# El uso de p5.js para aprender a programar: figuras y transformaciones

Juan Carlos Ponce Campuzano

SUMA+ núm. 94
pp. 83-96
Artículo solicitado por *Suma* en abril de 2020 y aceptado en junio de 2020

En los últimos 20 años han surgido varios proyectos de acceso libre y gratuito con el objetivo de facilitar el aprendizaje de los elementos esenciales de los lenguajes de programación de manera sencilla. Esto se debe a la gran demanda de programadores que se ha generado como consecuencia del desarrollo acelerado de la tecnología computacional y digital.

Uno de estos proyectos es el de Casey Reas y Ben Fry quienes, en 2001, empezaron a trabajar en una nueva plataforma para hacer más fácil la programación de gráficas interactivas; la cual nombraron Processing. En McCarthy, Reas, y Fry (2018) se narra que Reas y Fry se encontraban frustrados con lo difícil que era escribir software con los lenguajes de programación que se usaban en ese entonces (como C++ y Java) y fueron inspirados por lo simple que era escribir programas interesantes con los lenguajes que habían aprendido en su niñez (Logo y BASIC).

Esencialmente Processing es un lenguaje de programación construido para enseñar programación a estudiantes de diseño y de arte y ofrecer a estudiantes más avanzados una manera más fácil de trabajar con gráficas digitales. Esta combinación es una aproximación diferente a la manera en que usualmente se enseña programación. Con Processing los nuevos usuarios comienzan concentrándose en gráficos e interacción en vez de estructuras de datos y resultados en forma de texto en la consola.

Con el paso del tiempo, Processing se ha transformado en una gran comunidad. De hecho, es usado actualmente en salones de clases a lo largo del mundo, en planes de estudios de artes, humanidades y ciencias de la computación y de forma profesional. Debido su gran popularidad y al desarrollo de nuevos navegadores web, en 2013 Lauren McCarthy creó p5.js una librería de JavaScript (uno de los 5 lenguajes de programación más populares a la fecha de

acuerdo con Carbonnelle 2019) que sigue la sintaxis y las convenciones de Processing con la finalidad de facilitar la transición a la web de los usuarios de la comunidad existente de Processing. Asimismo, p5.js tiene como objetivo facilitar lo más posible a los principiantes a aprender a programar aplicaciones gráficas interactivas (al tiempo que proporciona herramientas poderosas para expertos).

## ¿Qué es p5.js?

p5.js es una librería y un conjunto de herramientas que facilitan el uso del lenguaje de programación JavaScript para la *programación creativa* (creative coding). La ventaja de usar JavaScript es su amplia disponibilidad y su soporte en casi cualquier sistema operativo: cada navegador web tiene incorporado un intérprete de JavaScript, lo que significa que los programas p5.js pueden (en general) ejecutarse en cualquier navegador web. Además, el lenguaje JavaScript está definido por un estándar internacional y la mayoría de los intérpretes de este lenguaje (incluidos los que se ejecutan dentro de los navegadores web), son de código abierto y están disponibles gratuitamente. En otras palabras, JavaScript es de acceso libre y cualquier persona puede usarlo de forma gratuita.

## ¿Para qué sirve p5.js?

Como sugieren McCarthy, Reas y Fry (2018), p5.js sirve para escribir software que produce imágenes, animaciones e interacciones. La motivación es escribir una línea de código y que, por ejemplo, un círculo aparezca en la pantalla. Después, añade unas pocas líneas de código, y ahora el círculo se hace más grande o más pequeño. Una línea más de código, y el círculo cambia de color cuando presionas un botón con el ratón. A esto lo llamamos bosquejar con código. Escribes una línea, luego añades otra, luego otra, y así sucesivamente. El resultado es un programa creado parte por parte. Por esta razón, los programas de p5.js (y Processing) generalmente se denominan «sketches» (bocetos), cuya nomenclatura usaremos de aquí en adelante.

## ¿Cómo comenzamos a escribir un programa de p5.js?

Hay varias formas de escribir sketches de p5.js. Debido a que p5.js es solo una biblioteca de JavaScript, podemos incluirla en cualquier página web normal e incluso usarla junto con otras bibliotecas de JavaScript. El tutorial oficial <https://p5js.org/es/get-started/> sugiere descargar un editor de texto separado (como Brackets o Atom) y trabajar con su código fuente p5.js como cualquier otro lenguaje de programación.

En este artículo, sin embargo, vamos a utilizar el *editor web* <https://editor.p5js.org/>, un entorno de programación basado en la web y creado específicamente para p5.js, el cual es muy sencillo de utilizar y debería funcionar en cualquier navegador web.

## Anatomía del navegador web

Cuando se carga el editor web (figura 1), podemos comenzar a escribir código de inmediato. Es recomendable crear una cuenta, lo cual nos permitirá guardar nuestro trabajo y compartirlo con otras personas. Así es como se ve el editor web y una explicación de las partes más importantes.

1. El panel de «código». Aquí es donde escribes tu código p5.js.
2. El panel de «consola». Aquí es donde aparecen los mensajes de error de tu código. También puedes usar la función `console.log()` para hacer que el texto aparezca aquí mientras tu programa se está ejecutando (esto es útil para fines de depuración, y lo discutiremos más adelante).
3. El panel de «vista previa». Tu sketch aparecerá aquí.
4. Controles: Presiona el botón «Reproducir» (▶) para «ejecutar» tu sketch. Presiona «Stop» (■) para detener el sketch.
5. El nombre de tu sketch. Haz clic en el pequeño icono de lápiz a la derecha del nombre para editarlo (cuando creas un nuevo sketch, se genera un nombre aleatorio).

Cuando abrimos por primera vez el editor web aparece por defecto el siguiente código:

```
function setup() {
createCanvas(400, 400);
}
function draw() {
background(220);
}
```

Todavía no sabemos para qué sirve, pero pronto lo descubriremos. Si oprimimos el botón «Reproducir» para ejecutarlo, observaremos que a la derecha se dibuja un cuadrado gris (figura 2).

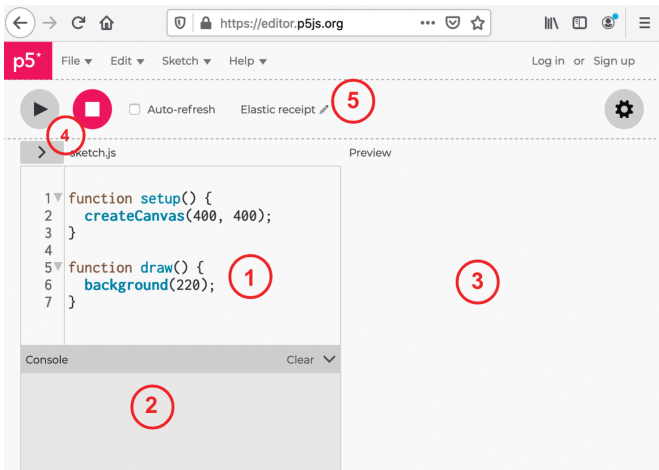



Figura 1. Editor web

La primera pregunta que aquí surge es: ¿Qué está pasando aquí? Para entender un poco lo que sucede primero podemos traducir cada palabra que aparece en el código:

— **function**: función
— **setup**: preparar
— **createCanvas**: crear lienzo
— **draw**: dibujar
— **background**: fondo

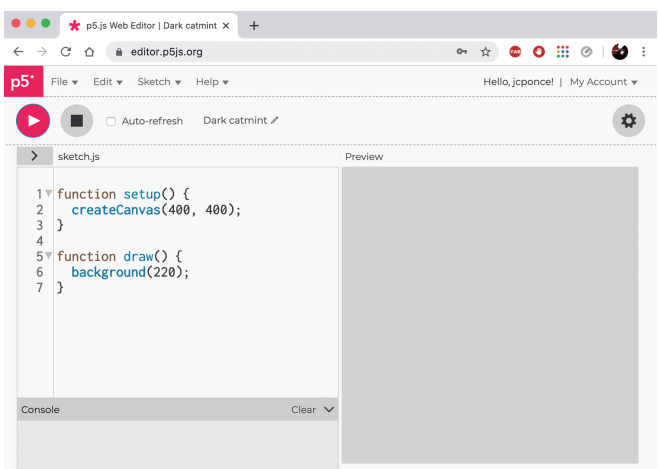



Figura 2. Editor web con lienzo gris

En este caso, la instrucción **function** indica las dos funciones principales que necesitamos para crear un sketch (cabe aclarar que, en este contexto, el término «función» denotará un comando de JavaScript o p5.js que realiza una tarea específica y que en algunos casos puede requerir parámetros como *input*). Si unimos estos dos términos podemos deducir que tenemos dos funciones:

— **setup()**: la cual prepara el programa y crea un lienzo
— **draw()**: la cual dibuja o pinta un fondo en el lienzo

La librería p5.js es muy intuitiva y con un poco de conocimiento de idioma inglés podemos hacernos una idea de lo que hacen todas las instrucciones. En este caso podemos inferir que se ha preparado un espacio para dibujar, un lienzo, en el cual hemos dibujado el fondo con un color (gris).

Aquí surgen un par de preguntas más: ¿Qué hacen exactamente las instrucciones **createCanvas()** y **background()**?, y ¿qué significan los números que aparecen entre paréntesis? Esencialmente, estas son funciones integradas en p5.js, las cuales siempre van seguidas de un par de paréntesis, dentro de estos paréntesis por lo general hay una lista de valores separados por comas. Estos valores se llaman *parámetros* de la función. Cada función usa sus parámetros de una manera ligeramente diferente, y parte del aprendizaje de una función es comprender lo que significan sus parámetros.

— createCanvas(ancho, alto): Esta función crea un lienzo en el documento y define sus dimensiones los parámetros «ancho» y «alto».
— background(valor): Esta función define el color usado como fondo del lienzo. El color se determina con el parámetro «valor» que puede ser un número entre 0 y 255.

Podemos colocar tantas funciones como deseemos dentro de las llaves {...} en la función draw(). No hablaremos sobre lo que significa la función draw() por ahora, o lo que significan las llaves. Solo es necesario saber que podemos poner más cosas entre esas llaves si deseamos que sucedan más cosas.

Los creadores de esta librería han dedicado mucho tiempo y esfuerzo para desarrollar instrucciones que se pueden utilizar de forma intuitiva. Si bien p5.js contiene varias docenas de funciones integradas que realizan diversas tareas, como dibujar figuras en la pantalla o calcular valores utilizando fórmulas matemáticas, aprender a programar en p5.js se trata principalmente de aprender estas funciones y lo que hacen. Se pueden encontrar una gran lista de funciones de p5.js en la página oficial <https://p5js.org/es/>, en la sección de referencia, la cual contiene descripciones en español. Por ahora, nos centraremos solo en las funciones más simples para dibujar figuras geométricas en la pantalla.

## Figuras geométricas y colores

Es momento de dibujar algunas figuras geométricas con p5.js. El código predefinido anteriormente puede resultar un poco aburrido, así que ahora dibujaremos algunas figuras geométricas para darle un poco de vida a nuestro sketch. Para esto escribiremos las siguientes dos líneas dentro de nuestra función draw():

— ellipse(130, 130, 200, 200);
— rect(80, 190, 180, 100);

No es difícil deducir que estas funciones dibujarán una elipse y un rectángulo (figura 3), respectivamente:

```
function setup() {
  createCanvas(400, 400);
}
function draw() {
  background(220);
  ellipse(130, 130, 200, 200);
  rect(80, 190, 180, 100);
}
```

Los parámetros de la función elipse tienen los siguientes significados:

— Parámetro 1: la posición *X* de la esquina superior izquierda del rectángulo
— Parámetro 2: la posición *Y* de la esquina superior izquierda del rectángulo
— Parámetro 3: el ancho de la elipse
— Parámetro 4: la altura de la elipse

Para el caso de la función rectángulo tenemos que:

— Parámetro 1: la posición *X* de la esquina superior izquierda del rectángulo
— Parámetro 2: la posición *Y* de la esquina superior izquierda del rectángulo
— Parámetro 3: el ancho del rectángulo
— Parámetro 4: la altura del rectángulo

Observa que las funciones se ejecutan en el mismo orden en que se dibujan. Esto significa que el rec-

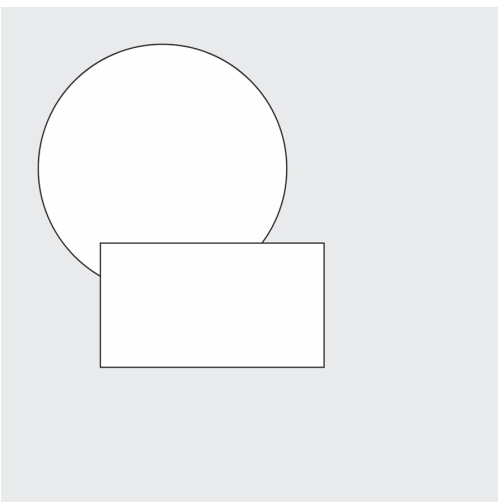


Figura 3. Elipse y rectángulo

tángulo se dibuja encima de la elipse. Si inviertes el orden de las funciones, la elipse se dibujará en la parte superior (figura 4):

```
function setup() {
  createCanvas(400, 400);
}
function draw() {
  background(220);
  rect(80, 190, 180, 100);
  ellipse(130, 130, 200, 200);
}
```

Ahora veamos qué sucede cuando modificamos los parámetros de la función **background()**. Como ya mencioné anteriormente, esta función define el color de fondo del lienzo con un parámetro. El valor de dicho parámetro determina el tono de gris que aparecerá el fondo. 0 significa negro y 255 significa blanco; otros valores se refieren a todos los tonos intermedios de gris. En realidad, podemos introducir tres parámetros en esta función, lo cual nos permitirá definir el color de fondo de nuestra elección, no solamente gris. Los tres parámetros corresponden a las componentes Rojo,Verde y Azul del esquema de coloración definido en los ordenadores (comúnmente llamado *esquema RGB*, las siglas del inglés *Red, Green, Blue*). Por ejemplo, los valores (128, 191, 255), definen un color azul (figura 5):

```
function setup() {
  createCanvas(400, 400);
}
function draw() {
  background(128, 191, 255); rect(80, 190, 180,
  100); ellipse(130, 130, 200, 200);
}
```

## Coordenadas en p5.js

Hemos visto en la sección anterior que los números en las funciones **ellipse()** y **rect()** especifican posiciones y dimensiones. ¿Pero cuáles son las unidades? Si 50 es la posición *X* de alguna figura, ¿cuál es la unidad? En este caso, las distancias en el procesamiento se miden en *píxeles*.

Casi todas las pantallas digitales se dividen en pequeños cuadrados llamados píxeles. Normalmente, cada píxel es en realidad un pequeño dispositivo físico que puede mostrar un color. La pantalla de nuestro dispositivo (ordenador, tableta o móvil) en este momento probablemente tenga cientos de miles, quizá millones, de estos pequeños dispositivos. Cada uno tiene exactamente el mismo tamaño, y juntos forman una «cuadrícula».

Cada sketch de p5.js tiene un tamaño en píxeles: el número de píxeles de ancho y el número de píxeles

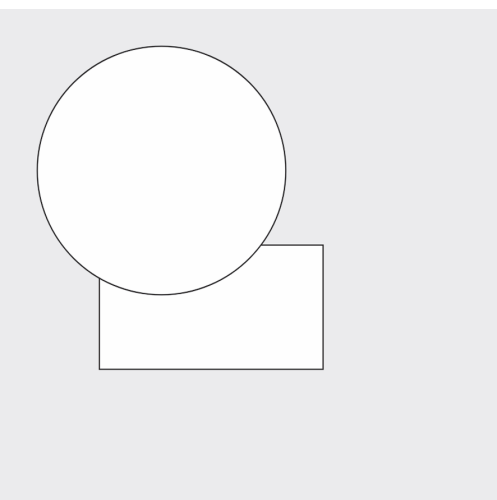


Figura 4. Rectángulo y elipse

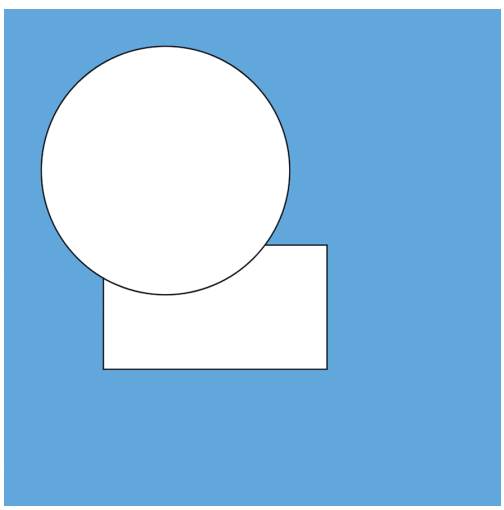


Figura 5. Lienzo azul

de alto. Por defecto, el sketch del editor web es de 400 píxeles por 400 píxeles. El píxel en la esquina superior izquierda se designa como el píxel en la coordenada (0,0). A medida que se mueve más a la derecha, la coordenada *X* aumenta. A medida *que avanza, la coordenada Y* aumenta, de modo que la coordenada del punto medio del sketch es (200,200) y la coordenada en la esquina inferior derecha es (400,400). Si estás familiarizado con las coordenadas cartesianas, esto parece extraño, ya que normalmente el valor Y disminuye a medida que avanza hacia abajo. Afortunadamente no cuesta mucho tiempo acostumbrarse a las coordenadas que se emplean en p5.js (figura 6).

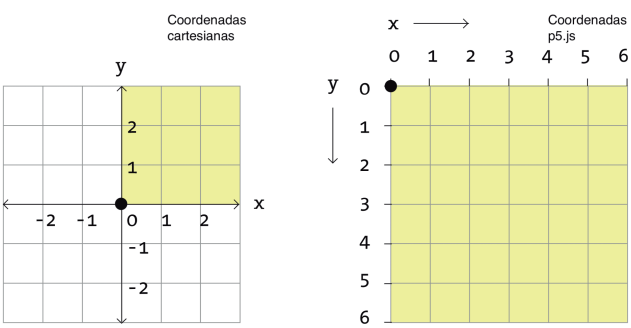



Figura 6. Comparación de sistema de coordenadas

## Relleno y trazo

p5.js nos permite controlar no solo las formas que dibujas, sino también dos aspectos críticos de la apariencia de esas formas: de qué color son y de qué color es su contorno. Para lograrlo podemos usar las funciones:

— **fill()**: Define el color usado para el relleno de figuras geométricas.
— **stroke()**: Define el color usado para dibujar líneas y bordes de figuras geométricas.

Al igual que con **background()**, podemos usar **fill()** y **stroke()** con un solo parámetro para establecer su color de escala de grises, o tres parámetros para establecer un color usando valores RGB.

También hay una función llamada **strokeWeight()** que nos permite establecer el ancho del trazo. El ancho se mide en píxeles y, al igual que **fill()** y **stroke()**, se aplica a todas las funciones que se encuentran después. La figura 7 muestra un ejemplo usando estas funciones:

```
function setup() {
  createCanvas(400, 400);
}
function draw() {
  background(249, 199, 79);
  fill(248, 150, 30);
  strokeWeight(10);
  stroke(243, 114, 44);
  rect(80, 190, 180, 100);
  fill(144, 190, 109);
  strokeWeight(20);
  stroke(87, 117, 144);
  ellipse(130, 130, 200, 200);
  fill(67, 170, 139);
  strokeWeight(25); stroke(249, 65, 68);
  rect(210, 250, 120, 120);
}
```

Por supuesto, podemos crear muchas otras figuras geométricas en p5.js como puntos, líneas, arcos, triángulos con las funciones:

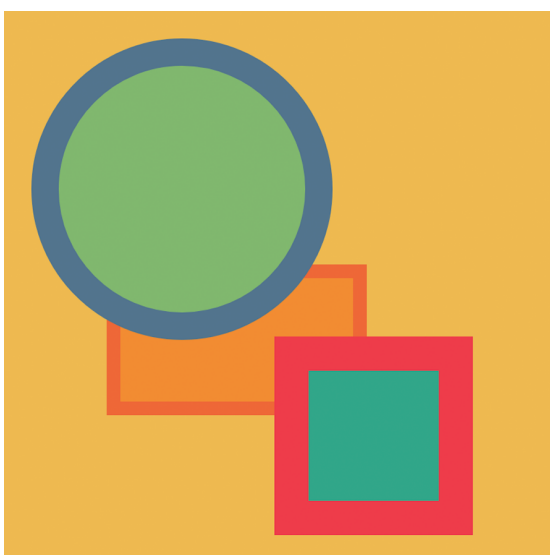


Figura 7. Figuras coloridas

— point()
— line()
— arc()
— triangle()

Cada una de ellas funciona un poco diferente, así que te recomiendo que consultes la página de referencia <https://p5js.org/es/reference/> para obtener más información e instrucciones.

**Actividad 1**

Usa las funciones hasta aquí mencionadas para crear tu propio diseño geométrico. Experimenta con diferentes parámetros para posicionarlas en distintos lugares del lienzo. Prueba cambiar las dimensiones de las figuras y los colores.

## Errores al programar: cuando algo sale mal

La programación puede ser un trabajo perspicaz y es fácil equivocarse en la sintaxis (especialmente para los principiantes, porque todavía no han aprendido exactamente cómo funciona la sintaxis). Pero no te preocupes, si algo sale mal, recibirás un mensaje de error. El mensaje de error aparecerá en ese pequeño cuadrado debajo del área de edición del código fuente, llamada «consola».

Si no puedes encontrar el problema, ¡no te preocupes! El mensaje de error te dará algunas pistas, pero es posible que algunas veces te diga exactamente donde se encuentra el error. El número al final del error indica el número de la línea en donde p5.js ha detectado el problema o error (a veces habrá más de dos líneas, por lo que tendrás que buscar). El mensaje te dará una idea de cuál es el problema, aunque la descripción se filtra a través del lenguaje extraño de los analizadores de lenguaje de programación, pero no entraremos mucho en detalles técnicos aquí.Véase en la figura 8 un ejemplo.


Figura 8.Mensaje de error

Este mensaje de error muestra que hay un problema en la línea 7: deliberadamente escribí «ellise» en lugar de «ellipse», y JavaScript nos informa que no tiene la menor idea de qué es «ellise». Cuando JavaScript dice que algo como «is not defined» (no está definido), eso es lo que significa. JavaScript solo sabe acerca de los objetos integrados, las variables y las funciones definidas en este lenguaje, y las variables y funciones en las bibliotecas que está utilizando, como la de p5.js.

Con las funciones hasta aquí mencionadas, es posible realizar una variedad de coloridos diseños geométricos. Pero esto es tan solo el comienzo para aprender a programar con p5.js. Después de familiarizarse con los elementos básicos, podemos entonces estudiar otros conceptos matemáticos y métodos más avanzados de programación (como *variables* —numéricas o boolean—, *arreglos* —conjuntos de datos u objetos—, *loops*, *condicionales, funciones recursivas, clases de objetos,* entre muchas otras cosas más). En la siguiente sección mostraremos algunos ejemplos sencillos que pueden servir como actividades para explorar transformaciones básicas en el plano.

## Traslación, rotación y escala de figuras geométricas en el plano

La posición y tamaño de las figuras geométricas que podemos dibujar en p5.js se determinan con base en coordenadas y parámetros. Si deseamos cambiar la posición o tamaño de estas figuras, simplemente cam-

biamos sus coordenadas o parámetros. Alternativamente, podemos cambiar el sistema de coordenadas definido por defecto para obtener el mismo resultado. De hecho, podemos crear diferentes transformaciones incluyendo translación, rotación y escala.

## TRASLACIÓN

La función **translate()**, por ejemplo, cambia el sistema de coordenadas directamente como se muestra en las figuras 9 y 10.

En el ejemplo 1 notemos cómo el rectángulo se dibuja en la coordenada (0,0), pero se mueve con respecto a la posición del ratón en el lienzo, porque este es afectado por la función **translate()**.

```
translate(40, 20);
rect(20, 20, 20, 40);
```

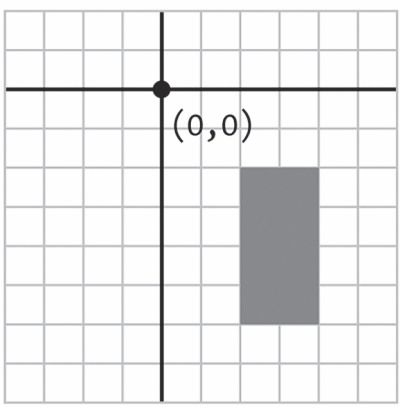



Figura 9. Traslación 40, 20

```
translate(60, 70);
rect(20, 20, 20, 40);
```

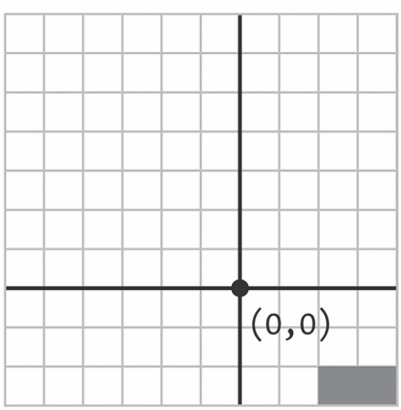



Figura 10. Traslación 60, 70

### *Ejemplo 1. Traslación de posición*

```
function setup() {
  createCanvas(400, 400);
  background(204);
}
function draw() {
  translate(mouseX, mouseY);
  rect(0, 0, 30, 30);
}
```

La variable de sistema **mouseX/mouseY** siempre contiene la posición horizontal/vertical actual del ratón, relativa al origen (0, 0) del lienzo.

La función **translate()** define el punto (0,0) en la pantalla como la posición (**mouseX, mouseY**). Cada vez que la función **draw()** se repite, el rectángulo **rect()** se dibuja en un nuevo origen, el cual depende de la posición del ratón.

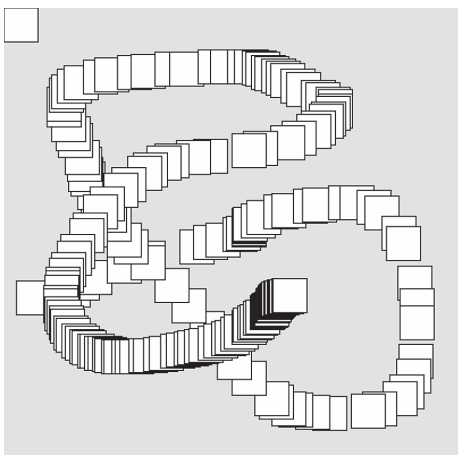


Figura 11. Traslación

**Actividad 2**

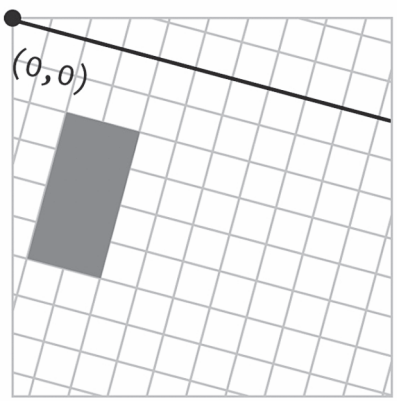
Observa qué sucede cuando cambiamos las coordenadas de posición (0, 0) del rectángulo por (–15, –15), esto es, rect(–15,–15,30,30). Explora también en tu código otros valores en la función translate() y observa el comportamiento de la translación definida.

## ROTACIÓN

La función **rotate()** realiza una rotación del sistema de coordenadas. Requiere solamente un parámetro,

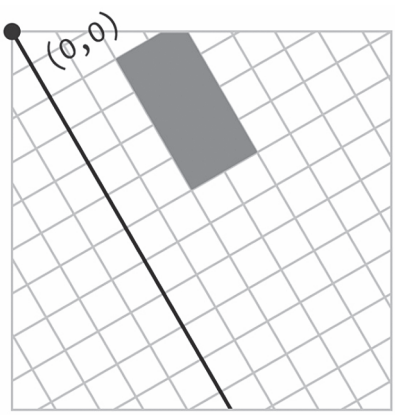
el cual es un ángulo (en radianes) para rotar. Siempre realiza las rotaciones con respecto al punto (0,0), es decir, al origen. Las figuras 12 y 13 muestran la diferencia entre rotar con valores positivos y negativos.

Para rotar una figura geométrica, primero definimos el ángulo de rotación con **rotate()**, luego dibujamos la figura. En el siguiente sketch, el parámetro para rotar **mouseX/100.0** estará entre 0 y 4 para definir el ángulo de rotación porque el valor de **mouseX** estará entre 0 y 400, el ancho del lienzo como se define en **createCanvas()**. Cuando mueves el ratón sobre el lienzo, el rectángulo rotará con respecto a la esquina superior izquierda, es decir, el origen (0,0).




```
rotate(PI/12);
rect(20, 20, 20, 40);
```

Figura 12. Rotación PI/12




```
rotate (-PI/3);
rect(20, 20, 20, 40);
```

Figura 13: Rotación –PI/3

### *Ejemplo 2. Rotación con respecto al origen*

```
function setup() {
  createCanvas(400, 400);
  background(204);
}
function draw() {
  rotate(mouseX / 100.0);
  rect(120, 100, 160, 20);
}
```

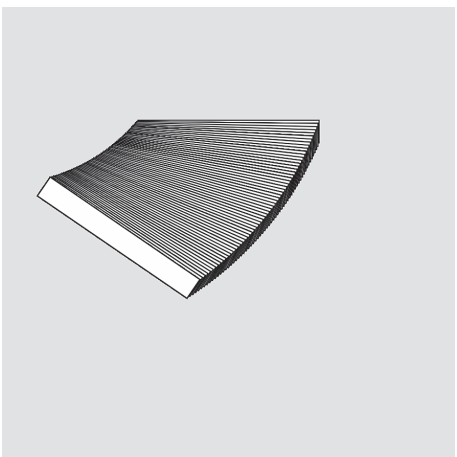


Figura 14. Rotación

**Actividad 3**

Observa qué sucede cuando cambiamos las coordenadas de posición (120, 100) del rectángulo por (-80, -10), esto es, rect(–80,–10,160,20). Con este cambio, ¿en dónde se encuentra el origen (0,0)? ¿Con respecto a qué punto rotará la figura?

## TRASLACIÓN Y DESPUÉS ROTACIÓN

Para rotar una figura con respecto a su centro, en lugar del origen del lienzo (la esquina superior izquierda), primero usamos la función **translate()** para moverlo a la posición deseada. Después usamos la función **rotate()** y finalmente dibujamos la figura con su centro en la nueva coordenada para (0,0). Analiza el siguiente ejemplo, donde hemos definido una *variable global* **angulo** con valor inicial 0.0. Observa que dentro de la función **draw()** hemos definido la ope-

ración angulo += 0.1. Esto significa que su valor incrementa 0.1 cada vez que **draw()** se actualiza.

### *Ejemplo 3. Traslación y después rotación*

```
var angulo = 0.0;
function setup() {
  createCanvas(400, 400);
   background(204);
}
function draw() {
  translate(mouseX, mouseY);
  rotate(angulo); rect(-15, -15, 30, 30);
  angulo += 0.1;
}
```

Una *variable* almacena un valor en la memoria para que pueda usarse más tarde en un programa. Una variable se puede usar muchas veces dentro de un solo programa, y el valor se cambia fácilmente mientras el programa se está ejecutando.

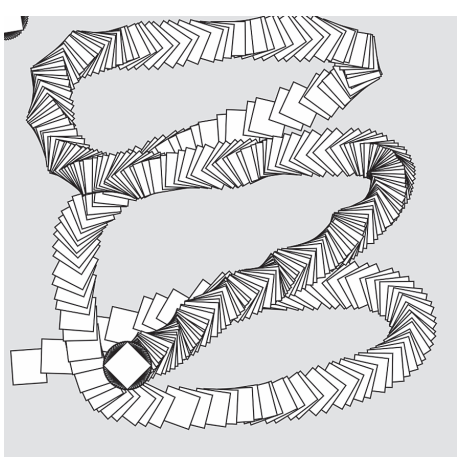


Figura 15. Traslación y rotación

**Actividad 4**

En el ejemplo anterior, cambia el orden de las funciones translate() y rotate(). Observa qué sucede. ¿Con respecto a qué punto rota la figura en este caso? ¿A dónde se traslada la figura?

[*Ayuda.* Te recomiendo usar las funciones rectMode(), ellipseMode() e imageMode() para dibujar más fácilmente las figuras geométricas desde su centro. Puedes leer los detalles de cómo funcionan estas funciones en la página de referencia <https://p5js.org/es/reference/>.

## ESCALA

La función **scale()** expande o contrae el plano del lienzo de forma proporcional en ancho y alto. Esto significa que la escala se modifica y por consecuencia todo lo que se dibuja en el lienzo cambia también de dimensiones. La cantidad de escala que se escribe debe ser en porcentajes decimales. Por ejemplo, el valor 1.5 en la función **scale()** determina un cambio del 150% mientras que el valor 3 determina un cambo del 300% (figuras 16 y 17).

Como en el caso de **rotate()**, la función **scale()** transforma el origen. Por lo que, similar a la función **rotate()**, para cambiar las dimensiones de una figura desde su centro, primero trasladamos su posición, después aplicamos la escala y finalmente dibujamos la figura con el centro en la coordenada (0,0). En el siguiente ejemplo podemos apreciar el efecto de la

```
scale(1.5);
rect(20, 20, 20, 40);
```

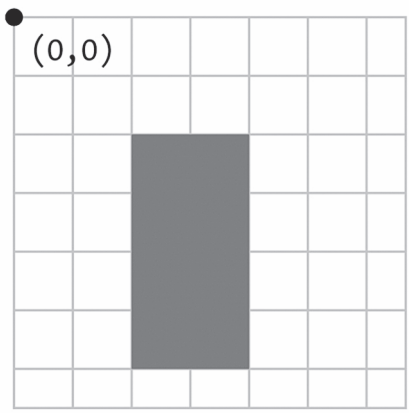



Figura 16. Escala 1.5

```
Scale(3);
rect(20, 20, 20, 40);
```

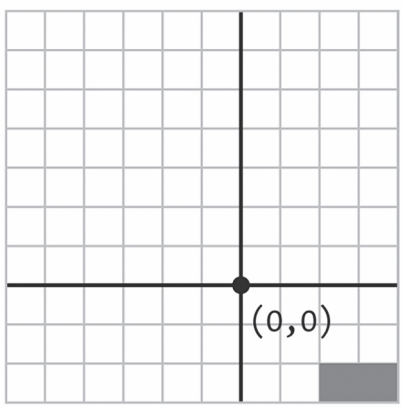



Figura 17. Escala 3

función escala. Cuando se mueve el ratón sobre el lienzo, observamos cómo cambian las dimensiones del cuadrado.

### *Ejemplo 4. Escala*

```
function setup() {
  createCanvas(400, 400);
  background(204);
}
function draw() {
  translate(mouseX, mouseY);
  scale(mouseX / 200.0);
  rect(-15, -15, 30, 30);
}
```

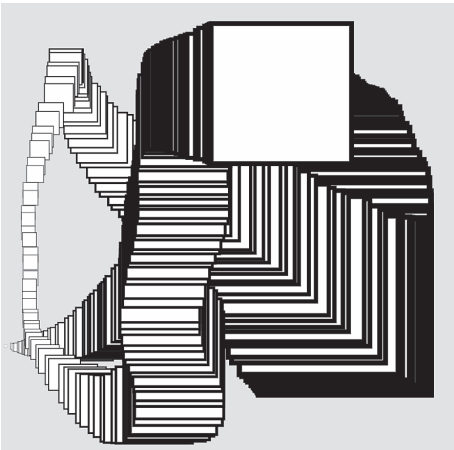


Figura 18. Escala

**Actividad 5**

Observa que, en el ejemplo anterior, la función scale() afecta también al borde del cuadrado. Es decir, cuando el valor de la escala es mayor, el borde será más grueso y viceversa. Para mantener un borde consistente cada vez que la figura cambia de dimensiones, podemos dividir el valor del tamaño del borde por un valor escalar. Para ello, define la siguiente variable dentro de la función draw():

```
var escalar = mouseX / 200.0;
```

Después modifica el código anterior con las siguientes líneas:

```
scale(escalar);
strokeWeight(1.0 / escalar);
rect(-15, -15, 30, 30);
```

Explora y analiza lo que sucede en el lienzo con estos cambios. ¿Por qué dividimos la variable mouseX entre 200.0? ¿Podemos dividirlo por otro valor? ¿Qué sucede si cambiamos este número?

Hasta aquí hemos explorado funciones para cambiar la posición y tamaño de figuras geométricas por medio de transformaciones del plano definido en el lienzo. Las actividades mencionadas anteriormente pueden ser útiles para explorar los conceptos de translación, rotación y escala.

Por supuesto, esta no es la única forma de modificar la posición o tamaño de figuras geométricas en p5.js. También podemos utilizar funciones matemáticas. El siguiente código utiliza la función seno para dibujar un círculo con un movimiento oscilatorio de arriba abajo:

```
var angulo = 0.0;
var escalar = 40;
var offset = 60;
var vel = 0.05;
function setup() {
  createCanvas(240, 120);
}
function draw() {
  background(0);
  var y = offset + sin(angulo + 0.4) * escalar;
  ellipse(120, y, 40, 40); angulo += vel;
}
```

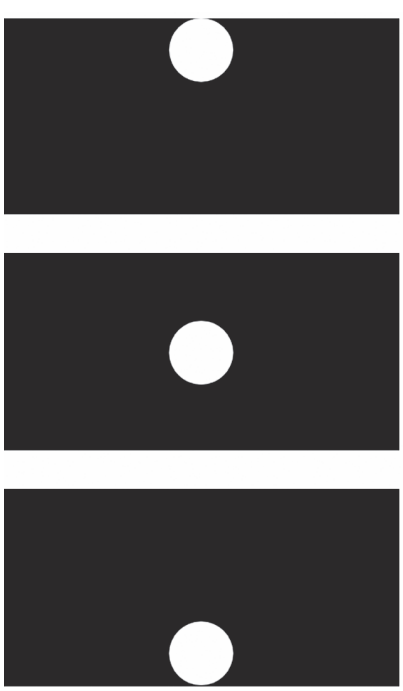


Figura 19: Movimiento oscilatorio

**Actividad 6.1**

Modifica el código anterior de tal forma que en el círculo se mantenga en el centro del lienzo, pero su tamaño cambie. ¿Cuáles son los parámetros o variables que necesitas modificar?

**Actividad 6.2**

Combina el cambio de dimensiones con el movimiento oscilatorio. ¿Cuáles son los parámetros o variables que necesitas modificar? ¿Es necesario definir nuevas variables?

## Enseñanza de la programación en las escuelas

La enseñanza de la programación en las escuelas no es una novedad. Logo, por ejemplo, fue diseñado y desarrollado por Seymour Papert, Wally Feurzeig y Cynthia Solomon a finales de la década de 1960 (Papert y Solomon, 1971). Según Terceros (2019) el uso de Logo tenía como principal objetivo iniciar en los lenguajes de programación a niños desde corta edad. Por medio de una interfaz con un personaje conocido como *Tortuga*, el cual era controlado a partir de una serie de órdenes simples para diseñar gráficos elementales (¿Te suena esto familiar?). A pesar de que esta tendencia no tuvo gran éxito en ese entonces, ha influido lo suficiente para dar vida, décadas más tarde, a proyectos como Processing o p5.js. (entre otros ejemplos se encuentra el lenguaje de programación *Scratch*).

El interés actual por la enseñanza de la programación en las escuelas se debe principalmente a que nos hemos introducido de lleno en una era en que la capacidad de procesar información es una habilidad fundamental. En tal contexto, quien sabe programar se encuentra en una situación ventajosa. En poco más de dos generaciones, los ordenadores pasaron de ser grandes aparatos reservados para algunas empresas tecnológicas a pequeños dispositivos de diferentes tipos, que se encuentran al alcance de la mano de cualquier persona. La masificación de los ordenadores ha conducido a una era de la información y ha abierto para todos las puertas a una fuente inagotable de conocimiento.

El aprendizaje de la programación por medio de los ordenadores constituye una gran ventaja a nivel de competencias en la época actual. Además, en un sentido estrictamente educacional, la programación puede dotar a los alumnos que la estudian y practican de una mayor capacidad de razonamiento lógico, pensamiento estructurado o incluso una mayor imaginación (Briz y Serrano, 2018).

## Comentarios finales

En mi opinión, p5.js puede ser una herramienta potencial para aprender a programar, y por supuesto también para aprender matemáticas, no solo de forma individual pero también en el salón de clases a partir de nivel secundaria en adelante. En primer lugar, porque esta librería está diseñada para aprender de forma intuitiva y constructiva. En segundo lugar, porque es de acceso libre, gratuito y además no requiere de instalación en el ordenador. Solo se necesita tener acceso a internet y un ordenador a la mano. Esto último ha permitido la creación y crecimiento de una gran comunidad que comparte constantemente materiales de referencia, tutoriales (ver, por ejemplo Costa, 2019) y ayuda en línea de forma gratuita para toda persona interesada en aprender a programar con p5.js. Un ejemplo reciente es la hoja de trucos para principiantes (ver Moren, 2019), traducida a 10 idiomas.

Asimismo, cabe resaltar que con p5.js no solo es posible crear imágenes estáticas coloridas sino también sofisticados diseños, animaciones artísticas, juegos interactivos e incluso aplicaciones web (figuras 20, 21, 22 y 23).

Finalmente, si deseas aprender a programar de forma individual o si eres un profesor en busca de alternativas para introducir la programación en el salón de clases de forma intuitiva y constructiva, entonces p5.js es una excelente opción.

Figura 20. Retrato de Marielle Semente
Autor: Vamoss
<https://www.openprocessing.org/sketch/873380>

Figura 21. Portal
Autor: Jason Labbe
<https://www.openprocessing.org/sketch/897940>

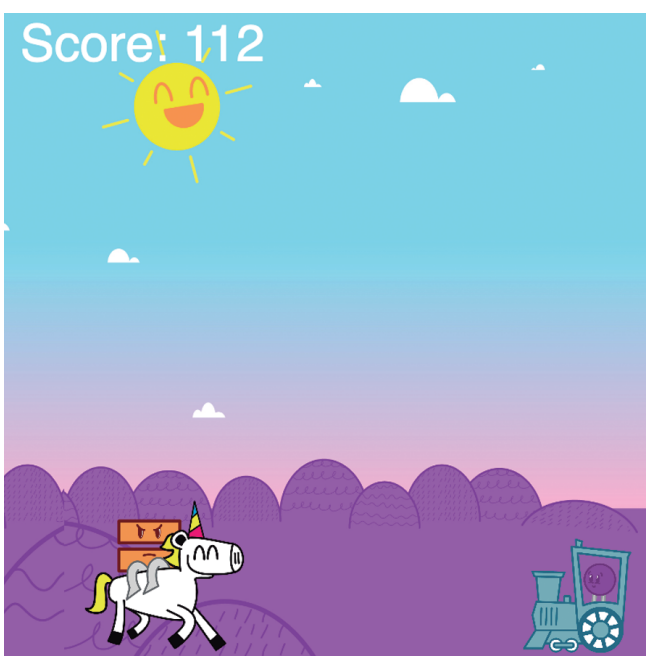



Figura 22. This Dot Jumper
Autor: Swiftpotato
<https://editor.p5js.org/swiftpotato/full/Ca6Q58Gq2>

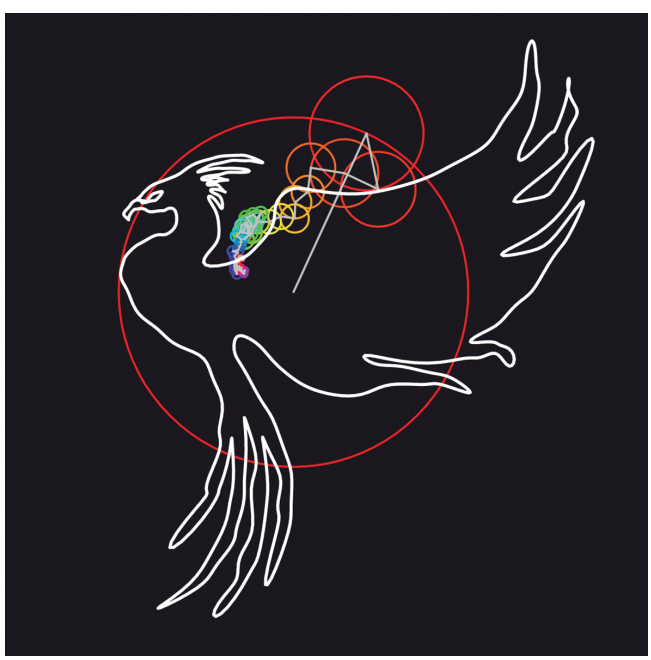


Figura 23.Transformada de Fourier: Dibujando curvas cerradas con epiciclos
Autor: Juan Carlos Ponce Campuzano
<https://jcponce.github.io/misc/draw-fourier.html>

## Referencias bibliográficas


Briz, Á., y Á. Serrano (2018), «Aprendizaje de las matemáticas a través del lenguaje de programación R en Educación Secundaria», *Educación Matemática,* vol. 30, n.º 1, 133-162.

Carbonnelle, P. (2019), *PYPL PopularitY of Programming Language*, <http://pypl.github.io/PYPL.html>.

Costa, H. (2019), *Introducción a la programación para artistas y principiantes,* <https://www.hektorprofe.net/curso/programacion-artistas-disenadores-principiantes/p5js-introduccion>.

McCarthy, L., C. Reas y B. Fry (2018), *Introducción a* p5.js, Processing Foundation.

Moren, B. (2019), A *p5js cheat sheet for beginners!*, <https://github.com/bmoren/p5js-cheat-sheet>.

p5.js. (2013), <https://p5js.org/es/>.

Papert, S., y C. Solomon (1971), T*wenty Things To Do With A Computer,* <https://dspace.mit.edu/handle/1721.1/5836>.

Terceros, I. (2019), «Programación creativa: pensamiento computacional y constructivismo desde contextos interculturales», en P. Véles y Y. Yaguana (eds.), *Nuevas tecnologías en el proceso de enseñanza-aprendizaje*, Universidad Técnica Particular de Loja, 121-125.



Juan Carlos Ponce Campuzano
The University of Queensland
<j.ponce@uq.edu.au>