Table of Contents

Prefacio

p5.j está inspirado y guiado por otro proyecto, que empezó hace 15 años. En el año 2001, Casey Reas y Ben Fry empezaron a trabajar en una nueva plataforma para hacer más fácil la programación de gráficas interactiva, la nombraron Processing. Ellos estaban frustrados por lo difícil que era escribir este tipo de software con los lenguajes que normalmente usaban (C++ y Java), y fueron inspirados por lo simple que era escribir programas interesantes con los lenguajes que usaban cuando niños (Logo y BASIC). Su mayor influencia fue Design by Numbers (DBN), un lenguaje que ellos estaban trabajando en mantenimiento y enseñando en ese tiempo (y que fue creado por su tutor de investigación, John Maeda). Con Processing, Ben y Casey estaban buscando una mejor manera de probar sus ideas en código, en vez de solo conversarlas o pasar demasiado tiempo programándolas en C++. Su otro objetivo era construir un lenguaje para enseñar cómo programar a estudiantes de diseño y de arte y también brindarles una manera más fácil de trabajar con gráficas a estudiantes más avanzados. Esta combinación es una desviación positiva de la manera en que comúnmente

se enseña programación. Los nuevos usuarios empiezan concentrándose en gráficas e interacción en vez de estructuras de datos y resultados en forma de texto en la consola. A través de los años, Processing se ha transformado en una gran comunidad. Es usado en salas de clases en todo el mundo, en planes de estudios de artes, humanidades y ciencias de la computación, además de profesionales. Hace dos años, Ben y Casey se me acercaron con una pregunta: ¿cómo se vería Processing si funcionara en la web? p5.js empieza con el objetivo original de Processing, hacer que programar sea accesible para artistas, diseñadores, educadores y principiantes, y luego lo reinterpreta para la web actual usando Javascript y HTML. El desarrollo de p5.js ha sido como acercar mundos distintos. Para facilitar la transición a la Web de los usuarios de la existente comunidad de Processing, nos adherimos a la sintaxis y a las convenciones de Processing tanto como fuera posible. Sin embargo, p5.js está construido con Javascript, mientras que Processing está construido con un lenguaje llamado Java. Estos dos lenguajes tienen distintos patrones y funciones, así que a en ocasiones nos tuvimos que desviar de la sintaxis de Processing. También fue importante que p5.js fuera integrado sin problemas a las existentes características, herramientas y marcos de la web, para atraer usuarios familiarizados con la web pero novatos en programación creativa. Sintetizar todos estos factores fue un gran desafío, pero el objetivo de unir estos marcos proporcionó un camino claro a seguir en el desarrollo de p5.js. Una primera versión beta fue lanzada en agosto del 2014. Desde ese entonces, ha sido usado e integrado a programas de estudios en todo el mundo. Existe un editor oficial de p5.js que está actualmente en desarrollo, y ya se ha avanzado en muchas nuevas características y librerías.

p5.js es un esfuerzo comunitario - cientos de personas han contribuido funciones esenciales, soluciones a errores, ejemplos, diseño, reflexiones y discusión. Pretendemos continuar la visión y el espíritu de la comunidad de Processing mientras la abrimos aún más en la Web.

Cómo este libro está organizado

Los capítulos de este libro están organizados de la siguiente manera:

Para quién es este libro

Este libro fue escrito para personas que quieren crear imágenes y programas interactivos simples a través de una casual y concisa introducción a la programación de computadores. Es para personas que quieren una ayuda para entender los miles de ejemplos de código en p5.js y los manuales de referencia disponibles en la web de manera gratuita. Introducción a p5.js no es un libro de referencia sobre programación. Como el título sugiere, te hará una introducción. Es para adolescentes, entusiastas, abuelos, y cualquier persona entremedio. Este libro es apropiado también para personas con experiencia en programación que quieren aprender los conceptos básicos sobre gráficas de computador interactivas. Introducció a p5.js contiene técnicas que pueden ser aplicadas a crear juegos, animaciones e interfaces.

Convenciones usadas en este libro

Las siguientes convenciones tipográficas son usadas en este libro:

Usando los ejemplos de código

El material complementario (ejemplos de código, ejercicios, etc.) está disponble para descarga.

Agradecimientos

Le agradecemos a Brian Jepson y Anna Kaziunas France por su gran energía, apoyo y visión.

No nos imaginamos este libro sin el ejemplo de Introducción a Arduino de Massimo Banzi. Este excelente libro de Massimo es el prototipo.

Capítulo 1. Hola

p5.js sirve para escribir software que produce imágenes, animaciones e interacciones. La intención es escribir una línea de códiga y que un círculo aparezca en la pantalla. Añade unas pocas líneas de código, y ahora el círculo sigue al ratón. Otra línea de código, y el círculo cambia de color cuando presionas el ratón. Le llamamos a esto bosquejar con código. Tú escribes una línea, luego añades otra, luego otra, y así. El resultado es el programa creado una pieza a la vez. Los cursos de programación típicamente se enfocan primero en estructura y teoría. Cualquier aspecto visual - una interfaz, una animación - es considerado un postre que solo puede ser disfrutado después de que terminas de comer tus vegetales, equivalente a varias semanas de estudiar algoritmos y métodos. A través de los años, hemos visto a muchos amigos tratar de tomar estos cursos, para luego abandonarlos después de la primera sesión o después de una muy larga y frustrante noche previa a la entrega de la primera tarea. Toda curiosidad inicial que tenían sobre hacer que el computador trabaje para ellos es perdida porque no pueden ver un camino claro entre lo que tienen que aprender al principio y lo que quieren crear.

p5.js ofrece una manera de programar a través de la creación de gráficas interactivas. Hay muchas maneras posibles de enseñar código, pero los estudiantes usualmente encuentran apoyo y motivación en retroalimentación visual inmediata. p5.js provee esta retroalimentación, y su énfasis en imágenes, prototipado y comunidad es discutido en las siguientes páginas.

Bosquejo y prototipado

Bosquejar es una manera de pensar, es juguetón y rápido. El objetivo básico es explorar muchas ideas en un corto periodo de tiempo. En nuestro propio trabajo, usualmente empezamos bosquejando en papel y luego trasladando nuestros resultados a código. Las ideas para animación e interacción son usualmente bosquejadas como un guión gráfico con anotaciones. Después de hacer algunos bosquejos en software, las mejores ideas son seleccionadas y combinadas en prototipos. Es un proceso cíclico de hacer, probar y mejorar que va y viene entre papel y pantalla.

Flexibilidad

Tal como un cinturón de herramientas para software, p5.js consiste de muchas herramientas que funcionan juntas en diversas combinaciones. Como resultado, puede ser usado para exploraciones rápidas o para investigación en profundidad. Porque un programa hecho con p5.js puede ser tan corto como unas pocas líneas de código o tan largo como miles, existe espacio para crecimiento y variación. Las librerías de p5.js lo extienden a otros dominios incluyendo trabajar con sonido y la adición de botones, barras deslizadoras, cajas de entrada y captura de cámara con HTML.

Gigantes

Las personas han estado haciendo imágenes con computadores desde los años 1960s, y hay mucho que podemos aprender de esta historia. Por ejemplo, antes de que los computadores pudieran proyectar a pantallas CRT o LCd, se usaban grandes máquinas trazadoras para dibujar las imágenes. En la vida, todos nos paramos sobre hombros de gigantes, y los titanes para p5.js incluyen pensadores del diseño, gráfica computacional, arte, arquitectura, estadística y disciplinas intermedias. Dale un vistazo a Sketchpad (1963) por Ivan Sutherland, Dynabook (1968) por Alan Kay y otros artistas destacados en Artist and Computer (Harmony Books, 1976) por Ruth Leavitt. El ACM SIGGRAPH y Ars Electronica brindan atisbos fascinantes en la historia de la gráfica y el software.

Árbol famiiliar

Como los lenguajes humanos, los lenguajes de programación pertenecen a familias de lenguajes relacionados. p5.js es un dialecto de un lenguaje de programación llamado Javascript. La sintaxis del lenguaje es casi idéntica, pero p5.js añade características personalizadas relacionadas a gráficas e interacción y provee un acceso simple a características de HTML5 nativas que ya están soportadas por los navegadores web. Por estas características compartidas, aprender p5.js es un paso útil para aprender a programar en otros lenguajes y usar otras herramientas computacionales.

Únete

Miles de personas usan p5.js cada día. Como ellos, tú puedes descargar p5.js gratuitamente. Incluso tienes la opción de modificar el código de p5.js para que se adapte a tus necesidades. p5.js es un proyecto FLOSS (esto es, free/libre/open source software) y en el espíritu de esta comunidad, te alentamos a participar y compartir tus proyectos y tu conocimiento en línea en http://p5js.org.

Capítulo 2. Empezando a programar

Para sacar el máximo provecho de este libro, no basta con solo leerlo. Necesitas experimentar y practica. No puedes aprender a programar solamente leyendo - debes hacerlo. Para empezar, descarga p5.js y haz tu primer bosquejo.

Ambiente

Primero, necesitarás un editor de código. Un editor de código es similar a un editor de texto (como Bloc de notas), excepto que tiene una funcionalidad especial para editar código en vez de texto plano. Puedes usar cualquier editor que quieras, te recomendamos Atom y Brackets, ambos disponibles para descarga. También existe un editor oficial de p5.js en desarrollo. Si lo quieres usar, lo puedes descargar visitando http://p5js.org/download y seleccionando el botón que dice "Editor". Si estás usando el editor de p5.js, puedes saltar a la sección "Tu primer programa".

Descarga y configuración de archivos

Empieza por visitar http://p5js.org/download y selecciona "p5.js complete". Tras la descarga, haz doble click en el archivo .zip y arrastra el directorio a alguna ubicación en tu disco duro. Puede ser Archivos de programa o Documentos o simplemente tu Escritorio, pero lo importante es que el directorio p5 sea extraído de este archivo .zip. El directorio p5 contiene un proyecto de ejemplo con el que puedes empezar a trabajar. Abre tu editor de código. Luego abre el directorio llamado "empty-example" en tu editor de código. En la mayoría de los editores de código, puedes hacer esto seleccionando en el menú Archivo la opción Abrir, y luego seleccionando "empty-example". ¡Ahora estás listo para empezar tu primer programa!

Tu primer programa

Cuando abras el directorio "empty-example", lo más probable es que veas una barra lateral con el nombre del directorio en la parte superior y una lista con los archivos contenidos en este directorio. Si haces click en alguno de estos archivos, verás los contenidos del archivo aparecer en el área principal.

Un bosquejo en p5.js está compuesto de unos cuántos lenguajes distintos usados en conjunto. HTML (HyperText Markup language) brinda la columna vertebral, enlazando todos los otros elementos en la página. Javascript (y la librería p5.js) te permiten crear gráficas interactivas que puedes mostrar en tu página HTML. A veces CSS (Cascading Style Sheets) es usado para definir elementos de estilo en la página HTML, pero no cubriremos esta materia en este libro.

Si revisas el archivo index.html, te darás cuenta que contiene un poco de código HTML. Este archivo brinda la estructura a tu projecto, uniendo la librería p5.js y otro archivo llamado sketch.js, donde tú escribiras tu propio programa. El código que crea estos enlaces tiene esta apariencia:

```
<script language="javascript" type="text/javascript"
src="../p5.js"></script>
<script language="javascript" type="text/javascript"
src="sketch.js"></script>
```

No necesitas hacer nada en el código HTML en este momento - ya está configurado para ti. Luego, haz click en sketch.js y revisa el código:

```
function setup() {
  // put setup code here
}

function draw() {
  // put drawing code here
}
```

El código plantilla contiene dos bloques, o funciones, setup() y draw(). Puedes poner tu código en cualquiera de los dos lugares, y cada uno tiene un propósito específico.

Cualquier código que esté involucrado en la definición del estado incial de tu programa corresponde al bloque setup(). Por ahora, lo dejaremos vacío, pero más adelante en el libro, añadirás código aquí para especificar el tamaño de tu lienzo para tus gráficas, el peso de tu trazado o la velocidad de tu programa.

Cualquier código involucrado en realmente dibujar contenido a la pantalla (definir el color de fondo, dibujar figuras, texto o imágenes) será colocado en el bloque draw(). Es aquí donde empezarás a escribir tus primeras líneas de código.

Ejemplo 2-1: dibuja una elipse

Entre las llaves del bloque draw(), borra el texto //put drawing code here y reemplázalo con el siguiente:

background(204; ellipse(50, 50, 80, 80);

Tu programa entero deberá verse así:

Esta nueva línea de código significa "dibuja una elipse, con su centro 50 pixeles a la derecha desde el extremo izquierdo y 50 pixeles hacia abajo desde el extremo superior, con una altura y un ancho de 80 pixeles". Graba el código presionando Command-S, o desde el menú con File-Save.

Para ver el código corriendo, puedes abrir index.html en cualquier navegador web (como Chrome, Firefox o Safari). Navega al directorio "empty-example" en tu explorador de archivos y haz doble click en index.html para abrirlo. Otra alternativa es desde el navegador web, escoger Archivo-Abrir y seleccionar el archivo index.html.

Si has escrito todo correctamente, deberías ver un círculo en tu navegador. Si no lo ves, asegúrate de haber copiado correctamente el código de ejemplo. Los números tienen que estar entre paréntesis y tener comas entre ellos. La línea debe terminar con un punto coma.

Una de las cosas más difíciles sobre empezar a programar es que tienes que ser muy específico con la sintaxis. El software p5.js no es siempre suficientemente inteligente como para entender lo que quieres decir, y puede ser muy exigente con la puntuación. Te acostumbrarás a esto con un poco de práctica.

A continuación, avanzaremos para hacer esto un poco más emocionante.

Ejemplo 2-2: hacer círculos

Borra el texto del ejemplo anterior, y prueba este. Graba tu código, y refresca (Command-R) index.html en tu navegador para verlo actualizado.

Este programa crea un lienzo para gráficas que tiene un ancho 480 pixeles y una altura de 120 pixeles, y luego empieza a dibujar círculos blancos en la posición de tu ratón. Cuando presionas un botón del ratón, el color del círculo cambia a negro. Explicaremos después y en detalle más de los elementos de este programa. Por ahora, corre el código, mueve el ratón y haz click para experimentarlo.

La consola

El navegador web tiene incluida una consola que puede ser muy útil para depurar programas. Cada navegador tiene una manera diferente de abrir la consola. Aquí están las instrucciones sobre cómo hacerlo con los navegadores más típicos:

Para abrir la consola en Chrome, desde el menú superior escoge View-Developer-Javascript Console

Con Firefox, desde el menú superior escoge Tools-Web-Developer-Web Console.

Usando Safari, necesitarás habilitar la funcionalidad antes de que puedas usarla. Desde el menú superior, selecciona Preferencias y luego haz click en la pestaña Avanzado y activa la casilla "Show develop menu in menu bar". Tras hacer esto, serás capaz de seleccionar Develop-Show Error Console.

En Internet Explorer, abre F12 Developer Tools, luego selecciona Console Tool.

Deberías ahora ver un recuadro en la parte inferior o lateral de tu pantalla. Si hay un error de digitación, aparecerá texto rojo explicando qué error es. Este texto puede a veces ser críptico, pero si revisas al lado derecho de la línea, estará el nombre del archivo y el número de la línea de código donde fue detectado el error. Ese es un lugar adecuado donde empezar a buscar errores en tu programa.

Creando un nuevo proyecto

Haz creado un bosquejo desde un ejemplo vacío, ¿pero cómo creas un nuevo proyecto? La manera más fácil de hacerlo es ubicando el directorio "empty-example" en tu explorador de archivos y luego copiar y pegarlo para crear un segundo "empty-example". Puedes renombrar la carpeta a lo que quieras - por ejemplo, Proyecto 2.

Ahora puedes abrir este directorio en tu editor de código y empezar a hacer un nuevo bosquejo. Cuando quieras verlo en el navegador, abre el archivo index.html dentro de tu nuevo directorio Proyecto 2.

Siemrpe es una buena idea grabar tus bosquejos frecuentemente. Mientras estás probando cosas nuevas, graba tu bosquejo con diferentes nombres (Archivo-Guardar como), para que así siempre seas capaz de volver a versiones anteriores. Esto es especialmente útil si - o cuando - algo se rompe.

Nota

Un error común es estar editando un proyecto pero estar viendo otro en el navegador web, haciendo que no puedas ver los cambios que has hecho. Si te das cuenta que tu programa se ve igual a pesar de haber hecho cambios a tu código, revisa que estás viendo el archivo index.html correcto.

Ejemplos y referencia

Aprender cómo programar con p5.js involucra explorar mucho código: correr, alterar, romper y mejorarlo hasta que lo hayas reformulado en algo nuevo. Con esto en mente, el sitio web de p5.js tiene docenas de ejemplos que demuestran diferentes características de la librería. Visita la pagína de Ejemplos para verlos. Puedes jugar con ellos editanto el código en la página y luego haciendo click en "Run". Los ejemplos están agrupados en distintas categorías según su función, como Forma, color, e imagen. Encuentra un tema que te interese en la lista y prueba un ejemplo.

Si ves una parte del programa con la que no estás familiarizado o sobre la que quieres aprender su funcionalidad, visita la referencia de p5.js.

La referencia de p5.js explica cada elemento de código con una descripción y ejemplos. Los programas en la Referencia son mucho más cortos (usualmente cuatro o cinco líneas) y más fáciles de seguir que los ejemplos de la página Learn. Ten en cuenta que estos ejemplos

usualmente omiten setup() y draw() por simplicidad, pero estas líneas de código que ves deberán ser puestas dentro de uno de estos bloques para poder se ejecutadas. Recomendamos mantener la página de Referencia abierta mientras estás leyendo este libro y mientras estás programando. Puede ser navegada por tema o usando la barra de búsqueda en la parte superior de la página.

La Referencia fue escrita con el principiante en mente, esperamos que sea clara y entendible. Estamos muy agradecidos de las personas que han visto errores y los han señalado. Si crees que puedes mejorar una entrada en al referencia o que has encontrado algún error, por favor haznos saber esto haciendo click en el link en la parte inferior de la página de referencia.

Capítulo 3. Dibuja

Al principio, dibujar en una pantalla de computador es como trabajar en papel cuadriculado. Parte como un procedimiento técnico cuidadoso, pero a medida que se introducen nuevos conceptos, dibujar formas simples con software se transforma en trabajar con animación e interacción. Antes de que hagamos este salto, tenemos que empezar por el principio.

Una pantalla de computador es una matriz de elementos de luz llamados pixeles. Cada pixel tiene una posición dentro de la matriz definida por coordenadas. Cuando creas un bosquejo en p5.js, lo puedes visualizar con un navegador web. Dentro de la ventana del navegador, p5.js create un lienzo para dibujar, un área en la que se dibujan las gráficas. El lienzo puede ser del mismo tamaño que la ventana, o puede tener dimensiones distintas. El lienzo está usualmente posicionado en la esquina superior izquierda de tu ventana, pero lo puedes posicionar en otros lugares.

Cuando dibujas en el lienzo, la coordenada x es la distancia desde el borde izquierdo del lienzo y la coordenada y es la distancia desde el borde superior. Escribimos las coordenadas de un pixel así (x,y). Así que, si el lienzo es de 200 x 200 pixeles, la esquina superior izquierda es (0,0), el centro está en (100, 100) y la esquina inferior derecha es (199, 199). Estos números pueden parecer confusos, ¿por qué contamos de 0 a 199 en vez de 1 a 200? La respuesta es que en programación, usualmente contamos partiendo en 0 por qué es más fácil así hacer cálculos que veremos más adelante.

El lienzo

El lienzo es creato y las imágenes son dibujadas dentro de él a través de elementos de código llamados funciones. Las funciones son el bloque fundamental de un programa en p5.js. El comportamiento de una función está definido por sus parámetros. Por ejemplo, casi todos los programas en p5.js tienen una función createCanvas() que crea un lienzo para dibujar con un ancho y una altura específicos. Si tu programa no tiene una función createCanvas(), el lienzo creado por defecto tiene dimensiones de 100x100 pixeles.

Ejemplo 3-1: crea un lienzo

La función createCanvas() tiene dos parámetros, el primero define el ancho del lienzo para dibujar, el segundo define la altura. Para dibujar un lienzo que es de 800 pixeles de ancho y 600 pixeles de altura, escribe:

```
function setup() {
  createCanvas(800, 600);
}
```

Corre esta línea de código para ver el resultado. Escribe diferentes valores para explorar las posibilidades. Trata con números muy pequeños y con números más grandes que las dimensiones de tu pantalla.

Ejemplo 3-2: dibuja un punto

Para definir el color de un solo pixel dentro del lienzo, usamos la función point(). Tiene dos parámetros que definen la posición: la coordenada x, seguida de la coordenada y. Para crear un pequeño lienzo y un punto en el centro de él, coordenada (240, 60), escribe:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  point(240, 60);
}
```

Escribe un programa que pone un punto en cada esquina del lienzo para dibujar y uno en el centro. Luego trata de poner puntos consecutivos de manera vertical, horizontal y en líneas diagonales.

Formas básicas

p5.js incluye un grupo de funciones para dibujar formas básicas (ver la figura 3-1). Formas simples, como líneas, pueden ser combinadas para crear formas más complicadas como una hoja o una cara.

Para dibujar solo una línea, necesitamos cuatro parámetros: dos para el punto de inicio y dos para el final.

Ejemplo 3-3: dibuja una línea

Para dibujar una línea entre la coordenada (20, 50) y (420, 110), prueba:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  line(20, 50, 420, 110);
}
```

Ejemplo 3-4: dibuja formas básicas

Siguiendo este patrón, un triángulo necesita seis parámetros y un cuadrilátero necesita ocho (un par para cada punto):

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  quad(158, 55, 199, 14, 392, 66, 351, 107);
  triangle(347, 54, 392, 9, 392, 66);
  triangle(158, 55, 290, 91, 290, 112);
}
```

Ejemplo 3-5: dibuja un rectángulo

Tanto rectángulos como elipses son definidos por cuatro parámetros: el primero y el segundo son las coordenadas x e y del punto ancla, el tercero es el ancho y el cuarto por la altura. Para dibujar un rectángulo (180, 60) con ancho de 220 pixeles y una altura de 40, usa la función rect() así:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  rect(180, 60, 220, 40);
}
```

Ejemplo 3-6: dibuja una elipse

Las coordenadas x e y para un rectángulo son la esquina superior izquierda, pero para una elipse son el centro de la figura. En este ejemplo, date cuenta que la coordenada y para la primera elipse está fuera del lienzo. Los objetos pueden ser dibujados parcialmente (o enteramente) fuera del lienzo sin arrojar errores:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
```

```
  background(204);
  ellipse(278, -100, 400, 400);
  ellipse(120, 100, 110, 110);
  ellipse(412, 60, 18, 18);
}
```

p5.js no tiene funciones distintas para hacer cuadrados y círculos. Para hacer estas figuras, usa el mismo valor para los parámetros de ancho y altura en las funciones ellipse() y rect().

Ejemplo 3-7: dibuja una parte de una elipse

La función arc() dibuja una parte de una elipse:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  arc(90, 60, 80, 80, 0, HALF_PI);
  arc(190, 60, 80, 80, 0, PI + HALF_PI);
  arc(290, 60, 80, 80, PI, TWO_PI+HALF_PI);
  arc(390, 60, 80, 80, QUARTER_PI, PI+QUARTER_PI);
}
```

El primer y segundo parámetro definen la ubicación, mientras que el tercero y el cuarto definen el ancho y la altura. El quinto parámetro define el ángulo de inicio y el sexto el ángulo de parada. Los ángulos están definidos en radianes, en vez de grados. Los radianes son medidas de ángulo basadas en el valor de pi (3.14159). La figura 3-2 muestra cómo ambos están relacionados. Como se ve en este ejemplo, cuatro valores de radianes son usados tan frecuentemente que fueron agregados con nombres especiales como parte de p5.js. Los valores PI, QUARTER_PI, HALF_PI y TWO_PI pueden ser usados para reemplazar los valores en radianes de 180, 45, 90 y 360 grados.

Ejemplo 3-8: dibuja con grados

Si prefieres usar mediciones en grados, puedes convertir a radianes con la función radians(). Esta función toma un ángulo en grados y lo transforma en su correspondiente valor en radianes. El siguiente ejemplo es el mismo que el ejemplo 3-7, pero usa la función radians() para definir en grados los valores de inicio y final:

```
function setup() {
  createCanvas(480, 120);
}
```

```
function draw() {
  background(204);
  arc(90, 60, 80, 80, 0, radians(90));
  arc(190, 60, 80, 80, 0, radians(270));
  arc(290, 60, 80, 80, radians(180), radians(450));
  arc(390, 60, 80, 80, radians(45), radians(225));
}
```

Ejemplo 3-9: usa angleMode

Alternativamente, puedes convertir tu bosquejo para que use grados en vez de radianes usando la función angleMode(). Esta función cambia todas las funciones que aceptan o retornan ángulos para que usen ángulos o radianes, basado en el parámetro de al función, en vez de que tú tengas que convertirlo. El siguiente ejemplo es el mismo que el 3-8, pero usa la función angleMode(DEGREES) para definir los valores en grados de inicio y final:

```
function setup() {
  createCanvas(480, 120);
  angleMode(DEGREES);
}

function draw() {
  background(204);
  arc(90, 60, 80, 80, 0, 90);
  arc(190, 60, 80, 80, 0, 270);
  arc(290, 60, 80, 80, 180, 450);
  arc(390, 60, 80, 80, 45, 225);
}
```

Orden de dibujo

Cuando un programa corre, el computador empieza por el principio y lee cada línea de código hasta que llega a la última línea y luego para.

Nota

Hay unas pocas excepciones a esto, como cuando cargas archivos externos, pero revisaremos esto más adelante. Por ahora, puedes asumir que cada línea correa en orden cuando dibujas.

Si quieres que una figura sea dibujada encima de todas las otras figuras, necesita estar después de las otras en el código.

Ejemplo 3-10: controla el orden tu código

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  ellipse(140, 0, 190, 190);
  // The rectangle draws on top of the ellipse
  // because it comes after in the code
  rect(160, 30, 260, 20);
}
```

Ejemplo 3-11: ponlo en reversa

Modifica el bosquejo invirtiendo el orden de rect() y ellipse() para ver el círculo encima del rectángulo:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  rect(160, 30, 260, 20);
  // The ellipse draws on top of the rectangle
  // because it comes after in the code
  ellipse(140, 0, 190, 190);
}
```

Puedes pensar esto como pintar con brocha o hacer un collage. El último elemento que añades es el que está visible encima.

Propiedades de las figuras

Puedes querer tener más control de las figuras que dibujas, más allá de su posición y su tamaño. Para lograr esto, existe un conjunto de funcioens que definen las propiedades de las figuras.

Ejemplo 3-12: define el grosor del trazado

El valor por defecto del grosor del trazado es de un pixel, pero esto puede ser cambiado con la función strokeWeight(). Un solo parámetro en la función strokeWeight() define el ancho de las líneas dibujadas:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  ellipse(75, 60, 90, 90);
  strokeWeight(8); //stroke weight to 8 pixels
  ellipse(175, 60, 90, 90);
  ellipse(279, 60, 90, 90);
  strokeWeight(20); //stroke weight to 20 pixels
  ellipse(389, 60, 90, 90);
}
```

Ejemplo 3-13: define los atributos del trazado

La función strokeJoin() cambia la forma en que las líneas se unen (y cómo se ven las esquinas), y la función strokeCap() cambia cómo las líneas son dibujadas en su inicio y su final:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  strokeJoin(ROUND); // Round the stroke corners
  rect(40, 25, 70, 70);
  strokeJoin(BEVEL); // Bevel the stroke corners
  rect(140, 25, 70, 70);
  strokeCap(SQUARE); // Square the line endings
  line(270, 25, 340, 95);
  strokeCap(ROUND); // Round the line endings
  line(350, 25, 420, 95);
}
```

La posición de las figuras como rect() y ellipse() son controladas por las funciones rectMode() y ellipseMode(). Revisa la referencia de p5.js para ver ejemplos de cómo posicionar

rectángulos según su centro (en vez de su esquina superior izquierda), o de cómo dibujar elipses desde su esquina superior izquierda como los rectángulos.

Cuando cualquiera de estos atributos es definido, todas las figuras dibujadas posteriormente son afectadas. Como se ve en el ejemplo 3-12, pon atención en cómo el segundo y tercer círculo tienen el mismo grosor de trazado, incluso cuando el grosor es definido solo una vez antes de que ambos sean dibujados.

Fíjate que la línea de código strokeWeight(12 aparece en el bloque de setup() en vez de en draw(). Esto es porque no cambia durante la duración de nuestro programa, así que podemos definirlo sólo una vez durante setup(). Esto es mayoritariamente por organización; poner la línea en draw() tendría el mismo efecto visualizar

Color

Todas las figuras hasta el momento han sido rellenas de color blanco con borde negro. Para cambiar esto, usa las funciones fill() y stroke(). Los valores de los parámetros varían entre 0 y 255, donde 255 es blanco, 128 es gris medio y 0 es negro. En la figura 3-3 se muestra cómo los valores entre 0 y 255 corresponden a diferentes niveles de gris. La función background() que hemos visto en ejemplos anteriores funciona de la misma manera, excepto que en vez de definir el color de relleno o de trazado para dibujar, define el color del fondo del lienzo.

Ejemplo 3-14: pinta con grises

Este ejemplo muestra tres diferentes valores de gris en un fondo negro:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(0);                // Black
  fill(204);                    // Light gray
  ellipse(132, 82, 200, 200);   // Light gray circle
  fill(153);                    // Medium gray
  ellipse(228, -16, 200, 200);  // Medium gray circle
  fill(102);                    // Dark gray
  ellipse(268, 118, 200, 200);  // Dark gray circle
}
```

Ejemplo 3-15: controla el relleno y el color del trazado

Puedes usar la función noStroke() para deshabilitar el trazado para que no se dibuje el borde, y puedes deshabilitar el relleno de una figura con noFill():

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  fill(153);                    // Medium gray
  ellipse(132, 82, 200, 200);   // Gray circle
  noFill();                     // Turn off fill
  ellipse(228, -16, 200, 200);  // Outline circle
  noStroke();                   // Turn off stroke
  ellipse(268, 118, 200, 200);  // Doesn't draw
}
```

Ten cuidado de no deshabilitar el relleno y el trazado al mismo tiempo, como lo hicimos en el ejemplo anterior, porque nada será dibujada en la pantalla.

Ejemplo 3-16: dibuja con color

Para ir más allá de la escala de grises, usa tres parámetros para espeficar los componentes de color rojo, verde y azul. Como está libro está impreso en blanco y negro, sólo verás valores grises aquí. Corre el código para revelar los colores:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(0, 25, 51);        // Dark blue color
  fill(255, 0, 0);              // Red color
  ellipse(132, 82, 200, 200);   // Red circle
  fill(0, 255, 0);              // Green color
  ellipse(228, -16, 200, 200);  // Green circle
  fill(0, 0, 255);              // Blue color
  ellipse(268, 118, 200, 200);  // Blue circle
}
```

Los colores en el ejemplo son referidos como color RGB, porque es cómo los computadores definen el color en la pantalla. Los tres números definen los valores de rojo, verde y azul, y varían entre 0 y 255 de la misma forma que los valores de gris. Estos tres números son los parámetros para tus funciones de background(), fill() y stroke().

Ejemplo 3-17: define la transparencia

Al añadir un cuarto parámetro a fill() o a stroke(), puedes controlar la transparencia. Este cuarto parámetro es conocido como el valor alpha, y también varía entre 0 y 255 para definir el monto de transparencia. El valor 0 define el color como totalmente transparente (no será mostrado en la pantalla), el valor 255 es enteramente ópaco, y los valores entre estos extremos hacen que los colores se mezclen en la pantalla:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204, 226, 225);    // Light blue color
  fill(255, 0, 0, 160);         // Red color
  ellipse(132, 82, 200, 200);   // Red circle
  fill(0, 255, 0, 160);         // Green color
  ellipse(228, -16, 200, 200);  // Green circle
  fill(0, 0, 255, 160);         // Blue color
  ellipse(268, 118, 200, 200);  // Blue circle
}
```

Formas personalizadas

No estás limitado a usar estas formas geométricas básicas - puedes dibujar nuevas formas conectando una serie de puntos.

Ejemplo 3-18: dibuja una flecha

La función beginShape() señala el comienzo de una nueva figura. La función vertex() es usada para definir cada par de coordenadas (x,y) de la figura. Finalmente, endShape() señala que la figura está completa:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
```

```
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape();
}
```

Ejemplo 3-19: cierra la brecha

Cuando corres el ejemplo 3-18, verás que el primer y el último punto no están conectados. Para hacer esto, añade la palabra CLOSE como parámetro a la función endShape, así:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape(CLOSE);
}
```

Ejemplo 3-20: crea algunas criaturas

El poder de definir figuras con vertex() es la habilidad de hacer figuras con bordes complejos. p5.js puede dibujar miles y miles de líneas al mismo tiempo para llenar la pantalla con figuras fantásticas que emanan de tu imaginación. Un ejemplo modesto pero más complejo es presentado a continuación:

```
function setup() {
  createCanvas(480, 120);
}
```

```
function draw() {
  background(204);

  // Left creature
  beginShape();
  vertex(50, 120);
  vertex(100, 90);
  vertex(110, 60);
  vertex(80, 20);
  vertex(210, 60);
  vertex(160, 80);
  vertex(200, 90);
  vertex(140, 100);
  vertex(130, 120);
  endShape();
  fill(0);
  ellipse(155, 60, 8, 8);

  // Right creature
  fill(255);
  beginShape();
  vertex(370, 120);
  vertex(360, 90);
  vertex(290, 80);
  vertex(340, 70);
  vertex(280, 50);
  vertex(420, 10);
  vertex(390, 50);
  vertex(410, 90);
  vertex(460, 120);
  endShape();
  fill(0);
  ellipse(345, 50, 10, 10);
}
```

Comentarios

Los ejemplos en este capítulo usan doble barra (//) al final de una línea para añadir comentarios al código. Los comentarios son una parte de los programas que son ignorados cuando el programa corre. Son útiles para hacer notas para ti mismo que expliquen lo que está pasando en el código. Si otras personas están leyendo tu código, los comentarios son especialmente importantes para ayudarles a entender tu proceso.

Los comentarios son también especialmente útiles para un número de diferentes opciones, como tratar de escoger el color correcto. Así que, por ejemplo, podríamos estar tratando de encontrar el rojo preciso que queremos para una elipse:

```
function setup() {
  createCanvas(200, 200);
}

function draw() {
  background(204);
  fill(165, 57, 57);
  ellipse(100, 100, 80, 80);
}
```

Ahora supón que quieres probar un rojo distinto, pero no quieres perder el antiguo. Puedo copiar y pegar la línea, hacer un cambio y luego comentar la línea de código antigua:

```
function setup() {
  createCanvas(200, 200);
}

function draw() {
  background(204);
  //fill(165, 57, 57);
  fill(144, 39, 39);
  ellipse(100, 100, 80, 80);
}
```

Poner // al principio de una línea temporalmente la anula. O puedo remover // y escribirlo al inicio de otra línea si quiero probarlo de nuevo:

```
function setup() {
  createCanvas(200, 200);
}

function draw() {
  background(204);
  fill(165, 57, 57);
  //(144, 39, 39);
  ellipse(100, 100, 80, 80);
}
```

Mientras trabajas con bosquejos de p5.js, te encontrarás a ti mismo creando docenas de iteraciones de ideas; usar comentarios para hacer notas o para deshabilitar líneas de códigos puede ayudarte a mantener registro de tus múltiples opciones.

Robot 1: dibuja

Ella es P5, la robot de p5.js. Hay 10 diferentes programas para dibujarla y animarla en este libro - cada uno explora una idea de programación diferente. El diseño de P5 está inspirado en

Sputnik I (1957), Shakey del Stanford Research Institute (1966 - 1972), el dron luchador en la película Dune (1984) de David Lynch y HAL 9000 de 2001: Una odisea en el espacio (1968), entre otros robots favoritos.

El primer programa de robot usa las funciones de dibujo introducidas anteriormente en este capítulo. Los parámetros de las funciones fill() y stroke() definen los valores de la escala de grises. Las funciones line(), ellipse() y rect() definen las formas que crean el cuello, las antenas, el cuerpo y la cabeza de la robot. Para familiarizarze mejor con las funciones, corre el programa y cambia los valores para rediseñar el robot:

```
function setup() {
  createCanvas(720, 480);
  strokeWeight(2);
  ellipseMode(RADIUS);
}

function draw() {
  background(204);

  // Neck
  stroke(102);                // Set stroke to gray
  line(266, 257, 266, 162);   // Left
  line(276, 257, 276, 162);   // Middle
  line(286, 257, 286, 162);   // Right

  // Antennae
  line(276, 155, 246, 112);   // Small
  line(276, 155, 306,  56);   // Tall
  line(276, 155, 342, 170);   // Medium

  // Body
  noStroke();                 // Disable stroke
  fill(102);                  // Set fill to gray
  ellipse(264, 377, 33, 33);  // Antigravity orb
  fill(0);                    // Set fill to black
  rect(219, 257, 90, 120);    // Main body
  fill(102);                  // Set fill to gray
  rect(219, 274, 90, 6);      // Gray stripe

  // Head
  fill(0);                    // Set fill to black
  ellipse(276, 155, 45, 45);  // Head
  fill(255);                  // Set fill to white
  ellipse(288, 150, 14, 14);  // Large eye
  fill(0);                    // Set fill to black
  ellipse(288, 150, 3, 3);    // Pupil
  fill(153);                  // Set fill to light gray
```

```
  ellipse(263, 148, 5, 5);    // Small eye 1
  ellipse(296, 130, 4, 4);    // Small eye 2
  ellipse(305, 162, 3, 3);    // Small eye 3
}
```

Capítulo 4. Variables

Una variable guarda un valor en memoria para que pueda ser usado posteriormente en un programa. Una variable puede ser usada muchas veces dentro del mismo programa, y el valor puede ser fácilmente modificado mientras el programa está corriendo.

Primeras variables

La razón principal por la que usamos variables es para evitar repetirnos en el código. Si estás escribiendo el mismo número una y otra vez, considera usar una variable para que tu código sea más general y más fácil de actualizar.

Ejemplo 4-1: reusa los mismos valores

Por ejemplo, cuando haces variables la coordenada y y el diámetro para los tres círculos en este ejemplo, los mismos valores son usados para cada elipse:

```
var y = 60;
var d = 80;

function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  //izquierda
  ellipse(75, y, d, d);
  //centro
  ellipse(175, y, d, d);
  //derecha
  ellipse(275, y, d, d);
}
```

Ejemplo 4-2: cambiar los valores

Simplemente cambiar las variables y y d entonces altera las tres elipses:

```
var y = 100;
var d = 130;

function setup() {
```

```
  createCanvas(480, 120);
}

function draw() {
  background(204);
  //izquierda
  ellipse(75, y, d, d);
  //centro
  ellipse(175, y, d, d);
  //derecha
  ellipse(275, y, d, d);
}
```

Sin las variables, necesitarías cambiar la coordenada y usada en el código tres veces y la del diámetro seis veces. Cuando comparas los ejemplos 4-1 y 4-2, revisa cómo todas las líneas son iguales, excepto las dos primeras líneas con variables que son diferentes. Las variables te permiten separar las líneas de código que cambian de las que no cambian, lo que hace que los programas sean fáciles de modificar. Por ejemplo, si pones las variables que controlan colores y tamaños en un lugar, entonces puedes explorar diferentes opciones visuales enfocándote en sólo unas pocas líneas de código.

Haciendo variables

Cuando haces tus propias variables, puedes determinar el nombre y el valor. Tú decides cómo se llama la variable. Escoge un nombre que sea informativo sobre lo que está almacenado en la variable, pero que sea consistente y no muy largo. Por ejemplo, el nombre de variable "radio" es mucho más claro que "r" cuando lo lees posteriormente en tu código.

Las variables primero deben ser declaradas, lo que reserva espacio en la memoria del computador para guardar la información. Cuando declaras una variable, usas la palabra var, para indicar que estás creando una nueva variable, seguida del nombre. Después de que el nombre es fijado, un valor puede ser asignado a la variable:

```
var x;  // Declara la variable x
x = 12; // Asigna un valor a x
```

Este código hace lo mismo, pero es más corto:

```
var x = 12;  // Declara la variable x y le asigna un valor
```

Los caracteres var son incluidos en la línea de código que declara la variable, pero no son escritos de nuevo. Cada vez que var es escrito antes que el nombre de una variable, el computador piensa que estás tratando de declarara una nueva variable. No puedes tener dos variables con el mismo nombre en la misma sección del programa (Apéndice C), o el programa podría comportarse extrañamente:

```
var x;      // Declara la variable x
var x = 12; // ERROR! No pueden haber dos variables x
```

Puedes situar tus variables afuera de setup() y draw(). Si creas una variable dentro de setup(), no puedes usarla dentro de draw(), así que necesitas situar estas variables en otro lugar. Estas variables reciben el nombre de variables globales, porque pueden ser usadas en cualquier lugar ("globalmente") del programa.

Variables de p5.js

p5.js tiene una serie de variables especiales para almacenar información sobre el programa mientras corre. Por ejemplo, el ancho y la altura del lienzo están almacenados en las variables width y height. Estos valores son definidos por la función createCanvas(). Pueden ser usados para dibujar elementos relativos al tamaño del lienzo, incluso si la línea de código de createCanvas() es alterada.

Ejemplo 4-3: ajusta el lienzo, observa lo que sucede

En este ejemplo, cambia los parámetros de createCanvas() para observar cómo funciona:

```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  line(0, 0, width, height); // Línea desde (0,0) a (480, 120)
  line(width, 0, 0, height); // Línea desde (480,0) a (0, 120)
  ellipse(width/2, height/2, 60, 60);
}
```

Existen también variables epsciales que mantienen registro del estado del ratón y de los valores del teclado, entre otras. Serán discutidas en el Capítulo 5.

Un poco de matemáticas

La gente a menudo asume que las matemáticas y la programación son lo mismo. Aunque un poco de conocimiento de matemáticas puede ser útil para ciertos tipos de programación, la aritmética básica cubre las partes más importantes.

Ejemplo 4-4: aritmética básica

```
var x = 25;
var h = 20;
var y = 25;

function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  x = 20;
  rect(x,y,300,h);          // Superior
  x = x + 100;
  rect(x, y + h, 300, h);   // Centro
  x = x - 250;
  rect(x, y + h*2, 300, h); // Inferior
}
```

En el código, símbolos como +, - y * son llamados operadores. Cuando se encuentran entre dos valores, crean una expresión. Por ejemplo, 5 + 9 y 1024 - 512 son expresiones. Los operadores para operaciones matemáticas básicas son:

Javascript tiene un conjunto de reglas para definir el orden de precedencia que los operadores tienen entre sí, lo que significa, cuáles cálculos son efectuados en primer, segundo y tercer lugar, etc. Estas reglas definen el orden en el que el código se ejecuta. Un poco de conocimiento sobre esto es un gran paso hacia el entendimiendo de cómo funciona una corta línea de código como esta:

```
var x = 4 + 4 * 5;  // Se le asigna el valor 24 a x
```

La expresión 4*5 es evaluada primero porque la multiplicación tiene la prioridad más alta. Luego, se le suma 4 al producto 4*5, resultando 24. Finalmente, como el operador de asignación (el signo igual) tiene la menor predecencia, el valor 24 es asignado a la variable x. Esto se puede aclarar con el uso de paréntesis, pero el resultado es el mismo:

```
var x = 4 + (4 * 5);  // Se le asigna el valor 24 a x
```

Si quieres forzar que la suma ocurra primero, usa paréntesis. Como los paréntesis tienen mayor precedencia que la multiplicación, al cambiar los paréntesis de lugar se cambia el cálculo efectuado:

```
var x = 4 + (4 * 5);  // Se le asigna el valor 24 a x
```

Un acrónimo para este orden se enseña en clases de matemáticas: PEMDAS, que significa paréntesis, exponentes, multiplicación, división, adición, substracción, donde los paréntesis tienen la mayor prioridad y la substracción la menor. El orden completo de operaciones se encuentra anotado en el Apéndice B.

Algunos cálculos son usados tan frecuentemente en programación que se han desarrollado atajos, es útil ahorrar tiempo en el teclado. Por ejemplo, cuando puedes sumar o restar a una variable con un operador:

```
x += 10;  // Es equivalente a x = x + 10;
x -= 15;  // Es equivalente a x = x - 15;
```

También es muy común sumar o restar 1 a una variable, así que esto también tiene un atajo. Los operadores ++ y -- hacen esto:

```
x ++;  // Es equivalente a x = x + 1;
x --;  // Es equivalente a x = x - 1;
```

Repetición

Mientras escribes programas, te darás cuenta que ocurren patrones al repetir líneas de código con pequeñas modificaciones. Una estructura de código llamada "for loop" hace posible que una línea de código corre más de una vez para condensar el tipo de repetición a unas pocas líneas de código. Esto hace que tus programas sean modulares y más simples de modificar.

Ejemplo 4-5: haz lo mismo una y otra vez

Este ejemplo tiene el tipo de patrón que puede ser simplificado con un "for loop":

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(8);
}

function draw() {
  background(204);
  line( 20, 40,  80, 80);
  line( 80, 40, 140, 80);
  line(140, 40, 200, 80);
  line(200, 40, 260, 80);
  line(260, 40, 320, 80);
  line(320, 40, 380, 80);
  line(380, 40, 440, 80);
}
```

Ejemplo 4-6: usa un for loop

Lo mismo puede ser logrado con un for loop, y con mucho menos código:

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(8);
}

function draw() {
  background(204);
  for (var i = 20; i < 400; i += 60) {
    line( i, 40,  i + 60, 80);
  }
}
```

El for loop es diferente en muchas maneras del código que hemos escrito hasta ahora. Fíjate en las llaves, los caracteres { y }. El código repetido entre las llaves es llamado bloque. Este es el código que será repetido en cada iteración del for loop.

Adentro del paréntesis hay tres declaraciones, separadas por punto y coma, que funcionan en conjunto para controlar cuántas veces el código dentro del bloque es ejecutado. De izquierda a derecha, estas declaraciones son nombradas así: inicialización (init), prueba (test), actualización (update):

```
for (init; test; update) {
  declaraciones
}
```

Init típicamente declara una variable nueva a ser usada en el for loop y le asigna un valor. El nombre de variable i es frecuentemente usado, pero esto no tiene nada de especial. El test evalúa el valor de esta variable, y update change el valor de la variable. La figura 4-1 muestra el orden en el que el código es ejectuado y cómo controlan el código dentro del bloque.

La prueba o test requiere más explicación. Siempre es una expresión de relación que compara dos valores con un operador relacional. En este ejemplo, la expresión es "i < 400" y el operador es el símbolo < (menor que). Los operadores relacionales más comunes son:

La expresión relacional siempre evalua a verdadero (true) o falso (false). Por ejemplo, la expresión 5 > 3 es true. Podemos preguntar, "¿es cinco mayor que tres?". Como la respuesta es "sí", decimos que la expresión es true. Para la expresión 5 < 3, podemos preguntar, "¿es cinco menor que tres?". Como la respuesta es no, decimos que la expresión es false. Cuando la evaluación es true, el código dentro del bloque se ejecuta y cuando es false, el código dentro del bloque no se ejecuta y el for loop se acaba.

Ejemplo 4-7: entrena tus músculos para hacer for loops

El poder definitivo que entregan los for loop es la habilidad para hacer cambios rápidos a tu código. Como el código dentro del bloque es ejecutado típicamente múltiples veces, un cambio al bloque es magnificado cuando el código es ejecutado. Al modificar el ejemplo 4-6 un poco, podemos crear una variedad de distintos patrones:

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(2);
}

function draw() {
  background(204);
  for (var i = 20; i < 400; i += 8) {
    line( i, 40,  i + 60, 80);
  }
}
```

Ejemplo 4-8: desplegando las líneas

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(2);
}

function draw() {
  background(204);
  for (var i = 20; i < 400; i += 20) {
    line( i, 0,  i + i/2, 80);
  }
}
```

Ejemplo 4-9: modificando las líneas

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(2);
}

function draw() {
  background(204);
```

```
  for (var i = 20; i < 400; i += 20) {
    line( i, 0,  i + i/2, 80);
    line( i + i/2,  80, i * 1.2, 80);
  }
}
```

Ejemplo 4-10: anidando un for loop dentro de otro

Cuando un for loop es anidado dentro de otro, el número de repeticiones se multiplica. Primero veamos un ejemplo corto y luego lo veremos por partes en el ejemplo 4-11.

```
function setup() {
  createCanvas(480, 120);
  noStroke();
}

function draw() {
  background(0);
  for (var y = 0; y <= height; y += 40) {
    for (var x = 0; y <= width; y += 40) {
      fill(255, 140);
      ellipse(x, y, 40, 40);
    }
  }
}
```

Ejemplo 4-11: filas y columnas

En este ejemplo, los for loops están adyacentes, en vez de estar uno dentro de otro. El resultado muestra que un for loop está dibujando una columna de 4 círculos y el otro está dibujando una fila de 13 círculos.

```
function setup() {
  createCanvas(480, 120);
  noStroke();
}

function draw() {
  background(0);
  for (var y = 0; y < height + 45; y += 40) {
    fill(255, 140);
    ellipse(0, y, 40, 40);
  }
```

```
  for (var x = 0; x <= width + 45; x += 40) {
    fill(255, 140);
    ellipse(x, 0, 40, 40);
  }
}
```

Cuando uno de estos for loop es puesto dentro del otro, como en el ejemplo 4-10, las 4 repeticiones del primer loop son compuestas con las 13 del segundo, para así ejecutar el código dentro del bloque compuesta 52 veces (4 x 13 = 52).

El ejemplo 4-10 es una buena base para explorar muchos tipos de patrones visuales repetitivos. Los siguientes ejemplos muestran un par de maneras en que esto puede ser extendido, pero esto es solo una pequeña muestra de lo que es posible.

Ejemplo 4-12: alfileres y líneas

En este ejemplo, el código dibuja una línea desde cada punto de la matriz hasta el centro de la pantalla:

```
function setup() {
  createCanvas(480, 120);
  fill(255);
  stroke(102);
}

function draw() {
  background(0);
  for (var y = 20; y < height - 20; y += 10) {
    for (var x = 20; x <= width - 20; x += 10) {
      ellipse(x, y, 4, 4);
      // Dibuja una línea al centro de la imagen
      line(x, y, 240, 60);
    }
  }
}
```

Ejemplo 4-13: Puntos semitono

En este ejemplo, las elipses se reducen en tamaño con cada nueva fila y son movidas hacia la derecha, por medio de añadir la coordenada y a la coordenada x:

```
function setup() {
  createCanvas(480, 120);
}
```

```
function draw() {
  background(0);
  for (var y = 32; y < height; y += 8) {
    for (var x = 12; x <= width; x += 15) {
      ellipse(x + y, y, 16 - y/10.0, 16 - y/10.0);
    }
  }
}
```

Robot 2: variables

Las variables introducidas en este programa hacen que el código se vea más difícil que el de la Robot 1 (ver "Robot 1: dibuja"), pero ahora es mucho más simple hacer modificaciones, porque los números que dependen uno de otro están en una misma ubicación. Por ejemplo, el dibujo del cuello está basado en la variable neckHeight. El grupo de variables al principio del código controla los aspectos del robot que queremos cambiar: ubicación, altura del cuerpo y altura del cuello. Puedes observar algunas de las posibles variaciones posibles en la figura; de izquierda a derecha, acá están los valores correspondientes:

Cuando alteras tu propio código para usar variables en vez de números, planea los cambios cuidadosamente y después haz las modificaciones en pasos cortos. Por ejemplo, cuando este programa fue escrito, cada variable fue creada de a una a la vez para minimizar la complejidad de la transición. Solo después de que una variable era creada y el código era ejecutado para asegurarse de que funcionara correctamente, se añadía una siguiente variable:

```
var x = 60;                                          // Coordenada X
var y = 420;                                        // Coordenada Y
var bodyHeight = 110;                               // Altura del cuerpo
var neckHeight = 140;                               // Altura del cuello
var radius = 45;
var ny = y - bodyHeight - neckHeight - radius;  // Y del cuello

function setup(){
  createCanvas(170, 480);
  strokeWeight(2);
  ellipseMode(RADIUS);
}

function draw() {
  background(204);

  // Cuello
  stroke(102);
  line(x +  2, y - bodyHeight, x +  2, ny);
  line(x + 12, y - bodyHeight, x + 12, ny);
```

```
  line(x + 22, y - bodyHeight, x + 22, ny);

  // Antenas
  line(x + 12, ny, x - 18, ny - 43);
  line(x + 12, ny, x + 42, ny - 99);
  line(x + 12, ny, x + 78, ny + 15);

  // Cuerpo
  noStroke();
  fill(102);
  ellipse(x, y - 33, 33, 33);
  fill(0);
  rect(x - 45, y - bodyHeight, 90, bodyHeight - 33);
  fill(102);
  rect(x - 45, y - bodyHeight + 17, 90, 6);

  // Cabeza
  fill(0);
  ellipse(x + 12, ny, radius, radius);
  fill(255);
  ellipse(x + 24, ny - 6, 14, 14);
  fill(0);
  ellipse(x + 24, ny - 6, 3, 3);
  fill(153);
  ellipse(x, ny - 8, 5, 5);
  ellipse(x + 30, ny - 26, 4, 4);
  ellipse(x + 41, ny + 6, 3, 3);
}
```

Capítulo 5. Respuesta

El código que responde a acciones de entrada del ratón, teclado u otros dispositivos depende en que el programa corra continuamente. Ya nos enfrentamos a las funciones setup() y draw() en el Capítulo 1. Ahora aprenderemos más sobre qué hacen y cómo usarlas para reaccionar a entradas al programa.

Una vez y para siempre

El código dentro del bloque draw() corre desde el principio al final, luego se repite hasta que cierras el programa cuando cierras la ventana. Cada iteración a través del bloque draw() es llamado un cuadro o frame. (La tasa de cuadros por defecto es de 60 cuadros por segundo, pero esto puede ser modificado).

Ejemplo 5-1: la función draw()

Para observar como la función draw() funciona, corre este ejemplo:

```
function draw() {
  //Muestra en la consola el contador de cuadros
  print("Estoy dibujando");
  print(frameCount);
}
```

Verás lo siguiente:

```
Estoy dibujando
1
Estoy dibujando
2
Estoy dibujando
3
...
```

En el ejemplo anterior, las funciones print() escriben el texto "Estoy dibujando" seguido del contador actual de cuadros, tarea efectuada por la variable especial frameCount. El texto aparece en la consola de tu navegador.

Ejemplo 5-2: la función setup()

Para complementar la repetitiva función draw(), p5.js posee la función setup() que solo corre una vez cuando el programa empieza:

```
function setup() {
  print("Estoy empezando");
}

function draw() {
  print("Estoy corriendo");
}
```

Cuando corres el código, en la consola se escribe lo siguiente:

```
Estoy empezando
Estoy corriendo
Estoy corriendo
Estoy corriendo
...
```

El texto "Estoy corriend" sigue escribiéndose en la consola hasta que el programa es parado.

En algunos navegadores, en vez de escribir una y otra vez "Estoy corriendo", lo imprimirá solo una vez, y después para cada subsecuente vez, incrementará un número junto a la línea, representando el número total de veces que la línea ha sido impresa de corrido.

En un programa típico, el código dentro de setup() es usado para definir las condiciones iniciales. La primera línea es usualmente la función createCanvas(), a menudo seguida de código para definir los colores de relleno y trazado iniciales. (Si no incluyes la función createCanvas(), el lienzo para dibujar tendrá una dimensión de 100x100 pixeles por defecto).

Ahora sabes cómo usar setup() y draw() en mayor detalle, pero esto no es todo.

Hay una ubicación adicional dónde has estado poniendo código - también puedes poner variables globales fuera de setup() y draw(). Esto se hace más claro cuando listamos el orden en que el código es ejecutado.

1. Las variables declaradas fuera de setup() y draw() son creadas.
2. El código dentro de setup() es ejecutado una vez.
3. El código dentro de draw() corre continuamente.

Ejemplo 5-3: setup(), te presento a draw()

El siguiente ejemplo pone en práctica todos estos conceptos:

```
var x = 280;
var y = -100;
var diameter = 380;
```

```
function setup() {
  createCanvas(480, 120);
  fill(102);
}

function draw() {
  background(204);
  ellipse(x, y, diameter, diameter);
}
```

Seguir

Como el código está corriendo continuamente, podemos seguir la posición del ratón y usar estos números para mover elementos en la pantalla.

Ejemplo 5-4: seguir al ratón

La variable mouseX graba la coordenada x, y la variable mouseY graba la coordenada y:

```
function setup() {
  createCanvas(480, 120);
  fill(0, 102);
  noStroke();
}

function draw() {
  ellipse(mouseX, mouseY, 9, 9);
}
```

En este ejemplo, cada vez que el código en el bloque draw() es ejecutado, un nuevo círculo es añadido al lienzo. La imagen fue hecha moviendo el ratón para controlar la posición del círculo. Como la función de relleno está definida para ser parcialmente transparente, las áreas negras más densas muestran dónde el ratón estuvo más tiempo o se movió más lento. Los círculos que están más separados muestran dónde el ratón estuvo moviéndose más rápido.

Ejemplo 5-5: el punto te persigue

En este ejemplo, un nuevo círculo es añadido al lienzo cada vez que el código dentro de draw() es ejecutado. Para refrescar la pantalla y sollo mostrar el círculo más reciente, escribe la función background() al principio del bloque draw() antes que la figura sea dibujada:

```
function setup() {
  createCanvas(480, 120);
  fill(0, 102);
```

```
  noStroke();
}

function draw() {
  background(204);
  ellipse(mouseX, mouseY, 9, 9);
}
```

La función background() pinta el lienzo completo , así que asegúrate de ponerlo antes que las otras funciones dentro de draw(). Si no haces esto, las figuras dibujadas antes serán borradas.

Ejemplo 5-6: dibuja de forma continua

Las variables pmouseX y pmouseY guardan la posición del ratón en el cuadro anterior. Como mouseX y mouseY, estas variables especiales son actualizadas cada vez que draw() es ejecutado. Cuando las combinas, pueden ser usadas para dibujar líneas continuas al conectar las posiciones actual y más reciente:

```
function setup() {
  createCanvas(480, 120);
  strokeWeight(4);
  stroke(0, 102);
}

function draw() {
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

Ejemplo 5-7: define el grosor sobre la marcha

Las variables pmouseX y pmouseY también pueden ser usadas para calcualr la velocidad del ratón. Esto se hace midiendo la distancia entre la posición actual y la más reciente del ratón. Si el ratón se está moviendo lentamente, la distancia es pequeña, pero si se empieza a mover más rápido, la distancia se incrementa. Una función llamada dist() simplifica este cálculo, como se muestra en el siguiente ejemplo. Aquí, la velocidad del ratón es usada para definir el grosor de la línea dibujada

```
function setup() {
  createCanvas(480, 120);
  stroke(0, 102);
}

function draw() {
  var weight = dist(mouseX, mouseY, pMouseX, pMouseY);
```

```
  strokeWeight(weight);
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

Ejemplo 5-8: el suavizado lo hace

En el ejemplo 5-7, los valores del ratón son convertidos directamente a posiciones en la pantalla. Pero a veces queremos que estos valores sigan al ratón más libremente - que se queden atrás para creen un movimiento más fluido. Esta técnica es llamada suavizado. Con el suavizado, hay dos valores: el valor actual y el valor objetivo (ver Figura 5-1). A cada paso en el programa, el valor actual se mueve un poco más cerca del valor objetivo:

```
var x = 0;
var easing = 0.01;

function setup() {
  createCanvas(220, 120);
}

function draw() {
  var targetX = mouseX;
  x += (targetX - x) * easing;
  ellipse(x, 40, 12, 12);
  print(targetX + " : " + x);
}
```

El valor de la variable x está siempre acercándose a targetX. La velocidad con la que lo alcanzo es definida por la variable de easing, un número entre 0 y 1. Un valor pequeño de easing causa más retraso que un valor más grande. Con un valor de easing de 1, no hay retraso. Cuando corres el ejemplo 5-8, los valores actuales son mostrados en la consola a través de la función print(). Cuando muevas el mouse, observa cómo los números están alejados, pero cuando dejas de moverlo, el valor de x se acerca al valor de targetX.

Todo el trabajo en este ejemplo ocurre en la línea que empieza con x+=. Aquí, se calcula la diferencia entre el valor objetivo y el actual, y luego es multiplicada por la variable easing y añadida a x para llevarla más cerca que el objetivo.

Ejemplo 5-9: suaviza las líneas

En este ejemplo, la técnica de suavizado es aplicada al Ejemplo 5-7. En comparación, las líneas son más fluidas:

```
var x = 0;
var y = 0;
```

```
var px = 0;
var py = 0;
function setup() {
  createCanvas(480, 120);
  stroke(0, 102);
}

function draw() {
  var targetX = mouseX;
  x += (targetX - x) * easing;
  var targetY = mouseY;
  y += (targetY - y) * easing;
  var weight = dist(x,y,px,py);
  strokeWeight(weight);
  line(x,y,px,py);
  py = y;
  px = x;
}
```

Click

Además de la ubicación del ratón, p5.js también mantiene registro de si el botón del ratón ha sido presionado o no. La variable mouseIsPressed tiene un valor diferente cuando el botón del ratón está presionado. La variable mouseIsPressed es una variable boolean, lo que significa que solo tiene dos posibles valores: verdadero (true) o falso (falso). El valor de mouseIsPressed es verdadero cuando un botón es presionado.

Ejemplo 5-10: haz click con el ratón

La variable mouseIsPressed es usada en conjunto con la declaración if para determinar si una línea de código será ejecutada o no. Prueba este ejemplo antes de sigamos explicado:

```
function setup() {
  createCanvas(240, 120);
  strokeWeight(30);
}

function draw() {
  background(204);
  stroke(102);
  line(40, 0, 70, height);
  if (mouseIsPressed == true) {
    stroke(0);
  }
```

```
  line(0,70,width,50);
}
```

En este programa, el código dentro del blocke if sólo corre cuando el botón del ratón es presionado. Cuando el botón no está presionado, el código es ignorado. Como el for loop discutido en "Repetition", el bloque if tiene una prueba (test) que es evaluada a verdadero (true) o falso (false).

```
if (test) {
  statements
}
```

Cuando el test es true, el código dentro del bloque es ejecutado y cuando es falso, no es ejecutado. El computador determina si el test es true o false al evaluar la expresión dentro del paréntesis. (Si quieres refrescar tu memoria, el ejemplo 4-6 discute en mayor detalle expresiones relacionales). El símbolo == compara los valores a la izquierda y la derecha para probar si son equivalentes o no. El símbolo == es diferente del operador de asignación, el símbolo unitario =. el símbolo == pregunta, "¿son estas cosas iguales?", mientras que el símbolo = define el valor de una variable

Nota

Es un error común, incluso para programadores avanzados, escribir = en el código en vez de ==. p5.js no siempre te advertirá cuándo lo hagas, así que sé cuidadoso.

Alternativamente, la prueba en draw() puede ser escrita así:

```
if (mouseIsPressed) {
```

Las variables Boolean, incluyendo a mouseIsPressed, no necesitan la comparación explícita con el operador ==, porque su valor es solo o true o false.

Ejemplo 5-11: detección de no clickeado

Un bloque if te da la oportunidad de correr una porción de código o de ignorarla. Puedes extender la funcionalidad del bloque if con el bloque else, permitiendo que tu programa escoja entre dos opciones. El código dentro del bloque else corre cuando el valor de la prueba del bloque if es false. Por ejemplo, el color de trazado de un programa puede ser negro cuando el botón del ratón no es presionado y puede cambiar a negro cuando sí es presionado:

```
function setup() {
  createCanvas(240, 120);
  strokeWeight(30);
}
function draw() {
```

```
  background(204);
  stroke(102);
  line(40, 0, 70, height);
  if (mouseIsPressed) {
    stroke(0);
  } else {
    stroke(255);
  }
  line(0, 70, width, 50);
}
```

Ejemplo 5-12: Múltiples botones del ratón

p5.js también registra cuál botón del ratón es presionado si es que tienes más de uno en tu ratón. La variable mouseButton puede tener uno de estos tres valores: LEFT, CENTER o RIGHT. Para probar cuál de los botones es presionado, el operador == es necesario, como se muestra a continuación:

```
function setup() {
  createCanvas(120, 120);
  strokeWeight(30);
}

function draw() {
  background(204);
  stroke(102);
  line(40, 0, 70, height);
  if (mouseIsPressed) {
    if (mouseButton == LEFT) {
      stroke(255);
    } else {
      stroke(0);
    }
    line(0,70,width,50);
  }
}
```

Un programa puede tener muchas más estructuras if y else (ver Figura 5-2) que las encontradas en estos ejemplos cortos. Pueden ser concatenadas en una larga serie con distintas pruebas, y los bloques if pueden estar anidados dentro de otros bloques if para hacer decisiones más complejas.

Ubicación

Una estructura if puede ser usada con los valores de mouseX y mouseY para determinar la ubicación del cursos dentro de la ventana.

Ejemplo 5-13: encuentra el cursos

En este ejemplo, buscamos el cursor para ver si está a la izquierda o hacia la derecha de la línea y luego movemos la línea hacia el cursor:

```
var x;
var offset = 10;

function setup() {
  createCanvas(240, 120);
  x = width/2;
}

function draw() {
  background(204);
  if (mouseX > x) {
    x += 0.5;
    offset = -10;
  }
  if (mouseX < x) {
    x -= 0.5;
    offset = 10;
  }
  //dibuja una flecha izquierda o derecha según el valor del "offset"
  line(x, 0, x, height);
  line(mouseX, mouseY, mouseX + offset    , mouseY - 10);
  line(mouseX, mouseY, mouseX + offset    , mouseY + 10);
  line(mouseX, mouseY, mouseX + offset * 3, mouseY);
}
```

Para escribir programas que tengan interfaces gráficas de usuario (botones, casillas, barras deslizadoras, etc.) necesitamos escribir código que sepa cuando el curso está dentro de un área de la pantalla. Los siguientes dos ejemplos introducen cómo verificar si el cursor está dentro de un círculo y de un rectángulo. El código está escrito en una forma modular variables, para que pueda ser usado para comprobar con cualquier círculo o rectángulo mediante la modificación de los valores.

Ejemplo 5-14: los bordes de un círculo

Para la prueba con el círculo, usamos la función dist() para obtener la distancia desde el centro del círculo al cursor, luego probamos si este valor es menor que el radio del círculo (ver Figura 5-3). Si lo es, sabemos que estamos dentro del círculo. En este ejemplo, cuando el cursos está denrro del área del círcuo, su tamaño aumenta:

```
var x = 120;
var y = 60;
var radius = 12;

function setup() {
  createCanvas(240, 120);
  ellipseMode(RADIUS);
}

function draw() {
  background(204);
  var d = dist(mouseX, mouseY, x, y);
  if (d < radius) {
    radius++;
    fill(0);
  } else {
    fill(255);
  }
  ellipse(x, y, radius, radius);
}
```

Ejemplo 5-15: Los bordes de un rectángulo

Usaremos otro enfoque para probar si el curso está dentro de un rectángulo. Hacemos cuatro pruebas separadas para comprobar si el cursor está en el lado correcto de cada uno de los lados del rectángulo, luego comparamos cada resultado de las pruebas y si todas son true, entonces sabemos que el cursor está dentro. Esto es ilustrado en la Figura 5-4. Cada paso es simple, pero lucen complicados al combinarse entre sí:

```
var x = 80;
var y = 30;
var w = 80;
var h = 60;

function setup() {
  createCanvas(240, 120);
}
```

```
function draw() {
  background(204);
  if ((mouseX > x) && (mouseX < x+w) &&
      (mouseY > y) && (mouseY < y+h)) {
        fill(0);
      } else {
        fill(255);
      }
      rect(x, y, w, h);
}
```

La prueba en la declaración if es un poco más complicada que lo que hemos visto hasta el momento. Cuatro pruebas individuales (como mouseX > x) son combinadas con el operador lógico AND, el símbolo &&, para asegurarse que cada expresión relacional en la secuencia sea true. Si alguna de ellas es false, el test entero es false y el color de relleno no será negro.

Tipo

p5.js mantiene registro de cualquier tecla que sea presionada en el teclado, además de la última tecla presionada. Tal como la variable mouseIsPressed, la variable keyIsPressed is true cuando cualquier tecla es presionada, y false cuando no hay teclas presionadas.

Ejemplo 5-16: presiona una tecla

En este ejemplo, la segunda línea es dibujada solo cuando hay una tecla presionada:

```
function setup() {
  createCanvas(240, 120);
}

function draw() {
  background(204);
  line(20, 20, 220, 100);
  if (keyIsPressed) {
    line(220, 20, 20, 100);
  }
}
```

La variable key guarda la tecla presionada más recientemente. A diferencia de la variable boolean keyIsPressed, que se revierte a false cada vez que la tecla es soltada, la variable key mantiene su valor hasta que la siguiente tecla es presionada. El siguiente ejemplo usa el valor de key para dibujar el caracter en la pantalla. Cada vez qeu una nueva tecla es presionada, el valor se actualiza y un nuevo caracter es dibujado. Algunas teclas, como Shift y Alt, no tienen un caracter visible, así que si las presionas, nada será dibujado.

Ejemplo 5-17: dibuja algunas letras

Este ejemplo introduce la función textSize() para definir el tamaño de las letras, la función textAlign() para centrar el texto en su coordenada x y la función text() para dibujar la letra. Estas funciones serán discutidas en mayor detalle en "Fonts".

```
function setup{
  createCanvas(120,120);
  textSize(64);
  textAlign(CENTER);
  fill(255);
}

function draw() {
  background(0);
  text(key, 60, 80);
}
```

Usando una estructura if, podemos probar si una tecla específica es presionad y escoger dibujar algo distinto en la pantalla a modo de respuesta.

Ejemplo 5-18: revisar diferentes teclas

En este ejemplo, revisamos si las teclas N o H son presionadas. Usamos el comparador de comparación, el símbolo ==, para revisar si el valor de la variable key es igual a los caracteres que estamos buscando:

```
function setup() {
  createCanvas(120, 120);
}

function draw() {
  background(204);
  if (keyIsPressed) {
    if ((key == 'h') || (key == 'H')) {
      line(30, 60, 90, 60);
    }
    if ((key == 'n') || (key == 'N')) {
      line(30, 20, 90, 100);
    }
  }
  line(30, 20, 30, 100);
  line(90, 20, 90, 100);
}
```

Cuando revisamos si está siendo presionada la tecla H o la N, necesitamos revisar tanto para las letras en mayúscula como en minúscula, en caso de que alguien presione la tecla Shift o tenga la función Caps Lock activada. Combinamos ambas pruebas con el operador lógico OR, el símbolo ll. Si traducimos la segunda declaración if en este ejemplo a lenguaje plano, dice "Si la tecla 'h' es presionada OR la tecla 'H' es presionada". A diferencia del operador lógico AND (el símbolo &&), solo una de estas expresiones necesita ser true para que la prueba entera sea evaluada a true.

Algunas teclas son más difíciles de detectar, porque no están asociadas a una letra en particular. Teclas como Shift, Alt, y las flechas están codificadas. Tenemos que revisar el código con la variable keyCode para revisar qué tecla es. Los valores más frecuentes de keyCode son ALT, CONTROL y SHIFT, además de las teclas con flechas UP_ARROW, DOWN_ARROW, LEFT_ARROW Y RIGHT_ARROW.

Ejemplo 5-19: mover con las flechas

El siguiente ejemplo muestra cómo usar las flechas izquierda y derecha para mover un rectángulo.

```
var x = 215;
function setup() {
  createCanvas(480, 120);
}

function draw() {
  if (keyIsPressed) {
    if (keyCode == LEFT_ARROW) {
      x--;
    } else if (keyCode == RIGHT_ARROW) {
      x++;
    }
  }
  rect(x, 45, 50, 50);
}
```

Toque

Para dispositivos que lo soportan, p5.js mantiene registr de si la pantalla es tocada y su ubicación. Como la variable mouseIsPressed, la variable touchIsdown es true cuando la pantalla es tocada, y false cuando no.

Ejemplo 5-20: toca la pantalla

En este ejemplo, la segunda línea es dibujada solo si la pantalla es tocada

```
function setup() {
  createCanvas(240, 120);
}

function draw() {
  background(204);
  line(220, 20, 220, 100);
  if (touchIsdown) {
    line(220, 20, 20, 100);
  }
}
```

Como las variables mouseX y mouseY, las variables touchX y touchY almacenan las coordenadas x e y del punto donde la pantalla está siendo tocada.

Ejemplo 5-21: rastrea el dedo

En este ejemplo, un nuevo círculo es añadido al lienzo cada vez que el código en draw() es ejecutado. Para refrescar la pantalla y solo mostrar el círculo más nuevo, escribe la función background() al inicio de draw() antes de dibujar la figura:

```
function setup() {
  createCanvas(480, 120);
  fill(0, 102);
  noStroke();
}

function draw() {
  ellipse(touchX, touchY, 15, 15);
}
```

Mapeo

Los números que son creados por el ratón y por el teclado muchas veces necesitan ser modificados para ser útiles dentro del programa. Por ejemplo, si un bosquejo tiene un ancho de 1920 pixeles y los valores de mouseX son usados para definir el color del fondo, el rango de 0 a 1920 de mouseX necesitará ser escalado para moverse en un rango de 0 a 255 para controlar mejor el color. Esta transformación puede ser hecha con una ecuación o con una función llamada map().

Ejemplo 5-22: mapeo de valores a un rango

En este ejemplo, la ubicación de dos líneas es controlada por la variable mouseX. La línea gris está sincronizada con la posición del cursor, pero la línea negra se mantiene más cerca del centro de la pantalla y se aleja de la línea blanca en los bordes izquierdos y derechos.

```
function setup() {
  createCanvas(240, 120);
  strokeWeight(12);
}

function draw() {
  background(204);
  stroke(102);
  line(mouseX, 0, mouseX, height);  // Línea gris
  stroke(0);
  var mx = mouseX/2 + 60;
  line(mx, 0, mx, height);          // Línea negra
}
```

La función map() es una manera más general de hacer este tipo de cambio. Convierte una variable desde un rango de valores a otro. El primer parámetro es la variable a ser convertida, el segundo y tercer valor son los valores mínimo y máximo de esa variable, y el cuarto y quinto son los valores mínimo y máximo deseados. La función map() esconde la matemática detrás de esta conversión.

Ejemplo 5-23: Mapeo con la función map()

Este ejemplpo reescribe el Ejemplo 5-22 usando map():

```
function setup() {
  createCanvas(240, 120);
  strokeWeight(12);
}

function draw() {
  background(204);
  stroke(255);
  line(120, 60, mouseX, mouseY);  // Línea blanca
  stroke(0);
  var mx = map(mouseX, 0, width, 60, 180);
  line(120, 60, mx, mouseY);      // Línea negra
}
```

La función map() hace que el código sea fácil de leer, porque los valores máximo y mínimo están claramente escritos como parámetros. En este ejemplo, los valores de mouseX entre 0 y width son convertidos a números entre 60 (cuando mouseX es 0) y 180 (cuando mouseX es width). Encontrarás esta útil función map() en muchos ejemplos a lo largo de este libro.

Robot 3: respuesta

Este programa usa las variables introducidas en Robot 2 (ver "Robot 2: variables") y hace posible cambiarlas mientras el programa corre de manera que las figuras respondan al ratón. El código dentro del bloque draw() es ejecutado muchas veces por segundo. En cada cuadro, las variables definidas en el programa cambian en respuesta a las variables mouseX y mouseIsPressed.

La variable mouseX controla la posición del robot con la técnica de suavizado para que los movimientos sean menos instantáneos y se vean más naturales. Cuando un botón del ratón es presionado, los valores de neckHeight y bodyHeight cambian para hacer al robot más corto:

```
var x = 60;           // Coordenada x
var y = 440;          // Coordenada y
var radius = 45;      // Radio de la cabeza
var bodyHeight = 160; // Altura del cuerpo
var neckHeight = 70;  // Altura del cuello

var easing = 0.04;

function setup() {
  createCanvas(360, 480);
  strokeWeight(2);
  ellipseMode(RADIUS);
}

function draw() {
  var targetX = mouseX;
  x += (targetX - x) * easing;
  if (mouseIsPressed) {
    neckHeight = 16;
    bodyHeight = 90;
  } else {
    neckHeight = 70;
    bodyHeight = 160;
  }

  var neckY = y - bodyHeight - neckHeight - radius;

  background(204);
```

```
  // Cuello
  stroke(102);
  line(x + 12, y - bodyHeight, x + 12, neckY);

  // Antenas
  line(x + 12, neckY, x - 18, neckY - 43);
  line(x + 12, neckY, x + 42, neckY - 99);
  line(x + 12, neckY, x + 78, neckY + 15);

  // Cuello
  noStroke();
  fill(102);
  ellipse(x, y - 33, 33, 33);
  fill(0);
  rect(x - 45, y - bodyHeight, 90, bodyHeight - 33);

  // Cabeza
  fill(0);
  ellipse(x + 12, neckY, radius, radius);
  fill(255);
  ellipse(x + 24, neckY - 6, 14, 14);
  fill(0);
  ellipse(x + 24, neckY - 6, 3, 3);
}
```

Capítulo 6. Trasladar, rotar, escalar

Una técnica alternativa para posicionar y mover objetos en la pantalla es cambiar el sistema de coordenadas de la pantalla. Por ejemplo, puedes mover una figura 50 pixeles a la derecha, o puedes mover la ubicación de la coordenada (0,0) 50 pixeles a la derecha - el resultado visual en la pantalla es el mismo.

Al modificar el sistema de coordenadas por defecto, podemos crear diferentes transformaciones incluyendo traslación, rotación y escalamiento.

Traslación

Trabajar con transformaciones puede ser difícil, pero la función translate() es la más sencilla, así que empezaremos con esta. Como muestra la Figura 6-1, esta función puede cambiar el sistema de coordenadas hacia la izquierda, derecha, arriba y abajo.

Ejemplo 6-1: trasladando la ubicación

En este ejemplo, observa que el rectángulo está dibujado en la coordenada (0,0), pero está en otra posición en el lienzo, porque es afectado por la función translate():

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
}
```

La función translate() define la coordenada (0,0) de la pantalla a la ubicación del ratón (mouseX y mouseY). Cada vez que el bloque draw() se repite, el rectángulo es dibujado en el nuevo origen, derivado de la posición actual del ratón.

Ejemplo 6-2: múltiples traslados

Después de que la transformación es realizada, es aplicada a todas las veces que la función draw() es ejecutada. Observa lo que pasa cuando una segunda función translate() es añadida para controlar un segundo rectángulo:

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
  translate(35, 10);
  rect(0, 0, 15, 15);
}
```

Los valores para la función translate() son acumulados. El pequeño rectángulo es trasladado según mouseX + 35 0,0y mouseY + 10. Las coordenadas x e y para ambos rectángulos son (0,0), pero las funciones translate() los mueven a otras posiciones en el lienzo. Sin embargo, incluso cuando las transformaciones se acumulan dentro del bloque draw(), se reinician cada vez que la función draw() empieza de nuevo.

Rotación

La función rotate() rota el sistema de coordenadas. Tiene un parámetro, que es el ángulo (en radianes) a rotar. Siempre rota relativo a (0,0), lo que se conoce como rotar en torno al origen. La Figura 3-2 muestra los valores de ángulo en radianes. La figura 6-2 muestra la diferencia entre rotar con números positivos y negativos.

Ejemplo 6-3: rotación de la esquina

Para rotar una figura, primero define el ángulo de rotación con rotate(), luego dibuja la figura. En este bosquejo, el parámetro para rotar (mouseX / 100.0) tendrá un valor entre 0 y 1.2 para definir el ángulo de rotación porque mouseX tendrá un valor entre 0 y 120, el ancho del lienzo según lo definido en createCanvas():

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(mouseX / 100.0);
  rect(40, 30, 160, 20);
}
```

Ejemplo 6-4: rotación del centro

Para rotar una figura en torno a su propio centro, deben ser dibujada con la coordenada (0,0) en su centro. En este ejemplo, como la figura tiene un ancho de 160 y una altura de 20 según lo definido en la función rect(), es dibujada en la coordenada (-80, -10) para poner la coordenada (0,0) al centro de la figura:

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(mouseX / 100.0);
  rect(-80, -10, 160, 20);
}
```

El par anterior de ejemplos muestra cómo rotar alrededor de un sistema de coordenadas (0,0), ¿pero qué otras posibilidades hay? Puedes usar las funciones translate() y rotate() para mayor control. Cuando son combinadas, el orden en que aparecen afecta el resultado. Si el sistema de coordenadas es trasladado y después rotado, es diferente que primero rotar y después mover el sistema de coordenadas.

Ejemplo 6-5: traslación, después rotación

Para girar una figura en torno a su centro a un lugar en la pantalla lejos del orgien, primero usa la función traslate() para mover la figura a la ubicación donde quieres la figura, luego usa rotate(), y luego dibuja la figura con su centro en la coordenada (0,0):

```
var angle = 0.0;

function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  rotate(angle);
  rect(-15, -15, 30, 30);
  angle += 0.1;
}
```

Ejemplo 6-6: rotación, después traslación

El siguiente ejemplo es idéntico al Ejemplo 6-5, excepto que translate() y rotate() ocurren en el orden inverso. La figura ahora rota alrededor de la esquina superior izquierda, con la distancia desde la esquina definida por translate():

```
var angle = 0.0;

function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  rotate(angle);
  translate(mouseX, mouseY);
  rect(-15, -15, 30, 30);
  angle += 0.1;
}
```

Nota

Puedes usar también las funciones rectMode(), ellipseMode() y imageMode() hacen más simple dibujar figuras desde su centro. Puedes leer sobre estas funciones en la Referencia de p5.js.

Ejemplo 6-7: un brazo articulado

En este ejemplo, hemos puesto juntas una serie de funciones translate() y rotate() para crear un brazo articulado. Cada función translate() mueve la posición de las líneas, y cada función rotate() añade a la rotación previa para doblar más:

```
var angle = 0.0;
var angleDirection = 1;
var speed = 0.005;

function setup() {
  createCanvas(120, 120);
}

function draw() {
  background(204);
  translate(20, 25);  // Mover a la posición inicial
  rotate(angle);
```

```
  strokeWeight(12);
  line(0, 0, 40, 0);
  translate(40, 0);   // Mover la siguiente articulación
  rotate(angle * 2.0);
  strokeWeight(6);
  line(0, 0, 30, 0);
  translate(30, 0);
  rotate(angle * 2.5);
  strokeWeight(3);
  line(0, 0, 20, 0);

  angle += speed * angleDirection;
  if ((angle > QUARTER_PI) || (angle < 0)) {
    angleDirection *= -1;
  }
}
```

La variable angle crece desde 0 hasta QUARTER_PI (un cuarto del valor de pi), luego decae hasta que es menor que cero, luego el ciclo se repite. El valor de la variable angleDirection está siempre entre 1 y -1 para hacer que el valor de angle correspondiente crezca o decrezca.

Escalar

La función scale() estira las coordenadas del lienzo. Como las coordenadas se expanden o se contraen cuando cambia la escala, todo lo que está dibujado en el lienzo aumenta o disminuye sus dimensiones. El monto de escalamiento está escrito en porcentajes decimales. Entonces, el parámetro 1.5 en la función scale() resulta en un 150% y 3 es 300% (Figura 6-3).

Ejemplo 6-8: escalamiento

Como rotate(), la función scale() transforma desde el origen. Entonces, tal como rotate(), para escalar una figura desde su centro, debemos trasladar su ubicación, escalar y luego dibujar con el centro en la coordenada (0,0):

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  scale(mouseYX / 60.0);
  rect(-15, -15, 30, 30);
}
```

Ejemplo 6-9: manteniendo los trazos constantes

De las líneas gruesas del Ejemplo 6-8, puedes ver cómo la función scale() afecta el grosor del trazado. Para mantener un grosor de trazado consistente a medida que la figura se escala, divide el trazado deeseado por el valor escalar:

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  translate(mouseX, mouseY);
  var scalar = mouseX / 60.0;
  scale(scalar);
  strokeWeight(1.0 / scalar);
  rect(-15, -15, 30, 30);
}
```

Push y pop

Para aislar los efectos de la transformación para que no afecten otras funciones, usa las funciones push() y pop(). When ejecutas push(), graba una copia del sistema de coordenadas actual y luego restaura ese sistema cuando ejecutas pop(). Esto es útil cuando las transformaciones son necesarias para una figura, pero no son deseadas para otras.

Ejemplo 6-10: aislando transformaciones

En este ejemplo, el rectángulo pequeño siempre dibuja en la misma pop():

```
function setup() {
  createCanvas(120, 120);
  background(204);
}

function draw() {
  push();
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
  pop();
  translate(35, 10);
```

```
  rect(0, 0, 15, 15);
}
```

Nota

Las funciones push() y pop() siempre se usan en pares. Por cada push(), tiene que haber un correspondiente pop().

Robot 4: trasladar, rotar, escalar

Las funciones translate(), rotate() y scale() son utilizadas para modificar el bosquejo del robot. En relación al ejemplo Robot 3: respuesta, translate() es usado para hacer el código más fácil de leer. Aquí, observa cómo ya no es necesario el valor de x a cada función de dibujo porque la función translate() mueve todo. Similarmente, la función scale() es usada para definir las dimensiones para todo el robot. Cuando el ratón no está presionado, el tamaño es de un 60% y cuando sí está presionado, es de un 100% en relación a las coordenadas originales. La función rotate() es usada dentro del loop para dibujar una línea, rotarla un poco, luego dibujar una segunda línea, luego rotarla un poco más, y así hasta que el loop ha dibujado 30 líneas en forma de círculo para estilizar el pelo de la cabeza del robot:

```
var x = 60;           // Coordenada x
var y = 440;          // Coordenada y
var radius = 45;      // Radio de la cabeza
var bodyHeight = 180; // Altura del cuerpo
var neckHeight = 40;  // Altura del cuello

var easing = 0.04;

function setup() {
  createCanvas(360, 480);
  strokeWeight(2);
  ellipseMode(RADIUS);
}

function draw() {
  var neckY = -1 * (bodyHeight + neckHeight + radius);

  background(204);

  translate(mouseX, y); // Mueve todo a (mouseX, y)

  if (mouseIsPressed) {
    scale(1.0);
  } else {
    scale(0.6);       // 60% de tamaño si el ratón está presionado
```

```
  }

  // Cuerpo

  noStroke();
  fill(102);
  ellipse(0, -33, 33, 33);
  fill(0);
  rect(-45, -bodyHeight, 90, bodyHeight - 33);

  // Cuello
  stroke(102);
  line(12, -bodyHeight, 12, neckY);

  // Pelo
  push();
  translate(12, neckY);
  var angle = -PI/30.0;
  for (var i = 0l i <= 30; i++) {
    line(80, 0, 0, 0);
    rotate(angle);
  }
  pop();

  // Cabeza
  noStroke();
  fill(0);
  ellipse(12, neckY, radius, radius);
  fill(255);
  ellipse(24, neckY - 6, 14, 14);
  fill(0);
  ellipse(24, neckY - 6, 3, 3);
}
```