Table of Contents

Prefacio

p5.js 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:

- 1/Hola: aprende sobre p5.js
- 2/Aprendiendo a programar: crea tu primer programa en p5.js
- 3/Dibujar: define y dibuja figuras simples
- 4/Variables: almacena, modifica y reusa datos
- 5/Respuesta: controla e influencia programas con el ratón, teclado y toques.
- 6/Trasladar, rotar, escalar: transforma las coordenadas
- 7/Media: carga y muestra medios, incluyendo imágenes y tipografías
- 8/Movimiento: mueve y haz coreografía de figuras
- 9/Funciones: construye nuevos módulos de código
- 10/Objetos: crea módulos de código que combinan variables y funciones
- 11/Arreglos: simplifica el trabajo con listas de variables
- 12/Datos: carga y visualiza datos
- 13/Extensión: aprender sobre sonido y DOM

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 en http://github.com/lmccart/gswp5.js-code.

Este libro está aquí para ayudarte a que puedas hacer tu trabajo. En general, puedes usar el código en este libro en tus programas y documentación. No necesitas contactarnos para permiso a menos que estés reproduciendo una porción significativa del código. Por ejemplo, si estás escribiendo un programa que usa múltiples trozos de código de este libro no requiere permiso. Vender o distribuir ejemplos de libros Make: sí requiere permiso. Responder una pregunta citando este libro y citando ejemplos de código no requiere permiso. Incorporar un monto significativo de código de ejemplo de este libro en la documentación de tu producto sí requiere permiso.

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 para el nuestro.

Un pequeño grupo de individuos ha, durante años, contribuido tiempo y energía esenciales a Processing. Dan Shiffman es nuestro compañero en la Fundación Processing, la organización 501(c)(3) que apoya el software Processing. La mayor parte del código principal de Processing 2.0 y Processing 3.0 proviene de las mentes brillantes de Andres Colubri y Manindra Moharana.

Scott Murray, Jamie Kosoy y Jon Gacnik han construido una maravillosa infraestructura web para el proyecto. James Grady está luciéndose en la interfaz de usuario 3.0. Le agradecemos a Florian Jenett por sus años de trabajo en diversas áreas, incluyendo foros, sitio web y diseño. Elie Zananiri y Andreas Schlegel han creado la infraestructura para construir y documentar librerías contribuidas y han pasado innumerables horas curando las listas. Muchos otros han contribuido significativamente al projecto, los datos precisos están disponibles en https://github.com/processing.

Este libro surgió del proceso de enseñar con Processing en UCLA. Chandler McWilliams ha sido instrumental en definir estas clases. Casey le agradece a los estudiantes de pregrado en el Departamento de Diseño y Artes Mediales en UCLA por su energía y entusiasmo. Sus

ayudantes del curso han sido grandes colaboradores al definir cómo Processing es enseñado. Agradecimientos a Tatsuya Saito, John Houck, Tyler Adams, Aaron Siegel, Casey Alt, Andres Colubri, Michael Kontopoulos, David Elliot, Christo Allegra, Pete Hawkes y Lauren McCarthy.

p5.js es desarrollado por una gran comunidad de contribuidores a lo largo del mundo. Dan Shiffman, Jason Sigal, Sam Lavigne, K. Adam White, Chandler McWilliams, Evelyn Eastmond, los miembros del grupo de trabajo de p5 en ITP, los asistentes a la primera Conferencia de Contribuidores de p5.js en el Frank-Ratchye STUDIO para la Investigación Creativa en la Universidad Carnegie Mellon, y los estudiantes y mentores del Processing Google Summer of Code han sido instrumentales en hacer que p5.js sea lo que es hoy. Apoyo significativo para el proyecto ha sido provisto por la Processing Foundation, NYU ITP, RISD y Bocoup. Puedes ver la lista completa de contribuidores en http://p5js.org/contribute/#contributors. Lauren también le agrade a Kyle McDonald por su perpetuo apoyo e inspiración.

Este libro ha sido transformado por las artísticas ilustraciones de Taeyoon Choi. Fueron desarrolladas en parte a través de una residencia en el Frank-Ratchye STUDIO para la Investigación Creativa en la Carnegie Mellon University, con apoyo del programa Art Works del National Endowment for the Arts. Charlotte Stiles ayudó tremendamente con la edición y los ejemplos e imágenes de este libro.

A través del financiamento del Aesthetics and Computation Group (1996-2002) en el MIT Media Lab, John Maeda hizo todo esto posible.

Capítulo 1. Hola

p5.js sirve para escribir software que produce imágenes, animaciones e interacciones. La intención es escribir una línea de código 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:

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:

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:

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 creado 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 se detiene.

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);
```

El rectángulo es dibujado sobre la elipse porque está después en el código

```
  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);
```

La elipse es dibujada sobre el rectángulo porque está después en el código

```
  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);        // Color azul oscuro
  fill(255, 0, 0);              // Color rojo
  ellipse(132, 82, 200, 200);   // Círculo rojo
  fill(0, 255, 0);              // Color verde
  ellipse(228, -16, 200, 200);  // Círculo verde
  fill(0, 0, 255);              // Color azul
  ellipse(268, 118, 200, 200);  // Círculo azul
}
```

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);    // Color azul claro
  fill(255, 0, 0, 160);         // Color rojo
  ellipse(132, 82, 200, 200);   // Círculo rojo
  fill(0, 255, 0, 160);         // Color verde
  ellipse(228, -16, 200, 200);  // Círculo verde
  fill(0, 0, 255, 160);         // Color azul
  ellipse(268, 118, 200, 200);  // Círculo azul
}
```

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);
```

fill(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
```

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="" {="" fill(255,="" 140);="" el
```

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,="'
```

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 <= 16="" width;="" x="" +="15)" {="" ellipse(x="" y,="'
```

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 corriendo" 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 cursor

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 ||. 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 ejemplo 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);="" rotate(ang
```

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);
}
```

Capítulo 7. Medios

p5.js es capaz de dibujar más que simplemente líneas y figuras. Es tiempo de aprender cómo crear imágenes y texto en nuestros programas para extener las posibilidades visuales a fotografía, diagramas detallados y diversas tipografías.

Antes de que hagamos esto, primero tenemos que hablar un poco sobre servidores. Hasta este punto, hemos estado viendo el archivo index.html directamente en el navegador. Esto funciona bien para correr animaciones simples. Sin embargo, si quieres hacer cosas como cargar una imagen externa en tu bosquejo, tu navegador no lo va a permitir. Si revisas la consola, te encontrarás con un error conteniendo el término "cross-origin". Para cargar archivos externos, tienes que correr un servidor. Un servidor es un programa que funciona como un manejador de capas. Responde cuando escribes una URL en la barra de direcciones, y sirve los archivos correspondientes a ti para su visualización.

Existen diferentes maneras de correr servidores. Visita https://github.com/processing/p5.js/wiki/Local-server para ver las instrucciones de cómo correr un servidor en sistemas Mac OS X, Windows y Linux. Una vez que lo tengas configurado, ¡estás listo para cargar media!

Hemos subido algunos archivos para que los uses en los ejemplos de este capítulo: http://p5js.org/learn/books/media.zip.

Descarga este archivo, descomprímelo en tu escritorio (o en otro lugar conveniente) y anota su ubicación.

Nota

Para descomprimir en Mac OS X, basta con hacer doble click en el archivo y se creará un directorio llamado media. En Windows, haz doble click en el archivo media.zip, el que abrirá una nueva ventana. En esa ventana, arrastra el directorio llamado media al escritorio.

Crea un nuevo bosquejo, y copia el archivo lunar.jpg desde el directorio media que acabas de descomprimir al directorio de tu bosquejo.

Nota

En Windows y Mac OS X, las extensiones de los archivos están escondidas por defecto. Es una buena idea cambiar esta opción para que siempre veas el nombre completo de tus archivos. En Mac OS X, selecciona Preferencias desde el menú principal, y luego asegúrate que "Mostrar la extensión completa" esté seleccionado en la pestaña de opciones avanzadas. En Windows, busca las Opciones de Directorio, y selecciona la opción ahí.

Imágenes

Estos son los tres pasos que tienes que seguir antes de que puedas dibujar una imagen en la pantalla:

1. Añade la imagen al directorio del bosquejo.
2. Crea una variable para almacenar la imagen.
3. Carga la imagen a la variable con la función loadImage().

Ejemplo 7-1: carga una imagen

Para cargar una imagen, introduciremos una nueva función llamada preload(). La función preload() corre una vez y antes de que la función setup() corra. Generalmente deberías cargar tus imagenes y otros archivos dentro de preload() para asegurarte que estén completamente cargadas antes de que tu programa empiece a correr. Discutiremos esto en mayor profundidad más adelante en el capítulo.

Después de que los tres pasos son completados, puedes dibujar la imagen en la pantalla con la función image(). El primer parámetro de image() especifica la imagen a dibujar, el segundo y tercero son las coordenadas x e y:

```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

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

function draw() {
  image(img, 0, 0);
}
```

Los parámetros opcionales cuarto y quinto determinan el ancho y altura de la imagen a dibujar. Si no se usan los parámetros cuarto y quinto, la imagen es dibujada al tamaño original que fue creada.

Los siguientes ejemplos muestran cómo trabajar con más de una imagen en el mismo programa y cómo escalar la imagen.

Ejemplo 7-2: carga más imágenes

Para este ejemplo, necesitarás agregar el archivo capsule.jpg (que está dentro del directorio media que descargaste) al directorio de tu bosquejo.

```
var img1;
var img2;

function preload() {
  img1 = loadImage("lunar.jpg");
  img2 = loadImage("capsule.jpg");
}

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

function draw() {
  image(img1, -120, 0);
  image(img1, 130, 0, 240, 120);
  image(img2, 300, 0, 240, 120);
}
```

Ejemplo 7-3: mover las imágenes con el ratón

Cuando las variables mouseX y mouseY son usadas como los parámetros cuarto y quinto de la función image(), el tamaño de la imagen cambia con el movimiento del ratón:

```
var img;

function preload() {
  img = loadImage("lunar.jpg");
}

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

function draw() {
  background(0);
  image(img, 0, 0, mouseX * 2, mouseY * 2);
}
```

Nota

Cuando una imagen es mostrada más grande o pequeña que su tamaño original, puede aparecer distorsionada. Ten cuidado en preparar tus imágenes a los tamaños en que serán usadas. Cuando el tamaño de la imagen es cambiado con la función image(), el archivo original en tu directorio del bosquejo no cambia.

p5.js puede cargar y mostrar imágenes raster en los formatos JPEG, PNG y GIF, además de imágenes vector en el formato SVG. Puedes convertir imágenes a los formatos JPEG, PNG, GIF y SVG usando programas como GIMP, Photoshop e Illustrator. La mayor parte de las cámaras digitales graban sus imágenes en el formato JPEG, pero usualmente necesitan ser reducidas en tamaño para ser usadas con p5.js. Una cámara digital típica crea una imagen que es varias veces más grande que el área de dibujo de gran parte de los bosquejos creados en p5.js. Cambiar el tamaño de estas imágenes antes de que sean añadidas al directorio del bosquejo hace que los bosquejos carguen más rápido, corran más eficientemente y ahorra espacio en el disco duro.

Las imágenes GIF, PNG y SVG soportan transparencia, lo que significa que los pixeles pueden ser invisibles o parcialmente visibles (recuerda la discusión de color() y valores alpha en el Ejemplo 3-17). Las imágenes GIF tienen transparencia de 1 bit, lo que significa que los pixeles son totalmente opacos o totalmente transparentes. Las imágenes PNG soportan transparencia de 8 bits, lo que significa que cada pixel tiene una variable de opacidad. Los siguientes ejemplos usan los archivos clouds.gif y clouds.png para mostrar las diferencias entre los formatos. Las imágenes están dentro del directorio media que has descargado anteriormente. Asegúrate de incluirlas al directorio de tu bosquejo antes de probar cada ejemplo.

Ejemplo 7-4: transparencia con GIF

```
var img;

function preload() {
  img = loadImage("clouds.gif");
}

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

function draw() {
  background(204);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

Ejemplo 7-5: transparencia con PNG

```
var img;

function preload() {
  img = loadImage("clouds.png");
}

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

function draw() {
  background(204);
  image(img, 0, 0);
  image(img, 0, mouseY * -1);
}
```

Ejemplo 7-6: mostrar una imagen SVG

```
var img;

function preload() {
  img = loadImage("network.svg");
}

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

function draw() {
  background(204);
  image(img, 0, 0);
  image(img, mouseX, 0);
}
```

Nota

Recuerda incluir la extensión apropiada del archivo (.gif, .jpg, .png o .svg) cuando cargas la imagen. También asegúrate que el nombre de la imagen esté escrito exactamente como aparece en el archivo, incluyendo mayúsculas y minúsculas.

Asincronicidad

¿Por qué necesitamos cargar las imágenes en preload()? ¿Por qué no usamos setup()? Hasta este punto, hemos estado asumiendo que nuestros programas corren desde la parte superior a la finerior, con cada línea de código siendo ejecutada completamente antes de avanzar a la siguiente. Aunque esto es generalmente cierto, en el caso de algunas funciones como cargar imágenes, tu navegador empezará el proceso de cargar la imagen, pero se saltará a la siguiente línea antes de que la imagen haya terminado de cargarase. Esto recibe el nombre de asincronicidad, o de función asíncrona. Es un poco inesperado al principio, pero esto permite que las páginas carguen y corran más rápido en la web.

Para ver esto con mayor claridad, considera el siguiente ejemplo. Es idéntico al ejemplo 7-1, excepto que usamos loadImage() dentro de setup() en vez de preload().

Ejemplo 7-7: demonstrando la asincronicidad

```
var img;

function setup() {
  createCanvas(480, 120);
  img = loadImage("lunar.jpg");
  noLoop();
}

function draw() {
  background(204);
  image(img, 0, 0);
}
```

Cuando corras este programa, te darás cuenta que el lienzo para pintar está gris y que la imagen no está sinedo mostrada. El bosquejo corre la función setup() primero y luego corre la función draw(). En la línea de loadImage(), empieza a cargar la imagen, pero continúa con el resto de setup() y con draw() antes de que la imagen esté cargada. La función image() no es capaz de cargar una imagen que todavía no esté cargada.

Para ayudar con este problema, p5.js tiene la función preload(). A diferencia de setup(), la función preload() forza al programa a esperar hasta que todo esté cargado. Es mejor hacer las llamadas para cargar archivos dentro de preload(), y hacer toda la configuración en setup().

Alternativamente, en vez de usar preload(), puedes usar algo llamado función de retrollamada (callback).

Ejemplo 7-8: cargando con un callback

```
function setup() {
  createCanvas(480, 120);
  loadImage("lunar.jpg", drawImage);
  noLoop();
}

function draw() {
  background(200);
}

function drawImage(img) {
  image(img, 0, 0);
}
```

En este ejemplo, añadimos un segundo argumento a loadImage(), que es la función que queremos que corra después de que la carga es completada. Una vez que la imagen ha cargado, la función callback drawImage() es automáticamente llamada, con un argumento, la imagen que ha sido cargada.

No hay necesidad de crear una variable global para guardar la imagen. La imagen es pasada directamente a la función callback, con el nombre del parámetro escogido en la definición de la función.

Fuentes de letras

p5.js puede mostrar texto en fuentes distintas que la por defecto. Puedes usar cualquier fuente que esté en tu computador (son llamadas fuentes del sistema). Ten en cuenta que si estás compartiendo esto en la web, otra gente necesitará añadir la fuente de sistema para poder ver el texto en la fuente que escogiste. Hay un número de fuentes que la mayor parte de los computadores y dispositivos tienen: estas incluyen "Arial", "Courier", "Courier New", "Georgia", "Helvetica", "Palatino", "Times New Roman", "Trebuchet MS" y "Verdana".

Ejemplo 7-9: dibujando con fuentes

Puedes usar la función textFont() para configurar la fuente actual. Puedes dibujar letras en la pantalla con la función text() y puedes cambiar el tamaño con la función textSize():

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

```
function draw() {
  background(102);
  textSize(32);
  text("one small step for man...", 25, 60);
  textSize(16);
  text("one small step for man...", 27, 90);
}
```

El primer parámetro de text() son los caracteres a ser dibujados en la pantalla. (Date cuenta que los caracteres están entre comillas). Los segundo y tercer parámetros definen la ubicación horizontal y vertical. La ubicación está basada en la base del texto (ver Figura 7-1).

Ejemplo 7-10: usar una fuente de la web

Si no quieres estar limitado a esta pequeña lista de fuentes, puedes usar una de la web. Dos sitios web que son buenos recursos para encontrar fuentes web con licencias abiertas para usar con p5.js son GoogleFonts y la Open Font Library.

Para usar una webfont en tu prorgrama, deberás referenciarla en tu archivo index.html. Cuando escoges una fuente desde cualquiera de estas librerías mencionadas, te mostará una línea de código para añadir a tu archivo HTML. Cuando copias y pegas este código en cualquier parte dentro de la sección head de tu HTML, tu archivo se verá algo así:

```html ```

Una vez que hayas referenciado la fuente, puedes usarla con textFont() tal como las fuentes de sistema:

```
function setup() {
  createCanvas(480, 120);
  textFont("Source Code Pro");
}

function draw() {
  background(102);
  textSize(28);
  text("one small step for man...", 25, 60);
  textSize(16);
  text("one small step for man...", 27, 90);
}
```
```

Ejemplo 7-11: define el trazado del texto y el relleno

Tal como las figuras, el texto es afectado por las funciones stroke() y fill(). El siguiente ejemplo resulta en texto negro con borde blanco:

```
function setup() {
  createCanvas(480, 120);
  textFont("Source Code Pro");
  fill(0);
  stroke(255);
}

function draw() {
  background(102);
  textSize(28);
  text("one small step for man...", 25, 60);
  textSize(16);
  text("one small step for man...", 27, 90);
}
```

Ejemplo 7-12: dibuja el texto en un recuadro

Puedes también definir que el texto se dibuje dentro de un recuadro añadiendo los parámetros cuarto y quinto para especificar el ancho y altura del recuadro:

```
function setup() {
  createCanvas(480, 120);
  textFont("Source Code Pro");
  textSize(24);
}

function draw() {
  background(102);
  text("one small step for man...", 26, 24, 240, 100);
}
```

Ejemplo 7-13: guardar el texto en una variable

En el ejemplo anterior, las palabras dentro de la función text() hacen que el código sea difícil de leer. Podemos guardar estas palabras en una variable para asegurar que el código sea más modular. Aquí está una nueva versión del ejemplo anterior que usa una variable:

```
var quote = "one small step for man...";

function setup() {
  createCanvas(480, 120);
  textFont("Source Code Pro");
  textSize(24);
}

function draw() {
  background(102);
  text(quote, 26, 24, 240, 100);
}
```

Hay un conjunto de funciones adicionales que afectan cómo las letras son mostradas en la pantalla. Son explicadas, con ejemplos, en la categoría de Tipografía en la Referencia de p5.js.

Robot 5: media

A diferencia de los robots creados con líneas y rectángulos dibujados en p5.js durante los capítulos anteriores, estos robots fueron creados con un programa de dibujo vectorial. Para algunas figuras, es más fácil apuntar y hacer click con un software como Inkscape o Illustrator que definir las figuras con coordenadas en código.

Existe un compromiso al seleccionar una técnica para creación de imágenes por sobre otra. Cuando defines figuras en p5.js, existe mayor flexibilidad para modificarlas mientras el programa está corriendo. Si las figuras están definidas en otro lugar y luego cargadas a p5.js, los cambios están limitados a la posición, ángulo y tamaño. Cuando cargas cada robot desde un archivo SVG, como este ejemplo muestra, las variaciones destacadas en el Robot 2 (ver "Robot 2: variables") son imposibles.

Las imágenes pueden ser cargadas con un programa para traer visuales creadas en otros programas o capturadas con una cámara. Con esta imagen en el fondo, nuestros robots ahora están buscando formas de vida en Noruega en los inicios del siglo 20.

Los archivos SVG y PNG usados en este ejemplo pueden ser descargados desde http://p5js.org/learn/books/media.zip:

```
var bot1;
var bot2;
var bot3;
var landscape;

var easing = 0.05;
var offset = 0;
```

Precarga las imágenes

```
function preload() {
  bot1 = loadImage("robot1.svg");
  bot2 = loadImage("robot2.svg");
  bot2 = loadImage("robot3.svg");
  landscape = loadImage("alpine.svg");
}

function setup() {
  createCanvas(720, 480);
}

function draw() {
```

Definir la imagen "landscape" como función Esta imagen debe tener el mismo ancho y altura que el programa

```
  background(landscape);
```

Definir el offset izquierdo y derecho y aplicar el suavizado para hacer la transición más suave

```
  var targetOffset = map(mouseY, 0, height, -40, 40);
  offset += (targetOffset - offset) * easing;
```

Dibuja el robot izquierdo

```
  image(bot1, 85 + offset, 65);
```

Dibuja el robot derecho más pequeño y haz que tenga un menor offset

```
  var smallerOffset = offset * 0.7;
  image(bot2, 510 + smallerOffset, 140, 78, 248);
```

Dibuja el robot más pequeño, dale un offset menor

```
  smallerOffset *= -0.5;
  image(bot3, 410 + smallerOffset, 225, 39, 124);
}
```

Capítulo 8. Movimiento

Tal como un folioscopio, la animación en la pantalla es creada para dibujar una imagen, luego otra, y así. La ilusión de movimiento fluido es creada por persistencia de visión. Cuando un conjunto de imágenes similares es presentado a una tasa suficiente, nuestros cerebros traducen estas imágenes en movimiento.

Cuadros

Para crear movimiento fluido, p5.js trata de correr el código dentro de draw() a una tasa de 60 cuadros por segundo. Un cuadro es una ejecución de la función draw() y la tasa de cuadros equivale a cuántos cuadros son dibujados cada segundo. Entonces, un programa que dibuja 60 cuadros cpor segundo corre todo el código dentro de la función draw() 60 veces por segundo.

Ejemplo 8-1: ve la tasa de cuadros

Para confirmar la tasa de cuadros, podemos usar la consola del navegador que aprendimos a usar en el Capítulo 1. La función frameRate() te arroja la velocidad actual de tu programa. Abre la consola, corre este programa y revisa los valores impresos:

```
function draw() {
  var fr = frameRate();
  print(fr);
}
```

Ejemplo 8-2: define la tasa de cuadros

La función frameRate() puede también cambiar la velocidad a la que el programa corre. Cuando es ejecutada sin parámetro (como en el Ejemplo 8-1), arroja la actual tasa de cuadros. Sin embargo, cuando la función frameRate() es llamada con un parámetro, define la tasa de cuadros a ese valor. Para ver el resultado, ejecuta las distintas versiones de frameRate() de este ejemplo, descomentándolas:

```
function setup() {
  frameRate(30);    // Treinta cuadros por segundo
```

frameRate(12); // Doce cuadros por segundo frameRate(2); // Dos cuadros por segundo frameRate(0.5); // Un cuadro cada dos segundos

```
}
function draw() {
  var fr = frameRate();
```

```
  print(fr);
}
```

Nota:

p5.js trata de correr el código a una tasa de 60 cuadros por segundo, pero si tarda más de de 1/60 segundos en correr el método draw(), entonces la tasa decrecerá. La función frameRate() especifica solo la tasa máxima, y la tasa real para cualquier programa depende en el computador corriendo el código.

Velocidad y dirección

Para crear exemplos de movimiento fluido, creamos variables que guardan números y los modifican un poco cada cuadro.

Ejemplo 8-3: mueve una figura

El siguiente ejemplo mueve una figura de izquierda a derecha, actualizando la variable x:

```
var radius = 40;
var x = -radius;
var speed = 0.5;

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

function draw() {
  background(0);
  x += speed;   // Aumenta el valor de x
  arc(x, 60, radius, radius, 0.52, 5.76);
}
```

Cuando corres este código, observarás que la figura se mueve más allá del borde derecho de la pantalla cuando el valor de la variable x es mayor que el ancho de la ventana. El valor de x sigue aumentando, pero la figura ya no es visible.

Ejemplo 8-4: dar la vuelta

Existen muchas alternativas a este comportamiento, que puedes escoger de acuerdo a tu preferencia. Primero, extenderemos el código para mostrar cómo mover la figura de vuelta al borde izquierdo de la pantalla después de que desaparece del borde derecho. En este caso,

imagina la pantalla como un cilindro aplanado, con la figura moviéndose por fuera para volver al borde izquierdo:

```
var radius = 40;
var x = -radius;
var speed = 0.5;

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

function draw() {
  background(0);
  x += speed;     // Aumenta el valor de x
  if (x > width + radius) { // Si la figura está fuera de la pantalla
    x = - radius; // Mueve la figura al borde izquierdo
  }
  arc(x, 60, radius, radius, 0.52, 5.76);
}
```

En cada viaje alrededor de draw(), el código prueba si el valor de x ha aumentado más allá del ancho de la pantalla (sumado al radio de la figura). Si lo ha hecho, hacemos que el valor de x sea negativo nuevamente, para que cuando siga aumentando, entre a la pantalla por la izquierda. Mira la Figura 8-1 para ver un diagrama de cómo funciona.

Ejemplo 8-5: rebota contra la pared

En este ejemplo, extenderemos el Ejemplo 8-3 para que la figura cambie de dirección cuando llegue a un borde, en vez de volver a aparecer por la izquierda. Para hacer que esto pase, añadimos una nueva variable para almacenar la dirección de la figura. Un valor de dirección de 1 mueve la figura hacia la derecha, mientras que un valor de -1 la mueve hacia la izquierda:

```
var radius = 40;
var x = 110;
var speed = 0.5;
var direction = 1;

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

function draw() {
  background(0);
```

```
  x += speed * direction;
  if ((x > width-radius) || (x < radius)) {
    direction = -direction;   // Cambiar dirección
  }
  if (direction == 1) {
    arc(x, 60, radius, radius, 0.52, 5.76); // Hacia la derecha
  } else {
    arc(x, 60, radius, radius, 3.67, 8.9);  // Hacia la izquierda
  }
}
```

Cuando la figura llega a un borde, este código invierte la dirección de la figura, cambiando el signo de la variable dirección. Por ejemplo, si la variable dirección es positiva cuando la figura llega a un borde, el código la invierte a negativa.

Posiciones intermedias (tweening)

A veces quieres animar una figura para ir de un punto de la pantalla a otro. Con unas pocas líneas de código, puedes configurar la posición inicial y final, y luego calcular las posiciones entremedio (tween) en cada cuadro.

Ejemplo 8-6: calcula las posiciones intermedias

Para hacer que el ejemplo de este código sea modular, hemos creado un grupo de variables en la parte superior. Corre el código unas cuantas veces y cambia los valores para ver cómo este código puede mover a la figura desde cualquier ubicación a cualquier otra en cualquier rango de velocidades. Cambia la variable step para alterar la velocidad:

```
var startX = 20;  // Coordenada x inicial
var stopX = 160;  // Coordenada x final
var startY = 30;  // Coordenada y inicial
var stopY = 80;   // Coordenada y final
var x = startX;   // Coordenada x actual
var y = startY;   // Coordenada y actual
var step = 0.005; // createCanvas para cada paso (0.0 a 1.0)
var pct = 0.0;    // Porcentaje avanzado (0.0 a 1.0)

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

function draw() {
  background(0);
  if (pct < 1.0) {
    x = startX + ((stopX - startX) * pct);
```

```
    y = startY + ((stopY - startY) * pct);
    pct += step;
  }
  ellipse(x, y, 20, 20);
}
```

Aleatorio

A diferencia del movimiento linear y suave típico en las gráficas por computadora, el movimiento en el mundo físico es usualmente idiosincrático. Por ejemplo, si pensamos en una hoja flotando hacia la tierra, o una hormiga caminando por un terreno rugoso. Podemos simular las cualidades impredecibles del mundo generando números aleatorios. La función random() calcula estos valores, podemos definir un rango para afinar la cantidad de desorden en un programa.

Ejemplo 8-7: genera valores aleatorios

El siguiente ejemplo corto imprime valores aleatorios en la consola, con el rango limitado por la posición del ratón:

```
function draw() {
  var r = random(0, mouseX);
  print(r);
}
```

Ejemplo 8-8: dibuja aleatoriamente

Construyendo sobre el Ejemplo 8-7, este ejemplo usa valores de la función random() para cambiar la posición de líneas en el lienzo. Cuando el ratón está a la izquierda del lienzo, el cambio es pequeño; si se mueve a la derecha, los valores de random() aumentan y el movimiento se torna más exagerado. Como la función random() está dentro de un for loop, un nuevo valor aleatorio es calculado para cada punto de cada línea:

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

function draw() {
  background(204);
  for (var x = 20; x < width; x += 20) {
    var mx = mouseX / 10;
    var offsetA = random(-mx, mx);
    var offsetB = random(-mx, mx);
```

```
    line(x + offsetA, 20, x - offsetB, 100);
  }
}
```

Ejemplo 8-9: mueve figuras aleatoriamente

Cuando se usa para mover figuras alrededor de la pantalla, los valores aleatorios pueden generar imágenes que son más naturales en apariencia. En el siguiente ejemplo, la posición del círculo es modificada por valores aleatorios en cada ejecución de draw(). Como la función background() no es usad, las posiciones anteriores permanecen dibujadas:

```
var speed = 2.5;
var diameter = 20;
var x;
var y;

function setup() {
  createCanvas(240, 120);
  x = width/2;
  y = height/2;
  background(204);
}

function draw() {
  x += random(-speed, speed);
  y += random(-speed, speed);
  ellipse(x, y, diameter, diameter);
}
```

Si observas este ejemplo el tiempo suficiente, el círculo podría dejar la pantalla y volver. Esto depende del azar, pero podríamos añadir unas estructuras if o usar la función constrain() para hacer que el círculo no deje la pantalla.

La función constrain() limita el valor a un rango específico, el que puede ser usado para mantener x e y dentro de los límites del lienzo. Al reemplazar la función draw con el siguiente código, te asegurarás que la elipse permanezca en la pantalla:

```
function draw() {
  x += random(-speed, speed);
  y += random(-speed, speed);
  x = constrain(x, 0, width);
  y = constrain(y, 0, height);
  ellipse(x, y, diameter, diameter);
}
```

Nota

La función randomSeed() puede ser usada para forzar a randon() para producir la misma secuencia de números cada vez que un programa es ejecutado. Esto es descrito con mayor detalle en la Referencia de p5.js.

Temporizadores

Cada programa de p5.js cuenta el monto de tiempo que ha pasado desde que empezó. Cuenta en milisegundos (milésimas de segundo), así que después de 1 segundo el contador está en 1.000, después de 5 segundos está en 5.000 y después de un minuto en 60.000. Podemos usar este contador para gatillar animaciones en momentos específicos. La función millis() arroja el valor del contador.

Ejemplo 8-10: el tiempo pasa

Puedes ver cómo el tiempo pasa cuando corres este programa:

```
function draw() {
  var timer = millis();
  print(timer);
}
```

Ejemplo 8-11: gatillando eventos temporizados

Cuando se combina con un bloque if, los valores de millis() pueden ser usados para secuenciar tanto animaciones como eventos del programa. Por ejemplo, después de que han pasado dos segundos, el código dentro del bloque if puede gatillar un cambio. En este ejemplo, las variables llamadas time1 y time2 deerminan cuándo cambiar el valor de la variable x:

```
var time1 = 2000;
var time2 = 4000;
var x = 0;

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

function draw() {
  var currentTime = millis();
  background(204);
  if (currentTime > time2) {
    x -= 0.5;
```

```
  } else if (currentTime > time1) {
    x += 2;
  }
  ellipse(x, 60, 90, 90);
}
```

Circular

Si eres un as de la trigonometría, ya sabes cuán increíbles son las funciones seno y coseno. Si no lo eres, esperamos que los siguientes ejemplos pueden gatillar tu interés. no discutiremos la matemática en detalle aquí, pero aquí mostraremos unas pocas aplicaciones para generar movimiento fluido.

La figura 8-2 muestra una visualización de valores de la función seno y cómo se relacionan con ángulos. En la parte superior e inferior de la onda, observa cómo la tasa de cambio (el cambio en el eje vertical) desacelera, para y luego cambia de dirección. Es esta cualidad de la curva lo que genera un movimiento interesante.

Las funciones sin() y cos() en p5.js arrojan valores entre -1 y 1 para la función seno y coseno del ángulo especificado. Tal como arc(), los ángulos deben ser escritos en radianes (ver Ejemplo 3-7 y Ejemplo 3-8 para un recordatorio de cómo funcionan los radianes). Para ser útil para dibujar, los valores float arrojados por sin() y cos() son usualmente multiplicados por un valor más grande.

Ejemplo 8-12: valores de la onda sinusoidal

Este ejemplo muestra cómo los valores de sin() oscilan entre -1 y 1 a medida que el ángulo aumenta. Con la función map(), la variable sinval es convertida desde este rango a valores de 0 a 255. Este nuevo valor es usado para definir el color del fondo del lienzo:

```
var angle = 0.0;

function draw() {
  var sinval = sin(angle);
  print(sinval);
  var gray = map(sinval, -1, 1, 0, 255);
  background(gray);
  angle += 0.1;
}
```

Ejemplo 8-13: movimiento de una onda sinusoidal

Este ejemplo muestra cómo estos valores son convertidos a movimiento:

```
var angle = 0.0;
var offset = 60;
var scalar = 40;
var speed = 0.05;

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

function draw() {
  background(0);
  var y1 = offset + sin(angle) * scalar;
  var y2 = offset + sin(angle + 0.4) * scalar;
  var y3 = offset + sin(angle + 0.8) * scalar;
  ellipse( 80, y1, 40, 40);
  ellipse(120, y2, 40, 40);
  ellipse(160, y3, 40, 40);
  angle += speed;
}
```

Ejemplo 8-14: movimiento circular

Cuando las funcioens sin() y cos() son usadas en conjunto, pueden producir movimiento circular. Los valores de la función cos() proveen los valores de la coordenada x, y los valores de la función sin() proveen la coordenada y. Ambos son multiplicados por una variable llamada scalar para cambiar el radio del movimiento y son sumados con un valor offset para situar el centro de un movimiento circular:

```
var angle = 0.0;
var offset = 60;
var scalar = 30;
var speed = 0.05;

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

function draw() {
  var x = offset + cos(angle) * scalar;
  var y = offset + sin(angle) * scalar;
  ellipse(x, y, 40, 40);
  angle += speed;
}
```

Ejemplo 8-15: espirales

Un pequeño cambio hecho para aumentar el valor scalar en cada cuadro produce una espiral en vez de un círculo:

```
var angle = 0.0;
var offset = 60;
var scalar = 2;
var speed = 0.05;

function setup() {
  createCanvas(120, 120);
  fill(0);
  background(204);
}

function draw() {
  var x = offset + cos(angle) * scalar;
  var y = offset + sin(angle) * scalar;
  ellipse(x, y, 2, 2);
  angle += speed;
  scalar += speed;
}
```

Robot 6: movimiento

En este ejemplo, las técnicas para movimiento aleatorio y circular son aplicadas al robot. La función background() fue removida para ver más claramente cómo la posición del robot y su cuerpo cambian.

En cada cuadro, un número aleatorio entre -4 y 4 es añadido a la coordenada x, y un número aleatorio entre -1 y 1 es añadido a la coordenada y. Esto causa que el robot se mueva más de izquierda a derecha que de arriba a abajo. Los números calculados por la función sin() cambian la altura del cuello para que oscile entre 50 y 100 pixeles de altura:

```
var x = 180;          // Coordenada x
var y = 400;          // Coordenada y
var bodyHeight = 153; // Altura del cuerpo
var neckHeight = 56;  // Altura del cuello
var radius = 45;      // Radio de la cabeza
var angle = 0.0;      // Ángulo de movimiento

function setup() {
  createCanvas(360, 480);
  ellipseMode(RADIUS);
```

```
  background(204);
}

function draw() {
```

Cambia la posición en un monto aleatorio pequeño

```
  x += random(-4, 4);
  y += random(-1, 1);
```

Cambia la altura del cuello

```
  neckHeight = 80 + sing(angle) * 30;
  angle += 0.05;
```

Ajusta la altura de la cabeza

```
  var ny = y - bodyHeight - neckHeight - radius;
```

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);
}
```

Capítulo 9. Funciones

Las funciones son los bloques fundamentales de los programas hechos en p5.js. Han aparecido en cada ejemplo que hemos presentado. Por ejemplo, hemos frecuentemente usado la función createCanvas(), la función line(), y la función fill(). Este capítulo muestra cómo escribir nuevas funciones para extender las capacidades de p5.js más allá de sus características incorporadas.

El poder de las funciones es su modularidad. Las funciones son unidades de software independientes que son usadas para construir programas complejos - como bloques de LEGO, donde cada tipo de ladrillo sirve para un propósito específico y para lograr un modelo complejo requiere usar las diferentes partes en conjunto. Como con las funciones, el verdadero poder de estos ladrillos es la habilidad de construir muchas formas distintas usando el mismo conjunto de elementos. El mismo grupo de LEGOs que forma una navae espacial puede ser reusado para construir un camión, un rascacielos y muchos otros objetos.

Las funciones son útiles si quieres dibujar una forma más compleja como un árbol, una y otra vez. La función para dibujar un árbol puede estar compuesta con las funciones incorporadas de p5.js, como line(), para crear la forma. Después de que el código para dibujar el árbol es escrito, no necesitas pensar sobre los detalles de dibujar un árbol nuevamente - puedes simplemente escribir tree() (o algún otro nombre que le hayas puesto a la función) para dibujar la figura. Las funciones permiten que una secuencia compleja de declaraciones pueda ser abstraída, para que te puedas enfocar en una meta de alto nivel (como dibujar un árbol), y no los detalles de la implementación (las funciones line() que definen la forma del árbol). Una vez que una función es definida, el código dentro de la función no necesita ser repetido.

Funciones básicas

Un computador corre los programas una línea de código a la vez. Cuando una función es ejecutada, el computador salta a donde la función está definida y corre el código ahí, luego vuelve a donde estaba anteriormente.

Ejemplo 9-1: tira los dados

Este comportamiento es ilustrado con la función rollDice() escrita para este ejemplo. Cuando el programa empieza, corre el código en setup() y luego para. El programa toma un desvío y corre el código dentro de rollDice() cada vez que aparece.

```
function setup() {
  print("Listo para lanzar los dados!");
  rollDice(20);
  rollDice(20);
  rollDice(6);
  print("Listo.");
}
```

```
function rollDice(numSides) {
  var d = 1 + int(random(numSides));
  print("Lanzando... " + d);
}
```

Las dos líneas de código en rollDice() seleccionan un número aleatorio entre 1 y el número de caras del dado, e imprime ese número a la consola. Como los números son aleatorios, verás diferentes números cada vez que el prorgama es ejecutado:

Cada vez que la función rollDice() es ejecutada dentro de setup(), el código dentro de la función corre de arriba a abajo, luego el programa continúa con la siguiente línea dentro de setup().

La función random() (descrita en "Aleatorio") arroja un número entre 0 y hasta (pero sin incluir) el número especificado. Entonces random(6) entrega un número entre 0 y 5.99999... Como random() arroja un número con punto decimal, también usamos la función int() para convertir el número a uno entero. Entonces int(random(6)) arrojará 0, 1, 2, 3, 4 o 5. Como muchas otros casos en este libro, contar desde 0 hace más fácil usar los resultados de random() con otros cálculos.

Ejemplo 9-2: otra manera de tirar los dados

Si un programa equivalente hubiera sido hecho sin la función rollDice(), hubiera sido así:

```
function setup() {
  print("Listo para lanzar los dados!");
  var d1 = 1 + int(random(20));
  print("Lanzando... " + d1);
  var d2 = 1 + int(random(20));
  print("Lanzando... " + d2);
  var d3 = 1 + int(random(6));
  print("Lanzando... " + d3);
  print("Listo.");
}
```

La función rolldice() en el Ejemplo 9-1 hace que el código sea más fácil de leer y mantener. El programa es más claro, porque el nombre de la función claramente determina su propósito. En este ejemplo, podemos ver la función random() en setup(), pero su uso no es tan obvio. El número de lados en un dado es más claro con una función: cuando el código dice rollDice(6), es obvio que está simulando el lanzamiento de un dado de seis caras. Además, el Ejemplo 9-1 es fácil de mantener, porque la información no está repetida. La frase Lanzando... se repite tres veces en este caso. Si quieres cambiar el texto a algo distinto, tienes que actualizar el código en tres lugares, en vez de hacer una sola edición dentro de la función rollDice(). Además, como verás en el Ejemplo 9-5, una función puede hacer un programa mucho más

corto (y por lo tanto más fácil de mantener y leer), lo que ayuda a reducir el potencial número de errores.

Hacer una función

En esta sección, dibujaremos una lechuza para explicar los pasos involucrados en hacer una función.

Ejemplo 9-3: dibuja la lechuza

Primero, dibujaremos la lechuza sin usar una función:

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

function draw() {
  background(204);
  translate(110, 110);
  stroke(0);
  strokeWeight(70);
  line(0, -35, 0, -65); // Cuerpo
  noStroke();
  fill(255);
  ellipse(-17.5, -65, 35, 35);  // Pupila izquierda
  ellipse( 17.5, -65, 35, 35);  // Pupila derecha
  arc(0, -65, 70, 70, 0, PI);   // Barbilla
  fill(0);
  ellipse(-14, -65, 8, 8);  // Ojo izquierdo
  ellipse( 14, -65, 8, 8);  // Ojo derecho
  quad(0, -58, 4, -51, 0, -44, -4, -51);  // Pico
}
```

Observa que la función translate() es usada para mover el origen (0, 0) 110 pixeles a la derecha y 110 pixeles hacia abajo. Luego la leechuza es dibujada relativamente al (0,0), con sus coordenadas algunas veces positivas y otras negativas, centradas alrededor del nuevo punto (0,0). (Ver Figura 9-1).

Ejemplo 9-4: Dos son compañía

El código del Ejemplo 9-3 es razonable si solo hay una lechuza, pero cuando añadimos una segunda, el largo del código es casi el doble:

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

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

Lechuza izquierda

```
  translate(110, 110);
  stroke(0);
  strokeWeight(70);
  line(0, -35, 0, -65); // Cuerpo
  noStroke();
  fill(204);
  ellipse(-17.5, -65, 35, 35);  // Pupila izquierda
  ellipse( 17.5, -65, 35, 35);  // Pupila derecha
  arc(0, -65, 70, 70, 0, PI);   // Barbilla
  fill(0);
  ellipse(-14, -65, 8, 8);  // Ojo izquierdo
  ellipse( 14, -65, 8, 8);  // Ojo derecho
  quad(0, -58, 4, -51, 0, -44, -4, -51);  // Pico
```

Lechuza derecha

```
  translate(70, 0);
  stroke(0);
  strokeWeight(70);
  line(0, -35, 0, -65); // Cuerpo
  noStroke();
  fill(255);
  ellipse(-17.5, -65, 35, 35);  // Pupila izquierda
  ellipse( 17.5, -65, 35, 35);  // Pupila derecha
  arc(0, -65, 70, 70, 0, PI);   // Barbilla
  fill(0);
  ellipse(-14, -65, 8, 8);  // Ojo izquierdo
  ellipse( 14, -65, 8, 8);  // Ojo derecho
  quad(0, -58, 4, -51, 0, -44, -4, -51);  // Pico
}
```

El programa crece de 21 líneas de código a 34, el código para dibujar la primera lechuza fue copiado y pegado en el programa y una función translate fue insertada para moverla 70 pixeles a la derecha. Esto es una manera tediosa e ineficiente de dibujar la segunda lechuza, no sin mencionar el dolor de cabeza que será añadir una tercera lechuza con este método.

Pero duplicar el código es innecesario, porque este es el tipo de situación donde una función puede llegar al rescate.

Ejemplo 9-5: una función lechuza

En este ejemplo, una función es introducida para dibujar dos lechuzas con el mismo código. Si hacemos que el código que dibuja una lechuza en la pantalla sea una nueva función, entonces el código solo necesita aparecer una vez en el programa:

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

function draw() {
  background(204);
  owl(110, 110);
  owl(180, 110);
}

function owl(x,y) {
  push();
  translate(x,y);
  stroke(0);
  strokeWeight(70);
  line(0, -35, 0, -65); // Cuerpo
  noStroke();
  fill(255);
  ellipse(-17.5, -65, 35, 35);  // Pupila izquierda
  ellipse( 17.5, -65, 35, 35);  // Pupila derecha
  arc(0, -65, 70, 70, 0, PI);   // Barbilla
  fill(0);
  ellipse(-14, -65, 8, 8);  // Ojo izquierdo
  ellipse( 14, -65, 8, 8);  // Ojo derecho
  quad(0, -58, 4, -51, 0, -44, -4, -51);  // Pico
  pop();
}
```

Puedes ver por las ilustraciones de este ejemplo y del Ejemplo 9-4 que tienen el mismo resultado, pero que este es más corto, porque el código para usar la lechuza aparece solo una vez, dentro de la función llamada owl(). Este código es ejecutado dos veces, porque es llamado dos veces dentro de la función draw(). La lechuza es dibujada en dos ubicaciones distintas porque los parámetros pasados a la función determinan las coordenadas x e y.

Los parámetros son una parte importante de las funciones, porque proveen flexibilidad. Vimos otro ejemplo de esto en la función rollDice(), el parámetro único numSides hizo posible simular un dado de 6 caras, uno de 20 y cualquier otro número de caras. Esto es como otras

funciones de p5.js. Por ejemplo, los parámetros de la función line() hacen posible dibujar una línea de un pixel a otro en el lienzo. Sin los parámetros, la función solo sería capaz de dibujar una línea desde un punto fijo a otro fijo.

Cada parámetro es una variable que es creada cada vez que la función corre. Cuando este ejemplo corre, la primera vez que la función owl es llamada, el valor del parámetro x es 110 y el parámetro y es 110 también. En el segundo uso de la función, el valor de x es 180 y y es nuevamente 110. Cada valor es pasado a la función y luego cada vez que el nombre de la variable aparece dentro de la función, es reemplazado con el valor.

Ejemplo 9-6: aumentando la población

Ahora que tenemos una función básica para dibujar la lechuza en cualquier ubicación, podemos ahora dibujar muchas lechuzas eficientemente poniendo la función dentro de un for loop y cambiando el primer parámetro cada vez que corre el loop:

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

function draw() {
  background(204);
  for (var x = 35; x < width + 70; x += 70) {
    owl(x, 110);
  }
}
```

Insertar la función owl() del Ejemplo 9-5

Es posible seguir añadiendo más y más parámetros a la función para cambiar diferentes aspectos de cómo la lechuza es dibujada. Los valores son pasados a la función para

cambiar el color de la lechuza, la rotación, la escala o el diámetro de los ojos.

Ejemplo 9-7: lechuzas de diferentes tamaños

En este ejemplo, hemos añadido dos parámetros para cambiar el valor de gris y el tamaño de cada lechuza:

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

function draw() {
  background(204);
  randomSeed(0);
  for (var i = 35; i < width + 40; i += 40) {
    var gray = int(random(0, 102));
    var scalar = random(0.25, 1.0);
    owl(i, 110, gray, scalar);
  }
}

function owl(x, y, g, s) {
  push();
  translate(x,y);
  scale(s);   // Define la escala
  stroke(g);  // Define el valor de gris
  strokeWeight(70);
  line(0, -35, 0, -65); // Cuerpo
  noStroke();
  fill(255-g);
  ellipse(-17.5, -65, 35, 35);  // Pupila izquierda
  ellipse( 17.5, -65, 35, 35);  // Pupila derecha
  arc(0, -65, 70, 70, 0, PI);   // Barbilla
  fill(g);
  ellipse(-14, -65, 8, 8);  // Ojo izquierdo
  ellipse( 14, -65, 8, 8);  // Ojo derecho
  quad(0, -58, 4, -51, 0, -44, -4, -51);  // Pico
  pop();
}
```

Valores de retorno

Las funciones pueden hacer un cálculo y luego retornar un valor al programa principal. Hemos ya usado funciones de este tipo, incluyendo random() y sin(). Observa que cuando esta función aparece, el valor de retorno es usualmente asignado a una variable:

```javascript var r = random(1, 10); ```

En este caso, la función random() retorna un valor entre 1 y 10, el que luego es asignado a la variable r.

Las funciones que retornan un valor son frecuentemente usadas como un parámetro a otra función, como por ejemplo:

```javascript point(random(width), random(height)); ```

En este caso, los valores de random() no son asignados a una variable - son pasados como parámetros a la función point() y usados para posicionar el punto dentro del lienzo.

Ejemplo 9-8: retorna un valor

Para hacer que una función retorne un valor, especifica el dato a ser pasado de vuelta con la palabra clave return. En este ejemplo se incluye una función llamada calculateMars() que calcula el peso de una persona u objeto en nuestro planeta vecino:

```
function setup() {
  var yourWeight = 132;
  var marsWeight = calculateMars(yourWeight);
  print(marsWeight);
}

function calculateMars(w) {
  var newWeight = w * 0.38;
  return newWeight;
}
```

Revisa la última línea del código del bloque, que retorna la variable newWeight. En la segunda línea de setup(), el valor es asignado a la variable marsWeight. (Para ver tu propio peso en Marte, cambia el valor de la variable yourWeight a tu peso).

Robot 7: funciones

En contraste con el Robot 2 (ver "Robot 2: variables"), este ejemplo usa una función para dibujar cuatro variaciones del robot dentro del mismo programa. Como la función drawRobot() aparece cuatro veces dentro de la función draw(), el código dentro del bloque drawRobot() es

ejecutado cuatro veces, cada vez con un diferente conjunto de parámetros para cambiar la posición y la altura del cuerpo del robot.

Observa cómo las variables globales en Robot 2 ahora han sido aisladas dentro de la función drawrobot(). Como estas variables aplican solamente a dibujar el robot, ellas tienen que estar dentro de las llaves que definen el bloque de la función drawRobot(). Como el valor de la variable radius no cambia, no necesita ser un parámetro. En cambio, es definida al principio de drawRobot():

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

function draw() {
  background(204);
  drawRobot(120, 420, 110, 140);
  drawRobot(270, 460, 260,  95);
  drawRobot(420, 310,  80,  10);
  drawRobot(570, 390, 180,  40);
}

function drawRobot(x, y, bodyHeight, neckHeight) {

  var radius = 45;
  var ny = y - bodyHeight - neckHeight - radius;
```

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(102);
  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 10. Objetos

La programación orientada a objetos (OOP) es una manera diferente de pensar sobre tus programas. Los objetos son también una manera de agrupar variables con funciones relacionadas. Como ya sabes cómo trabajar con variables y funciones, los objetos simplemente combinan lo que ya has aprendido en un paquete más fácil de entender.

Los objetos son importantes, porque permiten dividir las ideas en bloques más pequeños. Esto se parece al mundo real donde, por ejemplo, los órganos están hechos de tejido, el tejido está hecho de células y así. Similarmente, a medida que tu código se va volviendo más complejo, tienes que pensar en términos de estructuras más pequeñas que forman estructuras más complicadas. Es más fácil escribir y mantener pedazos de código más pequeños y fáciles de entender, que trabajan en conjunto con otros, que es escribir un gran trozo de código que hace todo al mismo tiempo.

Propiedades y métodos

Un objeto es un conjunto de variables y funciones relacionadas. En el contexto de los objetos, una variable se llama propiedad (o variable de instancia) y una función es llamada método. Las propiedades y los métodos funcionan tal como las variables y las funciones vistas en los capítulos anteriores, pero usaremos estos nuevos términos para enfatizar que son parte de un objeto. Para decirlo de otra manera, un objeto combina datos relacionados (propiedades) con acciones y comportamientos relacionados (métodos). La idea es agrupar datos y métodos relacionados.

Por ejemplo, para hacer un modelo de una radio, piensa en los parámetros que pueden ser ajustados y las acciones que pueden afectar estos parámetros:

Propiedades volume, frequency, band(FM, AM), power (on, off) Métodos setVolume, setFrequency, setBand

Modelar un dispostivo mecánico simple es fácil comparado a modelar un organismo como una hormiga o una persona. No es posible reducir un organismo complejo a unas pocas propiedades y métodos, pero es posible modelarlo suficientemente bien como para crear una simulación interesante. El video juego The Sims es un claro ejemplo. Este juego consiste en administrar las actividades diarias de personas simuladas. Los personajes tienen la suficiente personalidad como para hacer un juego adictivo, pero no más que eso. De hecho, ellos solo tienen cinco atributos de personalidad: ordenado, extrovertido, activo, juguetón y simpático. Con el conocimiento de que es posible hacer un modelo altamente simplificado de organismos complejos, podríamos empezar a programar una hormiga con unas pocas propiedades y métodos:

Propiedades: tipo(trabajador, soldado), peso, ancho Métodos: caminar, picar, liberarFeromonas, comer

Si hicieras una lista de las propiedades y métodos de una hormiga, podrías escoger enfocarte en modelar diferentes aspectos de la hormiga. No existe una manera correcta de hacer un modelo, mientras lo hagas apropiado para el próposito de las metas de tu programa.

Define un constructor

Para crear un objeto, empieza por definir una función constructor. Una función constructor es la especificación de un objeto. Usando una analogía arquitectónica, una función constructor es como el plano de una casa, y un objeto es como la casa en sí misma. Cada casa construida con el mismo plano puede tener variaciones, y el plano es la única especificación, no una estructura fija. Por ejemplo, una casa puede ser azul y otra roja, una casa puede tener una chimenea y la otra no. Tal como los objetos, el constructor define los tipos de datos y comportamientos, pero cada objeto (casa) hecho de la misma función constructor (plano) tiene variables (color, chimenea) que tienen distintos valores. Para usar un término más técnico, cada objeto es una instancia y cada instancia tiene su propio conjunto de propiedades y métodos.

Antes de que escribas una función constructor, recomendamos un poco de planificación. Piensa en qué propiedades y métodos deberían tener tus objetos. Haz una lluvia de ideas para imaginar todas las opciones posibles y luego prioriza y haz tu mejor conjetura sobre qué funcionará. Harás cambios durante el proceso de programación, pero es importante tener un buen comienzo.

Para nuestra primera función constructor, convertiremos el Ejemplo 8-9 de antes en el libro. Empezamos por hacer una lista de propiedades del ejemplo:

```javascript var x; var y; var diameter; var speed; ```

El siguiente paso es resolver qué métodos pueden ser útiles para el objeto. Revisando la función draw() del ejemplo que estamos adaptando, vemos dos componentes primarios. La posición de la figura es actualizada y dibujada en la pantalla. Creemos dos métodos para nuestro objeto, uno para cada función:

```javascript function move(); function display(); ```

Ninguno de nuestros métodos retornan un valor. Una vez que hemos determinado las propiedades y métodos que el objeto debería tener, escribiremos nuestra función constructor para asignarlos a cada instancia del objeta que crearemos (Figura 10-1).

El código dentro la función constructor es corrido una vez cuando el objeto es creado. Para crear la función constructor, seguiremos tres pasos:

1. Crear un bloque de función.
2. Añadir las propiedades y asignarles valores.
3. Añadir los métodos.

Primero, creamos un bloque de función para nuestro constructor: ```javascript function JitterBug() { } ```

Observa que el nombre JitterBug empieza con mayúscula. Nombrar la función constructor con letra mayúscula no es requerida, pero es una convención (que recomendamos fuertemente) usada para denotar que es un constructor. (La palabra clave function, sin embargo, debe ser minúscula porque es una regla del lenguaje de programación).

Segundo, añadimos las propiedades. Javascript tiene una palabra reservada especial, this, que puedes usar dentro la función constructor para referirse al objeto actual. Cuando declaras una propiedad de un objeto, dejamos fuera el símbolo var, y en vez de eso anteponemos el nombre de la variable con this. para indicar que estamos asignando una propiedad, una variable del objeto. Podemos declarar y asignar la propiedad speed de la siguiente manera:

```javascript function JitterBug() { this.speed = 0.5; } ```

Mientras estamos haciendo esto, tenemos que decidir qué propiedades tendrán sus valores pasados a través del constructor. Como regla general, los valores de las propiedades que quieran ser diferentes para cada instancia son pasados a través del constructor, y los otros valores de propiedades pueden ser definidos dentro del constructor, como lo es speed en este caso. Para el objeto JitterBug, hemos decidido que los valores de x, y y diámetro serán pasados. Cada uno de los valores pasados es asignado a una variable temporal que existe solo mientras el código es ejecutado. Para clarificar esto, hemos añadido el nombre temp a cada una de estas variables, pero pueden ser nombradas con cualquier nombre que prefieras. Serán usadas solo para asignar los valores de las propiedades que son parte del objeto. Así que añadimos tempX, tempY y tempDiameter como parámetros para la función, y las propiedades son declaradas y asignadas así:

```javascript function JitterBug(tempX, tempY, tempDiameter) { this.x = tempX; this.y = tempY; this.diameter = tempDiameter; this.speed = 0.5; // El mismo valor para cada instancia } ```

El último paso es añadir los métodos. Esto es justo como escribir funciones, pero aquí están contenidas dentro de la función constructor, y la primera línea es escrita un poco diferente. Normalmente, una función para actualizar variables puede ser escrita así:

```javascript function move() { x += random(-speed, speed); y += random(-speed, speed); } ```

Como queremos hacer esta función un método del objeto, nuevamente necesitamos usar la palabra reservada this. La función anterior puede ser convertida en un método así:

```javascript this. move = function() { this.x += random(-this.speed, this.speed); this.y += random(-this.speed, this.speed); }; ```

La primera línea se ve un poco extraña, pero la manera de interpretarla es "crear una variable de instancia (propiedad) llamada move, y luego le asigna como su valor esta función". Luego, cada vez que nos referimos a las propiedades del objeto, podemos nuevamente usar this., tal como lo hacemos cuando están inicialmente declaradas. Juntando todo en el constructor el resultado es este:

```
function JitterBug(tempX, tempY, tempDiameter) {
  this.x = tempX;
  this.y = tempY;
  this.diameter = tempDiameter;
  this.speed = 2.5;

  this.move = function() {
    this.x += random(-this.speed, this.speed);
    this.y += random(-this.speed, this.speed);
  };

  this.display = function() {
    ellipse(this.x, this.y, this.diameter, this.diameter);
  };
}
```

También observa el espaciado en el código. Cada línea dentro del constructor está indentada unos pocos espacios para mostrar lo que está dentro del bloque. Dentro de estos métodos, el código está espaciado nuevamente para mostrar claramente la jerarquía.

Crea objetos

Ahora que has definido una función constructor, para usarla en un programa debes crear un instancia de objeto con ese constructor. Hay dos pasos para crear un objeto:

1. Declara la variable objeto.
2. Crea (inicializa) el objeto con la palabra clave new.

Ejemplo 10-1: haz un objeto

Para hacer tu primer objeto, empezaremos mostrando cómo esto funciona dentro un bosquejo de p5.js y luego continuaremos explicando cada parte en profundidad:

```
var bug;

function setup() {
  createCanvas(480, 120);
  background(204);
```

Crea un objeto y pasa los parámetros

```
  bug = new JitterBug(width/2, height/2, 20);
}

function draw() {
```

```
  bug.move();
  bug.display();
}
```

Copia aquí el código del constructor de Jitterbug

Declaramos variables de objeto en la misma manera que todas las otras variables - el objeto es declarado escribiendo la palabra reservada var seguida del nombre de la variable:

```javascript var bug; ```

El segundo paso es inicializar el objeto con la palabra reservada new. Hace espacio en la memoria para el objeto con todas sus propiedades y métodos. El nombre del constructor es escrito a la derecha de la palabra reservada new, seguido de los parámetros dentro del constructor, si es que tiene alguno:

```javascript bug = new JitterBug(width/2, height/2, 20); ```

Los tres números dentro de los paréntesis son los parámetros pasados dentro de la función constructor JitterBug. El número y orden de estos parámetros deben corresponder con los del constructor.

Ejemplo 10-2: haz múltiples objetos

En el Ejemplo 10-1, vimos algo nuevo: el punto que es usado para acceder a los métodos del objeto dentro de draw(). El operador punto es usado para unir el nombre del objeto con sus propiedades y métodos. Es análogo a la manera en que usamos this. dentro de la función constructor, pero cuando nos referimos a esto fuera del constructor, this es reemplazado por el nombre de la variable.

Esto se torna más claro en este ejemplo, donde dos objetos son hechos con el mismo constructor. La función jit.move() se refiere al método move() que pertenece al objeto nombrado jit, y bug.move() se refiere al método move() y pertenece al objeto llamado bug:

```
var jit;
var bug;

function setup() {
  createCanvas(480, 120);
  background(204);
  jit = new JitterBug(width * 0.33, height/2, 50);
  bug = new JitterBug(width * 0.55, height/2, 10);
}

function draw() {
  jit.move();
  jit.display();
```

```
  bug.move();
  bug.display();
}
```

Copia aquí el código del constructor de JitterBug

Ahora que la función constructor existe como su propio módulo de código, cualquier cambio modificará los objetos hechos con élla. Por ejemplo, podrías añadir una propiedad al constructor JitterBug que controla el color, u otra que determina su tamaño. Estos valores pueden ser pasados usados el constructor o usando métodos adicionales, como setColor() o setSize(). Y como es una unidad auto-contenida, también puedes usar el constructor JitterBug en otro bosquejo.

Ahora es un buen momento para aprender sobre usar múltiples archivos en Javascript. Esparcir tu código en más de un archivo hace que el código más largo sea más fácil de editar y más manejable en general. Un nuevo archivo es usualmente creado para cada constructor, lo que refuerza la modularidad de trabajar con objetos y hacer el código más fácil de encontrar.

Crea un nuevo archivo en el mismo directorio que tu actual archivo sketch.js. Puedes nombrarlo como quieras, pero es una buena idea nombrarlo JitterBug.sjs por conceptos de organización. Mueve la función constructor JitterBug a este nuevo archivo. Enlaza el archivo JitterBug.js en tu archivo HTML añadiendo una líena dentro de HEAD bajo la línea donde enlazas el archivo sketch.js:

```html ```

Robot 8: objetos

Un objeto en software puede combinar métodos (funciones) y propiedades (variables) en una unidad. La función constructor Robot en este ejemplo define todos los objetos robot que serán creados desde él. Cada objeto Robot tiene su propio conjunto de propiedades para almacenar una posicón y la ilustración que dibujará en la pantalla. Cada uno tiene métodos para actualizar la posición y mostrar la ilustración.

Los parámetros para bot1 y bot2 en setup() definen las coordenadas x e y y el archivo .svg que será usado para dibujar el robot. Los parámetros tempX y tempY son pasados al constructor y asignados a las propiedades xpos y ypos. El parámetro imgPath es usado para cargar la ilustración relacionada. Los objetos (bot1 y bot2) dibujan en su propia ubicación con una ilustración difrente porque cada uno tienen valores distintos pasados a los objetos a través de sus constructores:

```
var img1;
var img2;

var bot1;
```
```

```
var bot2;

function preload() {
  img1 = loadImage("robot1.svg");
  img2 = loadImage("robot2.svg");
}

function setup() {
  createCanvas(720, 480);
  bot1 = new Robot(img1,  90, 80);
  bot2 = new Robot(img2, 440, 30);
}

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

Actualiza y muestra el primer robot

```
  bot1.update();
  bot2.display();
```

Actualiza y muestra el segundo robot

```
  bot2.update();
  bot2.display();
}

function Robot(img, tempX, tempY) {
```

Define los valores iniciales para las propiedades

```
  this.xpos = tempX;
  this.ypos = tempY;
  this.angle = random(0, TWO_PI);
  this.botImage = img;
  this.yoffset = 0.0;
```

Actualiza las propiedades

```
  this.update = function() {
    this.angle += 0.05;
    this.yoffset = sin(this.angle) * 20;
  }
```

Dibuja el robot en la pantallla

```
  this.display = function() {
    image(this.botImage, this.xpos, this.ypos + this.yoffset);
  }
}
```

Capítulo 11. Arreglos

Un arreglo es una lista de variables que comparten un nombre común. Los arreglos son útiles porque hacen posible trabajar con más variables sin crear un nombre nuevo para cada uno. Esto hace que el código sea más corto, más fácil de leer, y más conveniente de actualizar.

De variables a arreglos

Cuando un programa necesita mantener registro de una o dos cosas, no es necesario crear un arreglo. Sin embargo, cuando un programa posee muchos elementos (por ejemplo, un campo de estrellas en un juego sobre el espacio o múltiples puntos de datos en una visualización), los arreglos hacen que el código sea más fácil de escribir.

Ejemplo 11-1: muchas variables

Para entender lo que significa, revisa el Ejemplo 8-3. Este código funciona bien si estamos moviendo una sola figura, ¿pero qué pasa si queremos tener dos? Necesitamos crear una nueva variable x y actualizarla dentro de draw():

```
var x1 = -20;
var x2 =  20;

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

function draw() {
  background(0);
  x1 += 0,5;
  x2 += 0.5;
  arc(x1, 30, 40, 40, 0.52, 5.76);
  arc(x2, 90, 40, 40, 0.52, 5.76);
}
```

Ejemplo 11-2: demasiadas variables

El código del ejemplo anterior es todavía manejable, ¿pero qué pasa si queremos tener cinco círculos? Necesitamos añadir tres otras variables a las dos que ya tenemos:

```
var x1 = -10;
var x2 =  10;
var x3 =  35;
```

```
var x4 =  18;
var x5 =  30;

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

function draw() {
  background(0);
  x1 += 0,5;
  x2 += 0.5;
  x3 += 0.5;
  x4 += 0.5;
  x5 += 0.5;
  arc(x1,  20, 20, 20, 0.52, 5.76);
  arc(x2,  40, 20, 20, 0.52, 5.76);
  arc(x3,  60, 20, 20, 0.52, 5.76);
  arc(x4,  80, 20, 20, 0.52, 5.76);
  arc(x5, 100, 20, 20, 0.52, 5.76);
}
```

¡Este código está empezando a salirse de control!

Ejemplo 11-3: arreglos, no variables

Imagina lo que pasaría si quisieras tener 3.000 variables. Esto significaría crear 3.000 variables individuale y luego actualizar cada una de ellas de forma separada. ¿Podrías mantener registro de esta cantidad de variables? ¿Quisieras hacerlo? En vez de eso, usemos un arreglo:

```
var x = [];

function setup() {
  createCanvas(240, 120);
  noStroke();
  fill(255, 200);
  for (var i = 0; i < 3000; i++) {
    x[i] = random(-1000, 200);
  }
}

function draw() {
  background(0);
  for (var i = 0; i < x.length; i ++) {
    x[i] += 0.5;
```

```
    var y = i * 0.4;
    arc(x[i], y, 12, 12, 0.52, 5.76);
  }
}
```

Durante el resto de este capítulo revisaremos los detalles que hacen que este ejemplo sea posible.

Construir un arreglo

Cada item en el arreglo es llamado un elemento, y cada uno tiene un índice para señalar su posición dentro del arreglo. Tal como las coordenadas en el lienzo, los valores de índice para un arreglo empiezan su cuenta desde 0. Por ejemplo, el primer elemento en el arreglo tiene un índice con valor 0, el segundo elemento del arreglo tiene un índice con valor 1, y así. Si hay 20 valores en el arreglo, el valor del índice del último elemento es 19. La Figura 11-1 muestra la estructura conceptual de un arreglo.

Usar arreglos es similar a trabajar con variables únicas, sigue los mismos patrones. Como ya sabes, puedes hacer una variable sola llamada x con este código:

```javascript var x; ```

Para hacer un arreglo, basta con determinar que el valor de la variable sea un par de corchetes vacíos:

```javascript var x = []; ```

Nota que no es necesario declarar por adelantado la longitud del arreglo, la longitud es determinada por el número de elementos que tú pones en él.

Nota

Un arreglo puede almacenar todos los diferentes tipos de datos (boolean, number, string, etc). Puedes mezclar y combinar diferentes tipos de datos en un mismo arreglo.

Antes de que nos adelantemos, desaceleremos y hablemos del trabajo con arreglos en mayor detalle. Hay dos pasos cuando se trabaja con arreglos:

1. Declara el arreglo.
2. Asigna valores a cada elemento.

Cada paso puede pasar en una línea distinta, o se pueden combinar los dos pasos en uno. Cada uno de los dos siguientes ejemplos muestra una técnica diferente para crear un arreglo llamado x que almacena dos números, 12 y 2. Toma mucha atención a lo que pasa antes de setup() y lo que pasa en setup().

Ejemplo 11-4: declara y asigna un arreglo

Primero, declararemos un arreglo fuera de setup() y luego creamos y asignamos valores dentro. La sintaxis x[0] se refiere al primer elemento del arreglo y x[1] es el segundo:

```
var x = [];   // Declara el arreglo

function setup() {
  createCanvas(200, 200);
  x[0] = 12;  // Asigna el primer valor
  x[1] = 2;   // Asigna el segundo valor
}
```

Ejemplo 11-5: asigna valores en un arreglo en una pasada

También puedes asignar valores a un arreglo cuando es creado, si todo es parte de la misma declaración:

```
var x = [12, 2];   // Declara y asigna valores en el arreglo

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

Nota

Evita crear arreglos dentro de draw(), porque crear un nuevo arreglo en cada cuadro desacelerará la tasa de cuadros.

Ejemplo 11-6: revisitando el primer ejemplo

Como un ejemplo completo de cómo usar arreglos, hemos reescrito el Ejemplo 11-1 aquí. A pesar de que no vemos todavía todos los beneficios revelados en el Ejemplo 11-3, sí vemos algunos detalles importantes de cómo funcionan los arreglos:

```
var x = [-20, 20];

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

function draw() {
```

```
  background(0);
  x[0] += 0.5; // Incrementa el primer elemento
  x[1] += 0.5; // Incrementa el segundo elemento
  arc(x[0], 30, 40, 40, 0.52, 5.76);
  arc(x[1], 90, 40, 40, 0.52, 5.76);
}
```

Repetición y arreglos

El for loop, introducido en "Repetición", hará más fácil trabajar con arreglos grandes mientras se mantiene el código conciso. La idea es escribir un loop para recorrer cada elemento del arreglo uno a uno. Para hacerlo, necesitarás saber el largo del arreglo. La propiedad length asocidada con cada arreglo almacena el número de elementos. Usamos el nombre del arreglo con el operador punto para acceder a este valor. Por ejemplo:

```javascript var x = [12, 20]; // Declara y asigna valores al arreglo print(x.length); // Imprime 2 en la consola var y = ["cat", 10, false, 50]; // Declara y asigna valores al arreglo print(y.length); // Imprime 4 en la consola var z = []; // Declara un arreglo vacío print(z.length); // Imprime 0 en la consola z[0] = 20; // Asigna un elemento al arreglo print(z.length); // Imprime 1 en la consola z[1] = 4; // Asigna un elemento al arreglo print(z.length); // Imprime 2 en la consola ```

Ejemplo 11-7: llenando un arreglo con un for loop

Un for loop puede ser usado para llenar un arreglo con valores, o para leer los valores. En este ejemplo, el arreglo es primero llenado con números aleatorios dentro de setup(), y luego esos números son usados para definir el valor del trazado dentro de draw(). Cada vez que el programa corre, un nuevo conjunto de valores aleatorios es puesto en el arreglo:

```
var gray = [];

function setup() {
  createCanvas(240, 120);
  for (var i = 0; i < width; i++) {
    gray[i] = random(0, 255);
  }
}

function draw() {
  background(204);
  for (var i = 0; i < gray.length; i++) {
    stroke(gray[i]);
    line(i, 0, i, height);
  }
}
```
```

En setup(), insertamos tantos elementos como el ancho del lienzo. Esto es un número arbitrario, lo hemos elegido para que al dibujar una línea vertical por cada elemento, se llene el ancho del lienzo. Podrías tratar de cambiar width a cualquier número. Una vez que los elementos son asignados al arreglo, somos capaces de iterar a través de ellos en draw() usando la propiedad length. No podemos iterar a través del arreglo en setup() porque antes de que cualquier elemento sea puesto dentro, el largo del arreglo gray es 0.

Ejemplo 11-8: seguir la trayectoria del ratón

En este ejemplo, existen dos arreglos para almacenar la posición del ratón - uno para la coordenada x y uno para la coordenada y. Estos arreglos graban la posición del ratón durante los últimos 60 cuadros. Con cada nuevo cuadro, los valores de las coordenadas x e y más antiguos son removidos y reemplazados con el valor actual de mouseX y mouseY. Los nuevos valores son añadidos a la primera posición del arreglo, pero antes de que esto pase, cada valor del arreglo es movido una posición a la derecha (desde el último hasta el primero) para hacer espacio para los nuevos números. (La Figura 11-2 es un diagrama que ilustra este proceso). Este ejemplo visualiza esta acción. Además, en cada cuadro, las 60 coordenadas son usadas para dibujar una serie de elipses en la pantalla:

```
var num = 60;
var x = [];
var y = [];

function setup() {
  createCanvas(240, 120);
  noStroke();

  for (var i = 0; i < num; i++) {
    x[i] = 0;
    y[i] = 0;
  }
}

function draw() {
  background(0);
```

Copia los valores del arreglo de atrás hacia adelante

```
  for (var i = num-1; i > 0; i--) {
    x[i] = x[i - 1];
    y[i] = y[i - 1];
  }
  x[0] = mouseX;  // Define el primer elemento
  y[0] = mouseY;  // Define el primer elemento
  for (var i = 0; i < num; i++) {
    fill(i * 4);
    ellipse(x[i], y[i], 40, 40);
```

```
  }
}
```

Arreglos de objetos

Los dos ejemplos cortos en esta sección juntan cada gran concepto de programación en este libro: iteración, condicionales, funciones, objetos y arreglos. Hacer un arreglo de objetos es casi lo mismo que construir los arreglos que introducimos en las páginas anteriores, pero existe una consideración inicial: porque cada elemento es un objeto, primero debe ser creado con la palabra reservada new (tal como cualquier otro objeto) antes de ser asignado a un arreglo. Con un objeto construido por el usuario como JitterBug (ver Capítulo 10), esto significa que new definirá cada elemento antes de ser asignado al arreglo.

Ejemplo 11-9: administrando varios objetos

Este ejemplo crea un arreglo de 33 objetos JitterBug y luego actualiza y muestra cada uno dentro de draw(). Para que este ejemplo funcione, necesitas añadir la función constructor JitterBug al código:

```
var bugs = [];

function setup() {
  createCanvas(240, 120);
  background(204);
  for (var i = 0; i < 33, i++) {
    var x = random(width);
    var y = random(height);
    var r = i + 2;
    bugs[i] = new JitterBug(x, y, r);
  }
}

function draw() {
  for (var i = 0; i < bugs.length; i++) {
    bugs[i].move();
    bugs[i].display();
  }
}
```

Copia aquí el código de la función constructor de Jitterbug

El ejemplo final de arreglos carga una secuencia y almacena cada elemento dentro de un arreglo.

Ejemplo 11-10: secuencias de imágenes

Para correr este ejemplo, obtén las imágenes desde media.zip tal como lo descrito en el Capítulo 7. Las imágenes son nombradas secuencialmente (frame-0000.png, frame-0001.png, etc.), lo que hace posile crear el nombre de cada archivo dentro de un for loop, tal como lo vemos en la séptima línea del programa:

```
var numFrames = 12;     // El número de cuadros
var images = [];        // Crear el arreglo
var currentFrame = 0;

function preload() {
  for (var i = 0; i < numFrames; i++) {
    var imageName = "frame-" + nf(i, 4) + ".png";
    images[i] = loadImage(imageName); //Carga cada imagen
  }
}

function setup() {
  createCanvas(240, 120);
  frameRate(24);
}

function draw() {
  image(images[currentFrame], 0, 0);
  currentFrame++;  // Siguiente cuadro
  if (currentFrame == images.length) {
    currentFrame = 0; // Retorna al primer cuadro
  }
}
```

La función nf() define el formato de números de modo que nf(1, 4) retorna el string "0001" y nf(11, 4) retorna "0011". Estos valores están concatenados con el inicio del nombre del archivo (frame-) y el final (.png) para crear el nombre completo del archivo almacenado en una variable. Los archivos son cargados en un arreglo en la siguiente línea. Las imágenes son mostradas en la pantalla una a la vez en draw(). Cuando la última imagen en el arreglo es mostrada, el programa vuelve al principio del arreglo y muestra las imágenes en secuencia.

Robot 9: arreglos

Los arreglos hacen más fácil que un programa trabaje con muchos elementos. En este ejemplo, un arreglo de objetos Robot es declarado al principio. El arreglo es luego posicionado dentro de setup(), y cada objeto Robot es creado dentro del for loop. En draw(), otro for loop es usado para actualizar y mostrar cada elemento del arreglo bots.

El for loop y un arreglo hacen una combinación poderosa. Observa las diferencias sutiles entre el código para este ejemplo y Robot 8 (ver "Robot 8: objetos") en contraste a los cambios extremos en el resultado visual. Una vez que el arreglo es creado y un for loop es incluido, es igualmente de fácil trabajar con 3 elementos que con 3,000.

```
var robotImage;
var bots = [];    // Declara un arreglo para almacenar objetos Robot

function preload() {
  robotImage = loadImage("robot1.svg");
}

function setup() {
  createCanvas(720, 480);

  var numRobots = 20;
```

Crea cada objeto

```
  for (var i = 0; i < numRobots; i++) {
```

Crea una coordenada x aleatoria

```
    var x = random(-40, width - 40);
```

Asigna la coordenada y basada en el orden

```
    var y = map(i, 0, numRobots, -100, height - 200);
    bots[i] = new Robot(robotImage, x, y);
  }
}

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

Actualiza y muestra cada bot en el arreglo

```
  for (var i = 0; i < bots.length; i++) {
    bots[i].update();
    bots[i].display();
  }
}

function Robot(img, tempX, tempY) {
```

Define los valores iniciales para las propiedades

```
  this.xpos = tempX;
  this.ypos = tempY;
  this.angle = random(0, TWO_PI);
  this.botImage = img;
  this.yoffset = 0.0;
```

Actualiza las propiedades

```
  this.update = function() {
    this.angle += 0.05;
    this.yoffset = sin(this.angle) * 20;
  }
```

Dibuja el robot en la pantalla

```
  this.display = function() {
    image(this.botImage, this.xpos, this.ypos + this.yoffset);
  }
}
```

Capítulo 12. Datos

La visualización de datos es una de las áreas más activas en la intersección de la programación y las gráficas y también uno de los usos más populares de p5.js. Este capítulo expando lo que se ha discutido sobre almacenamiento y carga de datos en el libro e introduce más características relevantes a conjuntos de datos que pueden ser usados para visualización.

Existe un gran rango de software que puede producir visualizaciones estándar como gráficos de barras y gráficos de dispersión. Sin embargo, escribir código desde cero para crear visualización nos brinda mayor control sobre el resultado y anima a los usuarios a imaginar, explorar y crear representaciones de datos únicos. Para nosotros, este es el objetivo sobre aprender a programar y usar software como p5.js, y encontramos mucho más interesante que estar limitado por métodos prefabricados o herramientas ya disponibles.

Resumen de datos

Es un buen momento para rebobinar y discutir cómo los datos fueron introducidos a través de este libro. Una variable en un bosquejo de p5.js es usado para almacenar datos. Empezamos con variables primitivas. En este caso, la palabra primitiva significa un solo trozo de información. Por ejemplo, una variable puede almacenar un número o un string.

Un arreglo es creado para almacenar una lista de elementos dentro de un único nombre de variable.En el Ejemplo 11-7 se almacenan cientos de números a usarse para definir el trazado de las líneas. Un objeto es una variable que almacena un conjunto de variables y funciones relacionadas.

Las variables y los objetos pueden ser definidos dentro del código, pero también pueden ser cargados dentro de un archivo en el directorio del bosquejo. Los siguientes ejemplos en este capítulo lo demuestran.

Tablas

Muchos conjuntos de datos son almacenados como filas y columnas (ver Figura 12-1), así que p5.js incluye un objeto tabla para hacer más fácil trabajar con ellos. Si haz trabajado con hojas de cálculo, tienes una ventaja al momento de trabajar con tablas al programar. p5.js puede leer una tabla desde un archivo, o crear una nueva directamente en código. También es posible leer y escribir cualquier fila y columna y modificar celdas individuales dentro de la tabla. En este capítulo, nos enfocaremos en trabajar con datos en tablas.

Los datos en tablas son usualmente almacenados en archivos de texto planos con las columnas usando comas o tabulación. Un archivo de valores separados por comas es abreviado como CSV y usa la extensión de archivo .csv. Cuando se usa tabulación, la extensión .tsv puede ser usada.

Para crear un archivo CSV o TSV, primero ponlo en el directorio del bosquejo tal como descrito en el comienzo del Capítulo 7, y luego usa la función loadTable() para obtener los datos y ponerlos en un objeto.

Nota

Solo las primeras líneas de cada conjunto de datos son mostradas en estos ejemplos. Si estás escribiendo manualmente el código, necesitarás el archivo .csv, .json o .tsv completo para replicar las visualizaciones mostradas en las figuras. Puedes encontrarlos en el archivo media.zip.

Los datos para el siguiente ejemplo son una versión simplificada de las estadísticas del jugador David Ortiz del equoipo Red Sox. De izquierda a derecha, está el año, el número de home runs, runs batted in (RBI), y el promedio de bateo. Cuando el archivo es abierto en un editor de texto, las primeras cinco líneas del código se ven así:

Ejemplo 12-1: lee la tabla

Para cargar los datos a p5.js, un objeto es creado usando el constructor p5.Table. El objeto en este ejemplo es llamado stats. La función loadTable() carga el archivo ortiz.csv desde el directorio de tu bosquejo. La función se encuentra dentro de preload() para asegurar que esté completamente cargada antes de que los datos sean usados en setup().

En setup(), el for loop lee cada fila de la tabla en secuencia. Toma los datos desde la tabla y los graba en variables. El método getRowCount() es usado para contar el número de filas en cada archivo de datos. Como los datos sobre las estadísticas de Ortiz van de 1997 a 2014, hay 18 filas que leer:

```
var stats;

function preload() {
  stats = loadTable("ortiz.csv");
}

function setup() {
  for (var i = 0; i < stats.getRowCount(); i++) {
```

Recupera el valor de la fila i, columna 0 del archivo

```
    var year = stats.get(i, 0);
```

Recupera el valor de la fila i, columna 1 del archivo

```
    var homeRuns = stats.get(i, 1);
    var rbi = stats.get(i, 2);
    var average = stats.get(i, 3);
```

```
    print(year, homeRuns, rbi, average);
  }
}
```

Dentro del for loop, el método get() es usado para tomar datos desde la tabla. Este método tiene dos parámetros: el primero es la fila a leer y el segundo es la columna.

Ejemplo 12-2: dibuja la tabla

El siguiente ejemplo expande el anterior. Crea un arreglo llamado homeRuns para grabar los datos después de ser cargados dentro de setup() y los datos del arreglo son usados dentro de draw(). El largo del arreglo es usado dos veces con el código homeRuns.length, para contar el número de iteraciones de un for loop. Es usado por primera vez para poner una marca vertical por cada item en el arreglo. Luego es usado para leer cada elemento del arreglo uno por uno y parar de leer del arreglo cuando termina. Después de que los datos son cargados dentro de preload() y leídos desde el arreglo en setup(), el resto de este programa aplica lo que aprendimos en el Capítulo 11. La función getNum() es usada en vez de get() para asegurar que el valor es entendido como un número a usar en el gráfico.

Este ejemplo es la visualización de una versión simplificada de las estadísticas de bateo entre 1997 y 2014 del jugador David Ortiz de los Boston Red Sox, dibujados con los datos de una tabla:

```
var stats;
var homeRuns = [];

function preload() {
  stats = loadTable("ortiz.csv");
}

function setup() {
  createCanvas(480, 120);
  var rowCount = stats.getRowCount();
  homeRuns = [];
  for (var i = 0; i < rowCount; i++) {
    homeRuns[i] = stats.getNum(i, 1);
  }
}

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

Dibuja la matriz de fondo para los datos

```
  stroke(153);
  line(20, 100,  20,  20);
```

```
  line(20, 100, 460, 100);
  for (var i = 0; i < homeRuns.length; i++) {
    var x = map(i, 0, homeRuns.length-1, 20, 460);
    line(x, 20, x, 100);
  }
```

Dibuja líneas basadas en los datos sobre home run

```
  noFill(0);
  stroke(0);
  beginShape();
  for (var i = 0; i < homeRuns.length; i++) {
    var x = map(i, 0, homeRuns.length - 1, 20, 460);
    var y = map(homeRuns[i], 0, 60, 100, 20);
    vertex(x, y);
  }
  endShape();
}
```

Este ejemplo es tan minimalista que no es necesario guardar estos datos en un arreglo, pero la idea puede ser aplicada a ejemplos más complejos que podrías querer hacer en el futuro. Adicionalmente, puedes ver cómo este ejemplo puede ser aumentado con más información - por ejemplo, la información en el eje vertical puede definir el número de home runs y en el horizontal definir el año.

Ejemplo 12-3: 29.740 ciudades

Para tener una mejor idea del potencial de trabajar con tablas de datos, el siguiente ejemplo usa un conjunto de datos más grande e introduce una característica conveniente. Esta tabla de datos es diferente, porque la primera fila, la primera línea en el archivo, es un encabezado. El encabezado define una etiqueta para cada columna para clarificar el contexto. Estas son las primeras cinco líneas de nuestro archivo de datos llamado cities.csv.

``` zip, state, city, lat, lng 35004, AL, Acmar, 33.584132, -86.51557 35005, AL, Adamsville, 33.588437, -86.959727 35006, AL, Adger, 33.434277, -87.167455 35007, AL, Keystone, 33.236868, -86.812861 ```

El encabezado hace más fácil leer el código - por ejemplo, la segunda línea del archivo establece que el código zip de Acmar, Alabama es 35004 y define la latitud de la ciudad como 33.584132 y la longitud como -86.51557. En total, el archivo tiene un largo de 29.741 líneas y define la ubicación y los códigos zip de 29.740 ciudades en Estados Unidos. El siguiente ejemplo carga estos datos dentro la función preload() y luego los dibuja en la pantalla con un for loop dentro de draw(). La función setXY() convierte la latitud y la longitud de un archivo en una elipse en la pantalla:
```

```
var cities;

function preload() {
  cities = loadTable("cities.csv", "header");
}

function setup() {
  createCanvas(480, 240);
  fill(255, 150);
  noStroke();
}

function draw() {
  background(0);
  var xoffset = map(mouseX, 0, width, -width * 3, -width);
  translate(xoffset, -600);
  scale(10);
  for (var i = 0; i < cities.getRowCount(); i++) {
    var latitude = cities.getNum(i, "lat");
    var longitude = cities.getNum(i, "lng");
    setXY(latitude, longitude);
  }
}

function setXY(lat, lng) {
  var x = map(lng, -180, 180, 0,  width);
  var y = map(lat,   90, -90, 0, height);
  ellipse(x, y, 0.25, 0.25);
}
```

Dentro de la función preload(), observa que hay un segundo parámetro "header" añadido a la función loadTable(). Si esto no es hecho, entonces el código tratará la primera línea como datos y no como el título de cada columna. p5.Table tiene docenas de métodos para lograr añadir y remover columnas y filas, obtener una lista de entradas únicas en una columna, o ordenar la tabla. Una lista más completa de métodos y ejemplos cortos son incluidos en la Referencia de p5.js.

JSON

El formato JSON (JavaScript Object Notation) es otro sistema común para almacenar datos. Tal como los formatos HTML y XML, los elementos tienen etiquetas asociadas a ellos. Por ejemplo, los datos de una película pueden incluir etiquetas para el título, director, año de lanzamiento, calificación y más. Estas etiquetas serán emparejadas con datos de la siguiente manera:

```JSON "title": "Alphaville" "director": "Jean-Luc Godard" "year": 1964 "rating": 9.1 ```
```

Para funcionar como un archivo JOSN, necesida un poco de puntuación para separar los elementos. Se usan comas entre cada par de datos y llaves para encapsularlo. Los datos definidos dentro de las llaves son un objeto JSON. Con estos cambios nuestro archivo de datos JSON con formato válido luce así:

```JSON { "title": "Alphaville", "director": "Jean-Luc Godard", "year": 1964, "rating": 9.1 } ```

Existe otro detalle interesante en este corto ejemplo JSON relacionado a los tipos de datos: te darás cuenta que los datos de título y director están encerrados en comillas para demarcarlos como datos de tipo string, mientras que el año y la calificación no tienen comillas para definirlos como números. Esta distinción se hace importante después de cargar los datos a un bosquejo.

Para añadir otra película a la lista, se usa un par de corchetes en el comienzo y el final del archivo JSON para significar que los datos son un arreglo de objetos JSON. Cada objeto es separado por una coma. Poniendo todo esto en práctica, luce así:

```JSON [{ "title": "Alphaville", "director": "Jean-Luc Godard", "year": 1965, "rating": 9.1 }, { "title": "Pierrot le Fou", "director": "Jean-Luc Godard", "year": 1965, "rating": 7.3 }] ```

Este patrón puede ser repetido para incluir muchas películas. En este punto, es interesante comparar esta notación JSON a la representación de tabla correspondiente a los mismos datos. Como un archivo CSV, los datos hubieran lucido así:

title, director, year, rating Alphaville, Jean-Luc Godard, 1965, 9.1 Pierrot le Fou, Jean-Luc Godard, 1965, 7.3

Observa que la notación CSV usa menos caracteres, lo que puede ser importante al momento de trabajar con conjuntos enormes de datos. Por otro lado, la versión JSON es usualmente más fácil de leer porque cada trozo de información está etiquetado.

Ahora que se han introducido las nociones básicos de JSON y su relación a tablas, revisemos el código necesario para leer un archivo JSON en un bosquejo de p5.js.

Ejemplo 12-4: leer un archivo JSON

Este bosquejo carga el archivo JSON visto al principio de esta sección, el archivo que incluye solo los datos de la película Alpaville:

```
var film;

function preload() {
  film = loadJSON("film.json");
}

function setup() {
  var title = film.title;
```

```
  var dir = film.director;
  var year = film.year;
  var rating = film.rating;
  print(title + " by " + dir + ", " + year + ". Rating: " + rating);
}
```

Los datos del archivo son cargados a la variable. Los valores individuales pueden ser accesados usando el operador punto, de forma similar a como accedemos a las propiedades dentro de un objeto.

Ejemplo 12-5: visualiza datos desde un archivo JSON

También podemos trabajar con un archivo JSON que contiene más de una película. Aquí, el archivo de datos comenzado en el ejemplo previo ha sido actualizado para incluir todas las películas del director entre 1960 y 1966. El nombre de cada película es posicionado en orden en la pantalla según el año de lanzamiento y es asignado un valor en la escala de grises según su calificación

.

Existen varias diferencias entre este ejemplo y el Ejemplo 12-4. La más importante es la manera en que el archivo JSON es cargado en los objetos FIlm. El archivo JSON es precargado dentro de preload(), poblando la variable filmData con un arreglo que tiene la misma estructura que los datos en el archivo. En setup(), un for loop es usado para iterar a través del arreglo de la información de las películas y crear un objeto basado en cada elemento en el arreglo, usando el constructor Film deinido aquí. El constructor accede a porciones de la información y los asigna a propiedades dentro de cada objeto. El constructor Film también define un método para mostrar el nombre de la película:

```
var films = [];
var filmData;

function preload() {
  filmData = loadJSON("films.json");
}

function setup() {
  createCanvas(480, 120);
  for (var i = 0; i < filmData.length; i++) {
    var o = filmData[i];
    films[i] = new Film(o);
  }
  noStroke();
}

function draw() {
```

```
  background(0);
  for (var i = 0; i < films.length; i++) {
    var x = i * 32 + 32;
    films[i].display(x, 105);
  }
}

function Film(f) {
  this.title = f.title;
  this.director = f.director;
  this.year = f.year;
  this.rating = f.rating;

  this.display = function(x,y) {
    var ratingGray = map(this.rating, 6.5, 81, 102, 255);
    push();
    translate(x, y);
    rotate(QUARTER_PI);
    fill(ratingGray);
    text(this.title, 0, 0);
    pop();
  }
}
```

Este ejemplo es crudo en su visualización de datos sobre películas, Muestra cómo cargar los datos y cómo dibujar basándose en esos valores de los datos, pero es tu desafío encontrar el formato que acentúe lo que encuentras interesante sobre los datos. Por ejemplo, ¿es más interesante mostrar el número de películas que hizo Godard cada año?, ¿es más interesante comprar y contrastar estos datos con las películas de otro director?, ¿sería más fácil de leer con una fuente, tamaño del bosquejo o razón de aspecto distintos?. Las habilidades introducidas en los capítulos anteriores de este libro pueden ser aplicadas para llevar este bosquejo al siguiente paso de refinamiento.

Datos de redes y APIs

El acceso público a cantidades gigantes de datos recolectados por gobiernos, corporaciones, organizaciones e individuos está cambiando nuestra cultura, desde la manera en que socializamos hasta cómo pensamos osbre ideas intangibles como la privacidad. Estos datos son muy frecuentemente accesados a través de estructuras de software lllamadas APIs.

El acrónimo API es misterioso y su signifciad - interfaz de programación de aplicaciones - no es mucho más clara. Sin embargo, las APIs son esenciales para trabajar con datos y no son necesariamente difíciles de comprender. Esencialmente, son requerimientos de datos hechos a un servicio. Cuando los conjuntos de datos son enormes, no es práctico ni deseable copiar la totalidad de los datos. Una API le permite a un programador pedir solo el goteo de datos que es relevante de este océano masivo.

El concepto puede ser ilustrado de manera más clara con un ejemplo hipotético. Asumamos que existe una organización que mantiene una base de datos de rangos de temperatura para cada ciudad dentro de un país. La API para este conjunto de datos le permite a un programador pedir las temperaturas máxima y mínimas de cualquier ciudad durante el mes de octubre del año 1972. Para requerir estos datos, la petición debe ser hecha mediante una específica línea o líneas de código, en el formato requerido por el servicio de datos.

Algunas APIs son completamente públicas, pero muchas requieren autenticación, que corresponde típicamente a una identidad (ID) de usuario o llave para que el servicio de datos pueda mantener registro de sus usuarios. La mayor parte de las APIs tiene reglas sobre cuántos y cuán frecuentemente se pueden hacer peticiones. Por ejemplo, podría ser posible hacer solo 1.000 requerimientos al mes, o no más de una petición por segundo.

p5.js puede solictar datos en Internet cuando el computador está corriendo el programa en línea. Los archivos CSV, TSV, JSON y XML correspondientes pueden ser cargados usando la función correspondiente de cargado usando la URL como parámetro. Por ejemplo, el clima actual en Cincinnati está disponible en formato JSON en esta URL: http://api.openweathermap.org/data/2.5/find?q=Cincinnati&units=imperial.

Lee la URL detenidamente para decodificarla:

1. Pide datos al subdominio api del sitio openweathermap.org.
2. Especifica una ciudad a buscar (q es abreviación de query, búsqueda, lo que frecuentemente es usado en URLs para especificar búsquedas).
3. También indica que los datos deben ser retornados en formato imperial, lo que significa que la temperatura estará en Fahrenheit. Reemplazando imperial con metric se obtendrá la temperatura en grados Celsius.

Revisar estos datos de OpenWeatherMap es un ejemplo más realístico de trabajar con datos encontrados que los conjuntos de datos simplificados anteriores. Al momento de escribir, el archivo que retornaba la URL era el siguiente:

Este archivo es mucho más fácil de leer cuando es retornado con saltos de línea, y el objeto JSON y las estructuras del arreglo están definidas con llaves y corchetes:

JSON { "message": "accurate", "count": 1, "cod": "200", "list": [{ "clouds": {"all": 80}, "dt": 1423501526, "coord": { "lon": -84.456886, "lat": 39.161999 }, "id": 4508722, "wind": { "speed": 9.48, "deg": 354.002 }, "sys": {"country": "US"}, "name": "Cincinati", "weather": [{ "id": 803, "icon": "04d", "description": "broken clouds", "main": "Clouds" }], "main": { "humidity": 77, "pressure": 999.98, "temp_max": 34.16, "sea_level": 1028.34, "temp_min": 34.16, "temp": 34.16, "grnd_level": 999.98 } }] } ```

Observa que los corchetes en las secciones de "list" y "weather", indican una sección de objetos JSON. Aunque el arreglo en este ejemplo solo contiene un solo ítem, en otros casos, la API podría retornar diversos días o varaciones de múltiples estaciones metereolóigicas.

Ejemplo 12-6: procesando la información del tiempo

El primer paso en trabajar con estos datos es estudiarlos y luego escribir un poco de código para extraer los datos deseados. En este caso, estamos curiosos sobre la temperatura actual. Podemos ver que nuestro dato de temperatura es de 34.16. Está marcado como temp y está dentro del objeto main y está adentro del arreglo list. Una función llamada getTemp() fue escrito específicamente para que este código funcione con el formato de organización de archivos JSON.

```
var weatherData;

function preload() {
  var temp = getTemp(weatherData);
  print(temp);
}

function setup() {
  var temp = getTemp(weatherData);
  print(temp);
}

function getTemp(data) {
  var list = data.list;
  var item = list[0];
  var main = item.main;
  var t = main.temp;
  return t;
}
```

Los datos del archivo JSON son cargados en preload() y son pasados a la función getTemp() dentro de setup(). Luego, por el formato del archivo JSON, una serie de variables es usada para llegar más y más profundo en la estructura de los datos para finalmente llegar al número que deseamos. Este número es almacenado en la variable temperature y luego retornada por la función para ser asignada a la variable temp en setup() donde es impresa en la consola.

Ejemplo 12-7: concatenando métodos

La secuencia de variables JSON creada en sucesión en el ejemplo anterior puede ser aproximada de forma distinta usando accesores. Este ejemplo funciona como el Ejemplo 12-6, pero los métodos son conectadas con el operador punto, en vez de ser calculados uno a la vez y asignados a variables entre medio:

```
var weatherData;

function preload() {
```

```
  weatherData = loadJSON("cincinnati.json");
}

function setup() {
  var temp = getTemp(weatherData);
  print(temp);
}

function getTemp(data) {
  return data.list[0].main.temp;
}
```

Además nota cómo la temperatura final es retornada por la función getTemp(). En el Ejemplo 12-6, una variable es creada para almacenar el valor, luego ese valor es retornado. Aquí, el valor de la temperatura es retornado directamente, sin una variable intermedia.

Este ejemplo puede ser modificado para acceder a más datos de la organización y para construir un bosquejo que muestre los datos en la pantalla en vez de solo escribirlos en la consola. También puedes modificarlo para que lea datos de otra API - te encontrarás con que los datos retornados por muchas APIs comparten un formato similar.

Robot 10: datos

El ejemplo final de robot en este libro es diferente del resto porque tiene dos partes. La primera parte genera un archivo usando valores aleatorios y for loops y la segunda parte lee ese archivo de datos para dibujar un ejército de robots en la pantalla.

El primer bosquejo usa dos nuevos elementos de código, la clase PrintWriter y la función createWriter(). Usadas en conjunto, crean y abren un archivo en el directorio del bosquejo para almacenar los datos generados por el bosquejo. En este ejemplo, el objeto creado con PrintWriter es llamado output y el archivo es llamado botArmy.tsv. En los loops, los datos son escritos al archivo corriendo el método println() en el objeto de salida. Aquí, los valores aleatorios son usados para definir cuál de las tres imágenes del robot serán dibujadas para cada coordenada. Para que el archivo se cree correctamente, el método close() debe ser ejecutado antes de que el programa se detenga. El código que dibuja una elipse es un adelanto visual para revelar la posición de la coordenada en la pantalla, pero noten que la elipse no está grabada en el archivo:

```
var output;

function setup() {
  createCanvas(720, 480);
```

Crea el nuevo archivo

```
  output = createWriter("botArmy.tsv");
```

Escribe una línea de encabezado con los títulos de las columnas

```
  output.println("type\tx\ty");
  for (var y = 0; y <= height;="" y="" +="60);" {="" for="" (var="" x="0;" <='
```

Después que el programa es corrido, el archivo botArmy.tsv será creado en el directorio del bosquejo. Ábrelo para revisar cómo se escriben los datos. Las primeras líneas de código de este archivo serán similares a esto:

La primera columna es esencial para definir qué imagen de robot se va a usar, la segunda columna es la coordenada x, la tercera columna es la coordenada y. El siguiente bosquejo carga nuestro archivo botArmy.tsv y usa los datos para estos propósitos:

```
var robots;
var bot1;
var bot2;
var bot3;

function preload() {
  bot1 = loadImage("robot1.png");
  bot2 = loadImage("robot2.png");
  bot3 = loadImage("robot3.png");
  robots = loadTable("botArmy.tsv", "header");
}

function setup() {
  createCanvas(720, 480);
  imageMode(CENTER);
  for (var i = 0; i < robots.getRowCount(); i++) {
    var bot = robots.getNum(i, "type");
    var x = robots.getNum(i, "x");
    var y = robots.getNum(i, "y");
    var sc = 0.15;
    if (bot == 1) {
      image(bot1, x, y, bot1.width*sc, bot1.height*sc);
    } else if (bot == 2) {
      image(bot2, x, y, bot2.width*sc, bot2.height*sc);
    } else {
      image(bot3, x, y, bot3.width*sc, bot3.height*sc);
    }
  }
}
```

Una variación más concisa (y flexible) de este bosquejo usa arreglos como un enfoque más avanzado:

```
var numRobotTypes = 3;
var images = [];
var scaling = 0.15;
var botArmy;

function preload() {
  for (var i = 0; i < numRobotTypes; i++) {
    images[i] = loadImage("robot" + (i+1) + ".png");
  }
  botArmy = loadTable("botArmy.tsv", "header");
}

function setup() {
  createCanvas(720, 480);
  imageMode(CENTER);
  for (var i = 0; i < botArmy.getRowCount(); i++) {
    var robotType = botArmy.getNum(i, "type");
    var x = botArmy.getNum(i, "x");
    var y = botarmy.getNum(i, "y");

    var bot = images[robotType - 1];
    image(bot, x, y, width*scaling, bot.height * scaling);
  }
}
```

Capítulo 13. Extensión

Este libro se enfoca en usar p5.js para hacer gráficas interactivas, porque eso es lo principal que p5.js hace. Sin embargo, este software puede realizar mucho más que esto, y está siendo extendido más allá todo el tiempo.

Una librería de p5.js es un conjunto de código que extiende el software más allá de sus funciones principales. Las librerías han sido una parte importante del crecimiento de este proyecto, porque le permite a los desarrolladores añadir nuevas funciones rápidamente. Como projectos autocontenidos y más pequeños, las librerías son más fáciles de manejar que si sus características estuvieran integradas al software principal.

El archivo .zip completo de p5.js incluye las librerías p5.dom y p5.sound. También puedes bajar otras librerías desde http://p5js.org/libraries/. Para usar una de estas librerías, primero asegúrate de que está dentro del directorio que contiene tus archivos HTML y JavaScript. En segundo lugar, añade una línea de código a tu archivo HTML para indicar que la librería será usada en el bosquejo actual. Esta línea debería verse así:

```html ```

relative/path debería ser reemplazado por la ubicación requerida para ubicar el archivo de librería relativo al archivo HTML. Si necesitas subir un directorio, inserta "..". Por ejemplo, si estás trabajando con el ejemplo vacío de la librería de p5.sound.js de la descarga completa de p5.js complet, la línea ser vería así:

```html ``` ## p5.sound

La librería p5.sound tiene la habilidad de reproducir, analizar y generar (sintetizar) sonido. A continuación se presentan una s pocas funciones clave; consultar la Referencia de p5.js para muchos más objetos que pueden ser creados y funciones que pueden ser llamadas: http://p5js.org/reference/#/libtstird/p5.sound.

Como las imágenes, archivos JSON, y los archivos de texto introducidos en el Capítulo 7, un archivo de audio es otro tipo de medio para aumentar un bosquejo de p5.js. Sigue las instrucciones en ese capítulo para aprender cómo cargar un archivo de sonido en el directorio de un bosquejo. La librería p5.sound puede cargar un gran rango de formatos de archivos de audio incluyendo WAV, AIFF y MP3. Una vez que un archivo es cargado, puede ser reproducido, pausado y repetido así mismo como distorsionado por un grupo de funciones de efecto.

Ejemplo 13-1: reproduce un sample

El uso más común de la librería p5.sound es para tocar un sonido cuando un evento ocurre en la pantalla o como música de fondo. Este ejemplo se basa en el Ejemplo 8-5 para reproducir un sonido cuando la figura llega a los bordes de la pantalla. El archivo blip.wav es incluido en la carpeta media que puede ser descargada siguiendo las instrucciones del capítulo 7. Así

como con otros medios, una variable para almacenar un objeto p5.SoundFile (que es lo que la función loadSound() retorna) es definida en la parte superior del bosquejo, es cargada con preload() y después de eso, puede ser usada en cualquier parte del prorgrama:

```
var blip;

var radius = 120;
var x = 0;
var speed = 1.0;
var direction = 1;

function preload() {
  blip = loadSound("blip.wav");
}

function setup() {
  createCanvas(440, 440);
  ellipseMode(RADIUS);
  x = width/2; // Parte en el centro
}

function draw() {
  background(0);
  x += speed * direction;
  if ((x > width - radius) || (x < radius)) {
    direction = - direction; // Invierte la dirección
    bip,play();
  }
  if (direction == 1) {
    arc(x, 220, radius, radius, 0.52, 5.76);
  } else {
    arc(x, 220, radius, radius, 3.67, 8.9);
  }
}
```

El sonido es gatillado cada vez que el método play() es ejecutado. Este ejemplo funciona bien porque el sonido solo está siendo reproducido cuando el valor de la variable x está en las orillas de la pantalla. Si el sonido fuera tocado en cada ejecución de draw(), el sonido se reiniciaría 60 veces por segundo y no tendría tiempo de terminar de sonar. El resultado sería un sonido distorsionando rápidamente. Para tocar un sample más largo mientras un programa corre, llama a los métodos play(), loop() para que suenen dentro de setup() así que el sonido así se gatilla una sola vez.

Nota

El objeto p5.soundFile tiene muchos métodos para controlar cómo un sonido es reproducido. Lo más esenciales son play() para tocar el sample una vez, loop() para tocarlo desde el principio hasta el fin una y otra vez, stop() para parar la reproducción y jump() para moverse a un momento específico dentro del archivo.

Ejemplo 13-2: escucha un micrófono

Además de reproduicr sonidos, p5.js puede escuchar. Si tu computador tiene un micrófono incorporado o conectado, la librería p5.sound puede leer audio en vivo a través de él. Una vez que los datos del micrófono están conectados al software, los puedes analizar, modificar o reproducir:

```
var mic;
var amp;

var scale = 1.0;

function setup() {
  createCanvas(440, 440);
  background(0);
```

Crea una entrada de audio y empieza a escuchar

```
  mic = new p5.AudioIn();
  mic.start();
```

Crea un nuevo analizador de amplitud y conéctalo a la entrada

```
  amp = new p5.Amplitude();
  amp.setInput(mic);
}

function draw() {
```

Dibuja un fondo que se vaya a negro

```
  noStroke(0);
  fill(0, 10);
  rect(0, 0, width, height);
```

El método getLevel() retorna valores entre 0 y 1 así que map() es usado para convertir los valores a números mayores

```
  scale = map(amp.getLevel(), 0, 1.0, 10, width);
```

Dibuja el círculo basado en el volumen

```
  fill(255);
  ellipse(width/2, height/2, scale, scale);
}
```

Hay dos partes que están tomando la amplitud (volumen) del micrófono añadido. El objeto p5.AudioIn es usado para obtener la señal del micrófono y el objeto p5.Amplitude es usado para medir la señal.

Las variables para almacenar ambos objetos está definidas en la parte de arriba del código y creadas dentro de setup(). Después del objeto p5.Amplitude() (en este programa es llamado amp), el objeto p5.AudioIn, nombrado mic, es conectado al objeto amp con el método setInput(). Después de eso, el método getLevel() del objeto amp puede ser ejecutado en cualquier momento para leer la amplitud del micrófono dentro del programa. En este ejemplo, eso es hecho cada vez en draw() y ese valor es usado para definir el tamaño del círculo.

Además de tocar un sonido y analizar un sonido como fue demostrado en los últimos dos ejemplos, p5.js puede sintetizar sonido directamente. Los fundamentos de la síntesis de sonido son formas de onda, entre las que se incluyen las ondas sinusoidales, triangulares y cuadradas. Una onda sinusoidal suena agradable, una onda cuadrada es más dura y una onda triangular está entre medio. Cada onda tiene un número de propiedades. La frecuencia, medida en Hertz, determina la altura, cuán grave o agudo es el tono. La amplitud de la onda determina el volumen.

Ejemplo 13-3: crear una onda sinusoidales

En el siguiente ejemplo, el valor de mouseX determina la frecuencia de la onda sinusoidal. Mientras el ratón se mueve a la izquierda y derecha, la frecuencia audible y la correspondiente visualización de la onda aumenta y decae:

```
var sine;

var freq = 440;

function setup() {
  createCanvas(440, 440);
```

Crea e inicia el oscilador sinusoidal

```
  sine = new p5.SinOsc();
  sine.start;
}

function draw() {
  background(0);
```

Mapea el valor de mouseX entre 20 Hz y 440 Hz para la frecuencia

```
  var hertz = map(mouseX, 0, width, 20.0, 440.0);
  sine.freq(hertz);
```

Dibuja una onda para visualizar la frecuencia del sonido

```
  stroke(204);
  for (var x = 0; x < width; x++) {
    var angle = map(x, 0, width, 0, TWO_PI * hertz);
    var sinValue = sin(angle) * 120;
    line(x, 0, x, height/2 + sinValue);
  }
}
```

El objeto sine, creado con el constructor de p5.SinOsc, es definido al principio del código y luego creado dentro de setup(). El método start() hace que la onda empieza a generar sonido. Dentro de draw(), el método freq() define continuamente la frecuencia de la onda, basándose en la posición izquierda y derecha del ratón.

p5.dom

La librería p5.dom tiene la habilidad de crear e interactuar con elementos HTML fuera del lienzo gráfico. La sigla DOM viene de Document Object Model, que se refiere a un conjunto de métodos para interactuar programáticamente con la página HTML. Los siguientes ejemplos presentan unas pocas funciones clave. Revisa la Referencia de p5.js para muchos más elementos que pueden ser creados y funciones que pueden ser llamadas: http://p5js.org/reference/#/libraries/p5.dom.

Tal como createCanvas() crea un lienzo gráfico en la página, p5.dom incluye un número de otros métodos create para añadir otros elementos HTML a la página. Ejemplos incluyen video, links URL, cuadros de entrada y barras deslizadoras.

Ejemplo 3-14: accede a la webcam

createCapture() accede a la webcam en tu computador y crea un elemento HTML que muestra su audio y video. Una vez que el elemento de captura es creado, puede ser dibujado en el lienzo y ser manipulado:

```
var capture;

function setup() {
  createCanvas(480, 120);
  capture = createCapture();
  capture.hide();
}

function draw() {
  var aspectRatio = capture.height/capture.width;
```

```
  var h = width * aspectRatio;
  image(capture, 0, 0, width, h);
  filter(INVERT);
}
```

El objeto de capture está definido en la parte superior del código y luego es creado dentro de setup(). createCapture() de hecho adjunta un nuevo elemento a la página, pero como queremos dibujarlo en el lienzo, el método hide() es usado para esconder el objeto de captura. Revisa lo que pasa cuando descomentas esta línea de código.

<

p>Deberías ver dos copias del video, una invertida y una normal.

Los datos del objeto de captura son dibujados en el lienzo en la función draw() e invertidos usando el método filter().

Ejemplo 13-5: crea una barra deslizadora

La función createSlider() crea una barra deslizadora que puede ser usada para manipular aspectos del bosquejo. Acepta tres argumentos - el valor mínimo, el valor máximo y el valor inicial:

```
var slider;

function setup() {
  createCanvas(480, 120);
  slider = createSlider(0, 255, 100);
  slider.position(20, 20);
}

function draw() {
  var gray = slider.value();
  background(gray);
}
```

El objeto barra deslizadora es definido en la parte superior del código y luego creado dentro de setup(). Por defecto, el elemento será adjuntado a la página, justo después del elemento más recientemente creado en la página. El método position() permite darle una posición relativa a la esquina superior izquierda. El método value() retorna el valor actual de la barra deslizadora, el cual está siendo usado para definir el color del fondo del lienzo en draw().

Ejemplo 13-6: crea un recuadro de entrada

La función createInput() añade un recuardo que puede ser usado para darle entrada de texto a tu programa. createButton() añade un botón que puede gatillar cualquier función que

escojas. En este caso, el botón es usado para entregar el texto dentro del recuadro de entrada al programa:

```
var input;
var button;

function setup() {
  createCanvas(480, 120);
  input = createInput();
  input.position(20, 30);
  button = createButton("submit");
  button.position(160, 30);
  button.mousePressed(drawName);

  background(100);
  noStroke();
  text("Enter your name. ", 20, 20);
}

function drawName() {
  background(100);
  textSize(30);
  var name = input.value();
  for (var i = 0; i < 30; i++) {
    text(name, random(width), random(height));
  }
}
```

Los objetos entrada y botón son definidos en la parte superior del código y creados dentro de setup(). createButton() aceptoa un argumento, la etiqueta a ser mostrada en el botón. El método mousePressed() es usado para asignar una función a ejecutar cuando el botón es presionado. Dentro de drawName(), los contenidos del recuadro de entrada son leídos usando el método value(), y usados para llenar el fondo con el texto.