生成背景颜色的红色成分，会得到一个不停变化的背景，因为背景颜色的红色成分每帧更新一次，代码如例 1-4 所示。

【例 1-4】 动态变化背景。

```
void setup(){
  size(400, 400);
}

void draw() {
  background(random(255),128,128);
  line(pmouseX, pmouseY, mouseX, mouseY);
}
```

在 Processing 中，有一些内置的事件函数用于处理用户的交互。一个常见的事件是 mousePressed 事件，它在鼠标被单击时触发。需要注意的是，这些事件必须与 draw 函数一起使用，不能单独存在。

下面是一个使用 mousePressed 事件的例子，代码如例 1-5 所示。

【例 1-5】

```
void setup() {
  size(400, 400);
  background(200);
}
void draw() {
  //每一帧的绘制代码
}
void mousePressed() {
  background(random(255), 128, 128);
}
```

在这个例子中，background(200)设置了初始的背景颜色为灰色(灰度值为 200)。mousePressed 函数定义了一个事件处理程序，当单击鼠标时，背景颜色的红色成分会变为一个随机数，而绿色和蓝色成分保持为 128。这样，每次单击鼠标时，背景颜色都会改变。

通过这种方式，可以控制交互事件的触发频率，使得程序更加符合预期的行为。这样，只有在单击鼠标时，背景颜色才会改变，而不是在每一帧都随机变化。

第2章

编程基础

本章将深入讲解编程的核心概念，包括变量、条件语句、循环以及作用域。这些概念构成了所有编程语言的基础，是编程思维的核心要素。通过学习这些内容，读者将理解如何控制程序的逻辑流程，并通过编程实现自动化操作。掌握这些基本语法结构，读者将能够创建更复杂的交互式和动态效果。

2.1 变量的基础知识

2.1.1 数据类型

在编程中，常见的数据类型有整数类型、浮点类型、布尔类型、字符类型、字符串类型。

(1) 整数类型(int)：用于存储整数，例如，2，1000。

(2) 浮点类型(float)：用于存储小数，例如，3.1415。

(3) 布尔类型(boolean)：用于存储逻辑值，值可以是 true 或 false。例如，布尔变量可以表示灯的开关状态(开为 true，关为 false)。

(4) 字符类型(char)：用于存储单个字符，字符用单引号括起来。例如，'a'或' '(空格)。

(5) 字符串类型(String)：用于存储字符串，字符串用双引号括起来。例如，"Hello, World!"，注意，虽然字符串类型很常见，但是它并不是基础数据类型。

2.1.2 变量的定义与初始化

在编程中，理解变量是非常重要的，Processing 使用的编程语言是 Java。掌握变量的使用不仅对绘制基本图形有帮助，还能为后续更复杂的编程打下坚实的基础。

定义一个变量需要两个步骤。

(1) 声明变量：通过指定数据类型和变量名来声明变量。例如，int x 和 float y 分别声明了一个整数类型的变量 x 和一个浮点数类型的变量 y。

(2) 初始化变量：给变量赋予初始值。例如，x=0 和 y=1.2 将变量 x 初始化为 0，将变量 y 初始化为 1.2。

完整的变量定义与初始化语句可以是：

```
int x = 0;
float y = 0.0;
```

如果没有显式地初始化，整数和浮点数类型的变量默认会被初始化为 0 和 0.0。

在编程中，等号(=)用于赋值操作，即将右边的值存储到左边的变量中。这与数学中的等式不同。例如：

```
int x = 3;      //声明整数类型变量 x,并初始化为 3
x = x + 2;      //为 x 重新赋值为 x+2,即 3+2
```

x=x+2 在数学上是不成立的，但是在这段代码中，第二行的意思是将 x 的当前值(3)加上 2，然后将结果(5)存储到变量 x 中。因此，最终 x 的值为 5。

如果需要比较两个值是否相等，使用双等号(==)。例如：

```
x == 5          //用于判断 x 和 5 是否相等
```

2.1.3　输出变量值

在 Processing 中，可以使用 print 和 println 函数在控制台(Console)中输出变量的值，println 函数会在输出内容后换行，而 print 函数则不会。学会使用控制台输出某些变量的值，对于后面运行复杂程序的调试非常有用。

例如：

```
int x = 3;
println("x 的值是:" + x);
x = x + 2;
println("现在 x 的值是:" + x);
```

程序运行结果如图 2-1 所示，图中下方黑色部分就是 Console。

括号内的内容"x 的值是:" 是一个字符串，字符串可以用加号和其他内容连接起来进行输出。Processing 在控制台中的输出内容如图 2-1 所示。

2.1.4　算术运算符

在编程中，算术运算符是最基本的操作符，用于执行各种数学运算。常见的算术运算符包括加(+)、减(-)、乘(*)、除(/)，以及取余运算符(%)用于获得余数。下面通过一个简单的例子来说明这些运算符的使用，代码如例 2-1 所示。

【例 2-1】 使用算术运算符。

```
void setup() {
  //定义变量
  int a = 10;
```

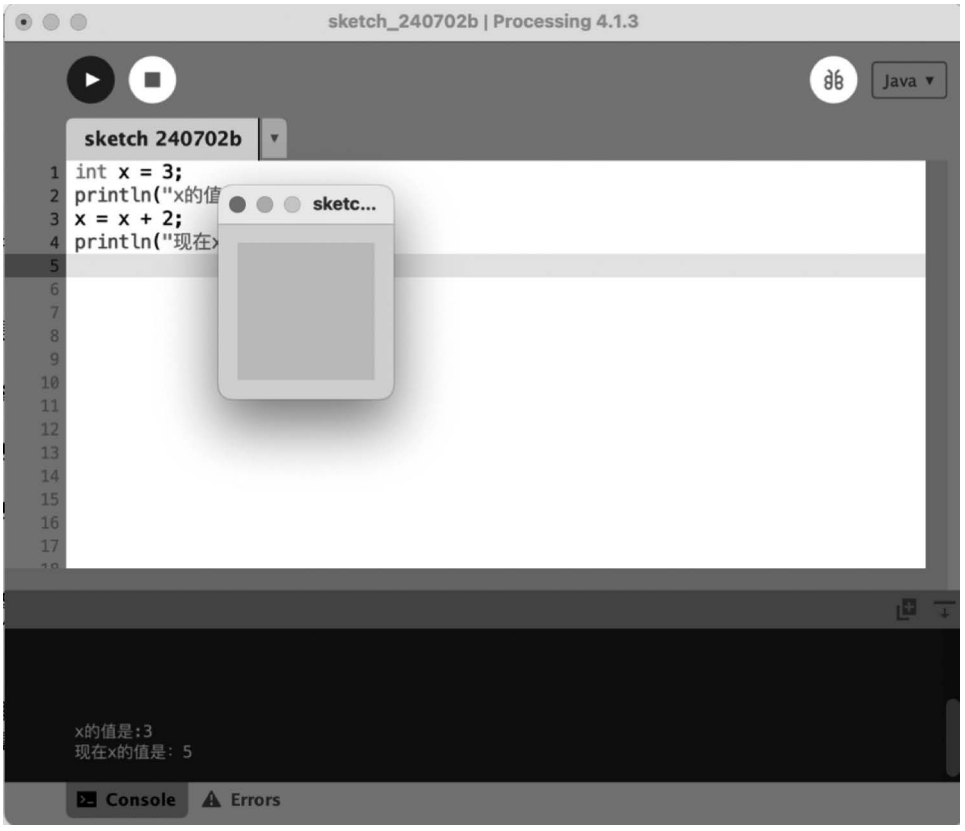


图 2-1　Processing 在控制台中输出内容

```
int b = 3;

//加法
int sum = a + b;
println("加法：" + a + " + " + b + " = " + sum);

//减法
int difference = a - b;
println("减法：" + a + " - " + b + " = " + difference);

//乘法
int product = a * b;
println("乘法：" + a + " * " + b + " = " + product);

//除法
float quotient = (float)a / b; //将 a 转换为浮点数以获得精确结果
println("除法：" + a + " / " + b + " = " + quotient);

//取余
int remainder = a % b;
```

```
  println("取余: " + a + " % " + b + " = " + remainder);
}

void draw() {

}
```

控制台中输出的结果如图 2-2 所示。

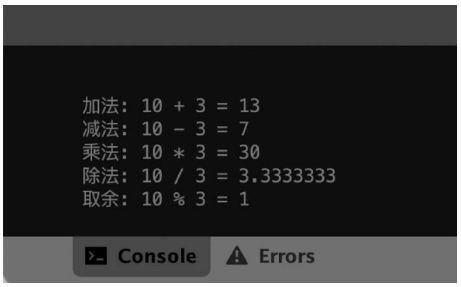


图 2-2　控制台中输出算术运算结果

这里需要注意的是除法的使用，如果 a、b 都是 int 类型，进行除法后的结果还是 int 类型，会取整数部分，得到的结果是 3。如果需要得到精确的结果，可以用 float(a)命令，将 a 强制转换成浮点型，参与运算的两个数中有任意一个浮点型，得到的结果都为浮点型，这样可以得到精确的结果。

通过下面的例子，可以复习所学的内容，代码如例 2-2 所示。这个例子将创建一个 600×600px 的窗口，并在其中绘制随机大小、颜色和位置的矩形。如果不更新背景颜色，将在窗口中累加生成许多矩形，呈现出一种科技感。读者也可以根据自己的设计和想法，在此基础上进行创作。

【例 2-2】　绘制随机矩形。

```
float x = 0;
float y = 0;
void setup(){
  size(600,600);
  background(#0B5998);
  noFill();                          //去除填充色
}
void draw(){
  x = random(0,width);               //x 坐标在 0 到窗口宽度之间
  y = random(0,height);              //y 坐标在 0 到窗口高度之间
  stroke(100,random(0,255),160);     //设置随机颜色的边框
//矩形宽度在 10 到 60 之间,高度在 10 到 80 之间
  rect(x,y, random(10,60),random(10,80));
}
```

运行结果如图 2-3 所示。

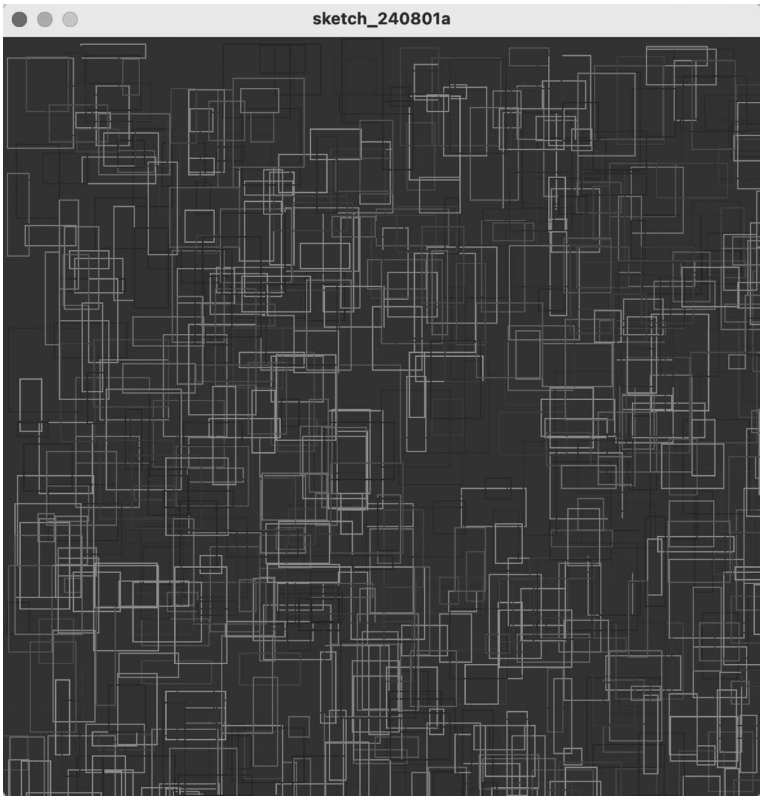


图 2-3　例 2-2 随机生成矩形运行结果

2.2　条件语句

2.2.1　条件语句的语法规则

条件语句是一个非常重要的概念，它能够帮助控制程序的逻辑流。需要了解的第一个逻辑语句的关键字是 if，后面跟随一个条件表达式。当条件表达式为真时，执行 if 语句块中的内容；否则，不执行。

这个例子中先随机生成一个 GPA 的值，范围为 0～4，然后根据 GPA 的值来决定是否录取，代码如例 2-3 所示。

【例 2-3】 使用 if 语句。

```
float GPA = random(0, 4);   //生成一个 0～4 的随机数
println("GPA: " + GPA);     //打印 GPA 值
if (GPA > 2.0) {
  println("恭喜,你被录取了!");
}
println("谢谢");
```

如果 GPA＞2.0，则打印“恭喜，你被录取了！”，无论是否满足 if 里面的条件，后面的“谢

谢”都会被打印。if 语句的逻辑如图 2-4 所示。

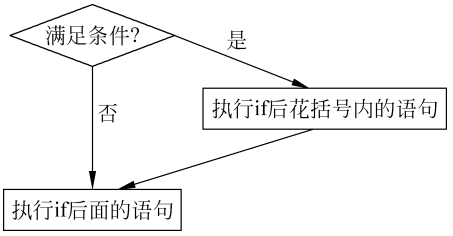


图 2-4　if 语句的逻辑

另外一种逻辑语句是二选一的 if-else 语句。例如，判断 GPA 是否大于或等于 2.0，如果是，则打印“恭喜，你被录取了!”；否则，打印“很遗憾，你没有被录取。”，代码如例 2-4 所示。

【例 2-4】 使用 if-else 语句。

```
float GPA = random(0, 4);        //生成一个 0～4 的随机数
println("GPA: " + GPA);          //打印 GPA 值
if (GPA >= 2.0) {
  println("恭喜,你被录取了!");
} else {
  println("很遗憾,你没有被录取.");
}
```

if-else 语句的逻辑如图 2-5 所示。

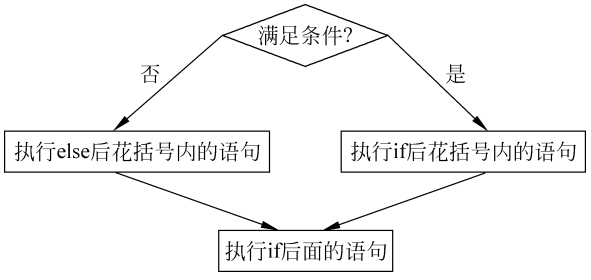


图 2-5　if-else 语句逻辑

现在，对之前的随机矩形程序进行优化，使其在特定条件下绘制不同的形状，代码如例 2-5 所示。

【例 2-5】 使用 if-else 语句绘图。

```
float x = 0;
float y = 0;
void setup(){
  size(600,600);
  background(100,120,160);
  noFill();                          //去除填充色
```

```
}
void draw(){
  x = random(0,width);                    //x 坐标在 0 到窗口宽度之间
  y = random(0,height);                   //y 坐标在 0 到窗口高度之间
  if(x < 300){
    stroke(255,255,0);                    //如果 x 坐标小于 300,边框设置为黄色
  }
  else{
    stroke(100,random(0,255),160);//否则,设置随机颜色的边框
  }
//矩形宽度在 10 到 60 之间,高度在 10 到 80 之间

  rect(x,y, random(10,60),random(10,80));
}
```

得到的结果如图 2-6 所示，左边的图形都为黄色，右边为随机颜色。

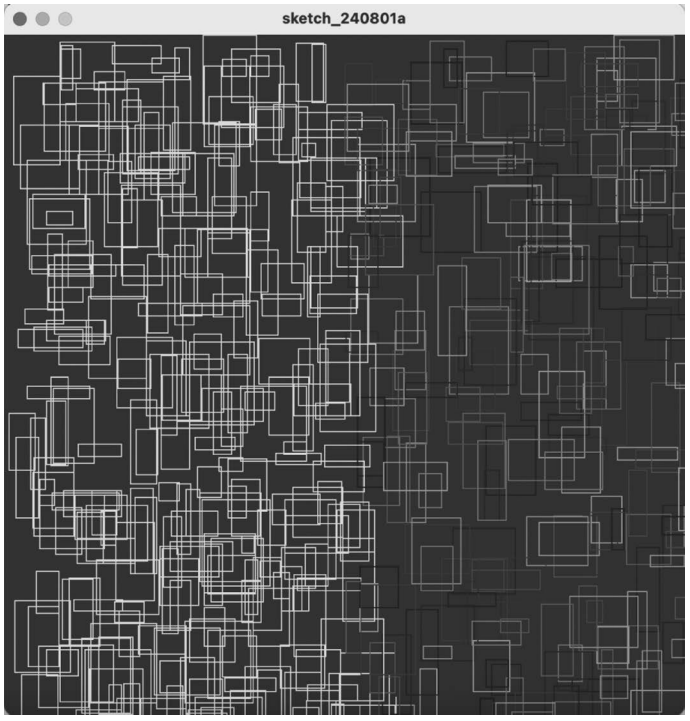


图 2-6　例 2-5 运行结果，左边为黄色矩形，右边为随机颜色

2.2.2　逻辑运算符

在条件判断中，还可以使用逻辑运算符来组合多个条件。

- ==表示等于。
- !=表示不等于。
- <表示小于。

- <=表示小于或等于。
- >表示大于。
- >=表示大于或等于。
- &&表示与(AND)。
- ||表示或(OR)。
- ！表示取反。

&&的意思是,同时满足这个符号前后两个条件。||的意思是,满足前后两个条件中的一个即可。

通过使用这些逻辑运算符,可以实现更加复杂和精确的条件判断,从而更好地控制程序的执行流程。例如,对上面的例子稍做更改,代码如例 2-6 所示。

【例 2-6】 使用逻辑运算符绘图。

```
float x = 0;
float y = 0;
void setup(){
  size(600,600);
  background(100,120,160);
  noFill();                                    //去除填充色
}
void draw(){
  x = random(0,width);                         //x 坐标在 0 到窗口宽度之间
  y = random(0,height);                        //y 坐标在 0 到窗口高度之间
  stroke(100,random(0,255),160);               //设置随机颜色的边框
  if(x < 300&&y < 300){//如果 x < 300 并且 y < 300
      ellipse(x,y,30,30);                      //绘制椭圆
  }
  else{//否则,即在其他部分
    rect(x,y, random(10,60),random(10,80)); //绘制矩形
  }
}
```

运行结果如图 2-7 所示,如果把上面的条件改成 if(x<300||y<300),请读者思考会在哪个部分绘制圆形,在哪个部分绘制矩形?

在编程中,逻辑运算符!表示取反操作。它将表达式的结果取反。例如,如果条件是x<300,在前面加上!变为!(x < 300),即表示 x 不小于 300,也就是 x >=300。

请读者尝试,把上面例子中的条件改为 if(!(x < 300)),将获得怎样的结果呢?

请读者看例 2-7 所示代码,预测会输出怎样的结果。

【例 2-7】 输出成绩等级。

```
int percent = random(50, 100);

if (percent >= 90) {
    println("你得到了 A!");
```

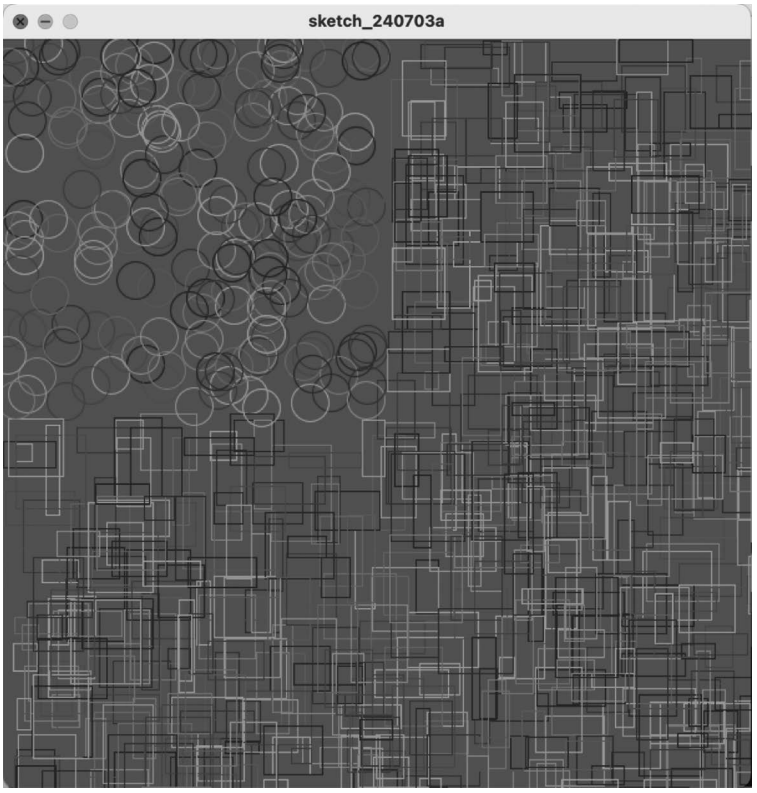


图 2-7　例 2-6 运行结果

```
}
if (percent >= 80) {
    println("你得到了 B!");
}
if (percent >= 70) {
    println(" 你得到了 C!");
}
if (percent >= 60) {
    println("你得到了 D!");
}
if (percent < 60) {
    println("你得到了 F!");
}
```

由于代码是从上到下顺序执行的，这个代码的问题在于，每个 if 语句都会被单独检查，导致当 percent 满足多个条件时，会打印多条信息。例如，如果 percent 是 95，那么它既大于 90，又大于 80，还大于 70 和 60。因此，所有符合条件的 println 语句都会执行，打印出多个成绩。

要解决这个问题，可以使用 else if 语句，这样一旦满足某个条件后，其余的条件就不会被检查了。修正后的代码如例 2-8 所示。

【例 2-8】　正确地输出成绩等级。

```
int percent = random(50,100);
if (percent >= 90) {
    println("You got an A!");
} else if (percent >= 80) { //如果不满足 percent >= 90
println("You got a B!");
//如果不满足 percent >= 90 也不满足 percent >= 80
} else if (percent >= 70) {
    println("You got a C!");
} else if (percent >= 60) {
    println("You got a D!");
} else {
    println("You got an F!");
}
```

下面通过一个例子来复习上面的内容，代码如例 2-9 所示。

【例 2-9】

```
int x;                    //声明变量 x
void setup(){
  size(600,400);
  x = 40;                 //x 初始化为 40
}
void draw(){
  background(255);
  fill(255,0,0);
//在(x,200)这个位置画一个 40×40px 的小球
  ellipse(x,200,40,40);
  x = x + 2;              //x 的值每帧增加 2,视觉上,小球有一种向右移动的效果
}
```

那么，该如何让小球停下呢？可以使用布尔型变量，通过这个变量 true 和 false 的变化来控制，代码如例 2-10 所示。

【例 2-10】　移动的小球。

```
int x;
//定义一个布尔变量 run,表示小球是否运动,初始值为 true
boolean run = true;
void setup(){
  size(600,400);
  x = 40;
}
void draw(){
  background(255);
  fill(255,0,0);
  ellipse(x,200,40,40);
```

```
  if(run == true){ //如果 run 为 true 时
    x = x + 2;
  }
}
```

声明一个布尔型的变量 run，初始化为 true，添加条件，如果 run 为 true 时，让小球移动；否则，不执行 x＝x＋2 操作，即小球不移动。如果把 run 初始化为 false，则小球不移动。

那么，是否可以用鼠标控制小球的移动呢？代码如例 2-11 所示。

【例 2-11】 通过鼠标控制小球的移动。

```
//定义一个整数变量 x,表示小球的水平位置
int x;
//定义一个布尔变量 run,表示小球是否移动,初始值为 true
boolean run = true;
void setup() {
  size(600,400);      //设置画布大小为 600×400px
  x = 40;             //初始化小球的水平位置为 40
}
void draw() {
  background(255);    //设置背景颜色为白色
  fill(255, 0, 0);    //设置填充颜色为红色
  ellipse(x, 200, 40, 40);
  if (run == true) { //如果 run 为 true,小球移动
    x = x + 2;        //小球的水平位置每帧增加 2
  }
}
void mousePressed() {
  if (run == true) { //如果 run 为 true
    run = false;      //将 run 设置为 false,小球停止移动
  } else { //如果 run 为 false
    run = true;       //将 run 设置为 true,小球开始移动
  }
}
```

在 mousePressed()函数中，通过鼠标单击切换 run 的值，如果当前为 true，则重置为 false，如果当前为 false，则重置为 true，实现控制小球的运动或停止。

mousePressed()函数内部，也可以简写成 run＝！run，表示将 run 的值从 true 切换为 false，或从 false 切换为 true。

运行结果如图 2-8 所示。

练习 2.1：

随机产生一个 10～40 的 BMI 指数。如果 BMI 指数小于 18.5，则打印“体重过轻”；如果介于 18.5 和 25，则打印“体重正常”；如果介于 25 和 30，则打印“超重”；如果大于 35，则打印“肥胖”。