

Código p5.js explicado

Esto es una copia de una conversación entre ChatGPT y Anónimo.

Dame un código sencillo en p5.js que use los conceptos: variables, condicionales, operadores, bucles y funciones. Comenta cada linea y explica de manera sencilla cada proceso. Una frase y sin emojis.

```
javascript

// Declaramos una variable para la posición horizontal de la pelota
let x;

// Declaramos una variable para la velocidad de movimiento
let speed;

// La función setup() se ejecuta una sola vez al inicio del programa
function setup() {
    // Creamos una ventana de 400 píxeles de ancho por 400 de alto
    createCanvas(400, 400);

    // Damos un valor inicial a la posición horizontal
    x = 50;

    // Damos un valor inicial a la velocidad
    speed = 2;
}

// La función draw() se repite muchas veces por segundo en un bucle automático
function draw() {
    // Pintamos el fondo de la pantalla con un gris claro en cada fotograma
    background(220);

    // Sumamos la velocidad a la posición para mover la pelota hacia la derecha
    x = x + speed;

    // Si la pelota se pasa del borde derecho o izquierdo, cambiamos el sentido
    if (x > width - 25 || x < 25) {
        // Invertimos la velocidad multiplicándola por -1 para que vaya al lado contrario
        speed = -speed;
    }

    // Usamos un bucle for para dibujar varias pelotas en diferentes alturas
    for (let i = 0; i < 5; i = i + 1) {
        // Calculamos la posición vertical de cada pelota usando la variable i
        let y = 100 + i * 50;

        // Llamamos a la función que dibuja una pelota en la posición calculada
        drawBall(x, y, 30);
    }
}

// Definimos una función que dibuja una pelota en la pantalla
function drawBall(px, py, size) {
    // Elegimos un color azul para la pelota
    fill(100, 150, 255);

    // Quitamos el borde del dibujo de la pelota
    noStroke();

    // Dibujamos un círculo usando la posición y el tamaño que recibe la función
    ellipse(px, py, size, size);
}
```

Explicación sencilla de cada proceso (una frase cada una):

- Variables: Guardamos datos como la posición y la velocidad en variables para poder usarlos y modificarlos durante el programa.
- Funciones setup y draw: Usamos setup para configurar una vez la ventana y draw para repetir continuamente el dibujo en la pantalla.
- Operadores: Utilizamos operadores como +, -, >, < y || para sumar valores, invertir la velocidad y comparar posiciones.
- Condicionales: Con el **if** comprobamos si la pelota toca los bordes y, si ocurre, cambiamos la dirección de movimiento.
- Bucles: Con el **for** repetimos varias veces el mismo código para dibujar varias pelotas en diferentes posiciones verticales.
- Función propia (drawBall): Creamos la función **drawBall** para agrupar el código de dibujar una pelota y reutilizarlo fácilmente varias veces.

Me gusta, pero las pelotas no llegan a tocar el borde. Quiero que parezca que rebotan en los extremos del canvas. También quiero que tengan diferentes ritmos. Explícame como lo haces de manera sencilla.

Aquí tienes una versión mejorada donde las pelotas sí llegan a tocar el borde y cada una tiene un ritmo diferente. Después del código te explico de manera sencilla qué se hizo y por qué funciona.

```
javascript

// Número de pelotas
let total = 5;

// Arreglos para guardar posición y velocidad de cada pelota
let xs = [];
let speeds = [];

// setup se ejecuta una vez al inicio
function setup() {
  createCanvas(400, 400);

  // Inicializamos posiciones y velocidades diferentes para cada pelota
  for (let i = 0; i < total; i++) {
    xs[i] = random(25, width - 25); // Posición inicial aleatoria sin salir del borde
    speeds[i] = random(1, 4); // Velocidad distinta para cada pelota
  }
}

// draw se repite muchas veces formando un bucle
function draw() {
  background(220);

  // Dibujamos y movemos cada pelota
  for (let i = 0; i < total; i++) {
    let y = 100 + i * 50; // Altura diferente para cada pelota

    // Movimiento
    xs[i] = xs[i] + speeds[i];

    // Rebote correcto tocando bordes
    if (xs[i] >= width - 15 || xs[i] <= 15) {
      speeds[i] = -speeds[i];
    }

    // Dibujo
    drawBall(xs[i], y, 30);
  }
}
```

```

}

// Función para dibujar una pelota
function drawBall(px, py, size) {
  fill(100, 150, 255);
  noStroke();
  ellipse(px, py, size, size);
}

```

Explicación sencilla

1. Por qué ahora tocan el borde

Antes usábamos `width - 25`, pero el tamaño de la pelota es 30, así que su radio es 15. Ahora usamos exactamente `width - 15` y `15` para que el borde del círculo toque el borde del canvas.

2. Cómo simulo el rebote

Cuando la posición llega a uno de esos límites, multiplico la velocidad por `-1` y así cambio su dirección.

3. Cómo les doy ritmos diferentes

En lugar de una sola velocidad, ahora uso un arreglo con muchas velocidades y a cada pelota le asigno una velocidad aleatoria al inicio.

4. Cómo muevo todas las pelotas a la vez

Uso un bucle `for` que repite el proceso para cada pelota, de modo que todas se mueven y rebotan de forma independiente.

Si quieras puedo agregar colores diferentes, tamaños distintos o incluso que reboten también verticalmente.

no entendí bien el punto 4, lo del bucle for(), me lo puedes explicar detalladamente?

Claro, vamos a destripar ese `for` con calma.

Primero te lo vuelvo a enseñar aislado:

```

javascript

for (let i = 0; i < total; i++) {
  let y = 100 + i * 50;

  xs[i] = xs[i] + speeds[i];

  if (xs[i] >= width - 15 || xs[i] <= 15) {
    speeds[i] = -speeds[i];
  }

  drawBall(xs[i], y, 30);
}

```

Te lo explico paso a paso.

1. ¿Qué es la `i`?

- `i` es una variable que usamos como **contador**.

- Empieza con un valor y va cambiando en cada vuelta del bucle.
- Sirve para saber **en qué pelota estamos trabajando**.

En este caso, las pelotas están numeradas así:

- Pelota 0
- Pelota 1
- Pelota 2
- Pelota 3
- Pelota 4

Por eso usamos **i** para acceder a la posición y velocidad de cada una: **xs[i]**, **speeds[i]**.

2. Estructura del **for**

El **for** tiene tres partes:

```
javascript

for (let i = 0; i < total; i++) {
    ↑      ↑      ↑
    inicio  condición  cambio
}
```

1. **let i = 0;**

Empieza el contador **i** en 0, o sea, empezamos por la **primera pelota**.

2. **i < total;**

Esta es la condición para seguir dando vueltas.

Si **total** es 5, entonces **i** tomará los valores: 0, 1, 2, 3, 4.

Cuando **i** sea 5, ya no se cumple **i < total** y el bucle se detiene.

3. **i++**

Es lo mismo que **i = i + 1**.

Después de cada vuelta, pasamos de la pelota 0 a la 1, luego a la 2, etc.

3. Qué pasa dentro del bucle (una vuelta)

Dentro del **for** está el código que se ejecuta para **una sola pelota**:

```
javascript

let y = 100 + i * 50;
xs[i] = xs[i] + speeds[i];
if (xs[i] >= width - 15 || xs[i] <= 15) {
    speeds[i] = -speeds[i];
}
drawBall(xs[i], y, 30);
```

Lo explico como si fuera una historia:

1. `let y = 100 + i * 50;`

Calculamos la altura de la pelota.

- Si `i` es 0, `y = 100`
- Si `i` es 1, `y = 150`
- Si `i` es 2, `y = 200`

Así cada pelota queda en una línea diferente.

2. `xs[i] = xs[i] + speeds[i];`

Actualizamos la **posición horizontal** de esa pelota.

- Usamos la posición de esa pelota `xs[i]`.
- Le sumamos su velocidad `speeds[i]`.

Cada pelota tiene su propia velocidad, por eso se mueven distinto.

3. `if (xs[i] >= width - 15 || xs[i] <= 15) { ... }`

Comprobamos si esa pelota ha llegado a uno de los bordes.

- Si toca el borde derecho (`xs[i] >= width - 15`)
- O el borde izquierdo (`xs[i] <= 15`)

Entonces dentro del `if` hacemos que cambie de dirección.

4. `speeds[i] = -speeds[i];`

Invertimos la velocidad de esa pelota concreta.

- Si iba a la derecha (velocidad positiva), pasa a ir a la izquierda.
- Si iba a la izquierda (velocidad negativa), pasa a ir a la derecha.

5. `drawBall(xs[i], y, 30);`

Dibujamos esa pelota en pantalla usando:

- Su posición horizontal `xs[i]`
- Su posición vertical `y`
- Su tamaño `30`

4. Cómo se aplica esto a todas las pelotas

Piensa que el bucle es como una lista de tareas:

- Vuelta con `i = 0`: mueve y dibuja la pelota 0
- Vuelta con `i = 1`: mueve y dibuja la pelota 1
- Vuelta con `i = 2`: mueve y dibuja la pelota 2
- Vuelta con `i = 3`: mueve y dibuja la pelota 3
- Vuelta con `i = 4`: mueve y dibuja la pelota 4

Y todo eso ocurre **en cada frame** dentro de `draw()`, muchas veces por segundo.

Por eso parece que todas las pelotas se mueven a la vez y rebotan de forma independiente.

5. Resumen muy corto

- El **for** repite el mismo código varias veces.
- La variable **i** indica de qué pelota nos estamos ocupando en cada vuelta.
- Los arreglos **xs** y **speeds** guardan los datos de **todas** las pelotas.
- Usando **xs[i]** y **speeds[i]** movemos y hacemos rebotar cada pelota por separado.

Si quieres, en el siguiente paso puedo reescribir el código sin bucle para que veas cómo quedaría y compares lo largo que sería.

Vale ya lo entendí. Ahora quiero que el fondo sea negro y que las pelotas, cambien de color cada vez que rebotan en uno de los bordes.

Aquí tienes una versión modificada donde:

- El fondo es negro.
- Cada pelota cambia de color justo cuando rebota en un borde.

```
javascript

// Número de pelotas
let total = 5;

// Arreglos para posición, velocidad y color de cada pelota
let xs = [];
let speeds = [];
let colors = [];

// setup se ejecuta una vez
function setup() {
  createCanvas(400, 400);

  for (let i = 0; i < total; i++) {
    xs[i] = random(25, width - 25); // Posición inicial
    speeds[i] = random(1, 4); // Velocidad distinta

    // Color inicial aleatorio
    colors[i] = color(random(255), random(255), random(255));
  }
}

// draw se repite constantemente
function draw() {
  background(0); // Fondo negro

  for (let i = 0; i < total; i++) {
    let y = 100 + i * 50;

    // Movimiento
    xs[i] = xs[i] + speeds[i];

    // Rebote
    if (xs[i] >= width - 15 || xs[i] <= 15) {
      speeds[i] = -speeds[i]; // Cambia de dirección

      // Cambia de color al rebotar
      colors[i] = color(random(255), random(255), random(255));
    }

    // Dibujo
    drawBall(xs[i], y, 30, colors[i]);
  }
}
```

```

    }
}

// Función para dibujar una pelota con color
function drawBall(px, py, size, col) {
  fill(col);
  noStroke();
  ellipse(px, py, size, size);
}

```

Si quieres, puedo hacer que el cambio de color sea más suave, que las pelotas brillen o que dejen un rastro visual. ¿Quieres alguna de esas mejoras?

Necesito que me expliques como has hecho lo del color

Claro, te explico detalladamente cómo funciona lo del color paso a paso.

1. Cada pelota necesita su propio color

Como tenemos varias pelotas, no podemos usar una sola variable de color. Por eso creamos un **arreglo**:

```

javascript

let colors = [];

```

Esto permite guardar un color distinto en cada posición, igual que hicimos con las posiciones (**xs**) y velocidades (**speeds**).

2. En `setup()` damos un color inicial a cada pelota

Dentro del bucle que prepara todos los datos iniciales, añadimos:

```

javascript

colors[i] = color(random(255), random(255), random(255));

```

Esto significa:

- `color(r, g, b)` crea un color usando los valores rojo, verde y azul.
- `random(255)` genera un número aleatorio entre 0 y 255.
- Así cada pelota empieza con un color aleatorio distinto.

Ejemplo mental:

- Pelota 0 -> azul verdoso
- Pelota 1 -> rosa
- Pelota 2 -> amarillo
(pero cambia cada ejecución)

3. Cambiar el color solo cuando hay rebote

En el código del rebote, añadimos esta línea:

```
javascript
colors[i] = color(random(255), random(255), random(255));
```

Justo cuando la pelota toca un borde y se invierte la velocidad, asignamos otro color aleatorio. Eso hace que parezca que la pelota “reaccione” visualmente al impacto.

La estructura queda así:

```
javascript
if (xs[i] >= width - 15 || xs[i] <= 15) {
  speeds[i] = -speeds[i]; // Cambia dirección
  colors[i] = color(random(255), random(255), random(255)); // Cambia color
}
```

4. Dibujar usando ese color

Para que cada pelota se pinte realmente con su color correspondiente, la función `drawBall` recibe ese color como argumento:

```
javascript
drawBall(xs[i], y, 30, colors[i]);
```

Y dentro de la función:

```
javascript
fill(col);
ellipse(px, py, size, size);
```

`fill(col)` le dice a p5.js que la figura que vamos a dibujar use ese color.

5. Resumen en una frase por paso

1. Creamos un arreglo para guardar un color por pelota.
2. Le damos a cada pelota un color inicial aleatorio.
3. Cuando rebota, asignamos un nuevo color aleatorio.
4. Pasamos el color a la función que dibuja la pelota.
5. Usamos `fill()` para pintar la pelota con ese color.

Siquieres, puedo hacer que los colores cambien gradualmente, que tengan brillo, o que el fondo también reaccione a los rebotes. ¿Quieres algo así?