

HTML5: CONTEXTO Y CARACTERIZACIÓN

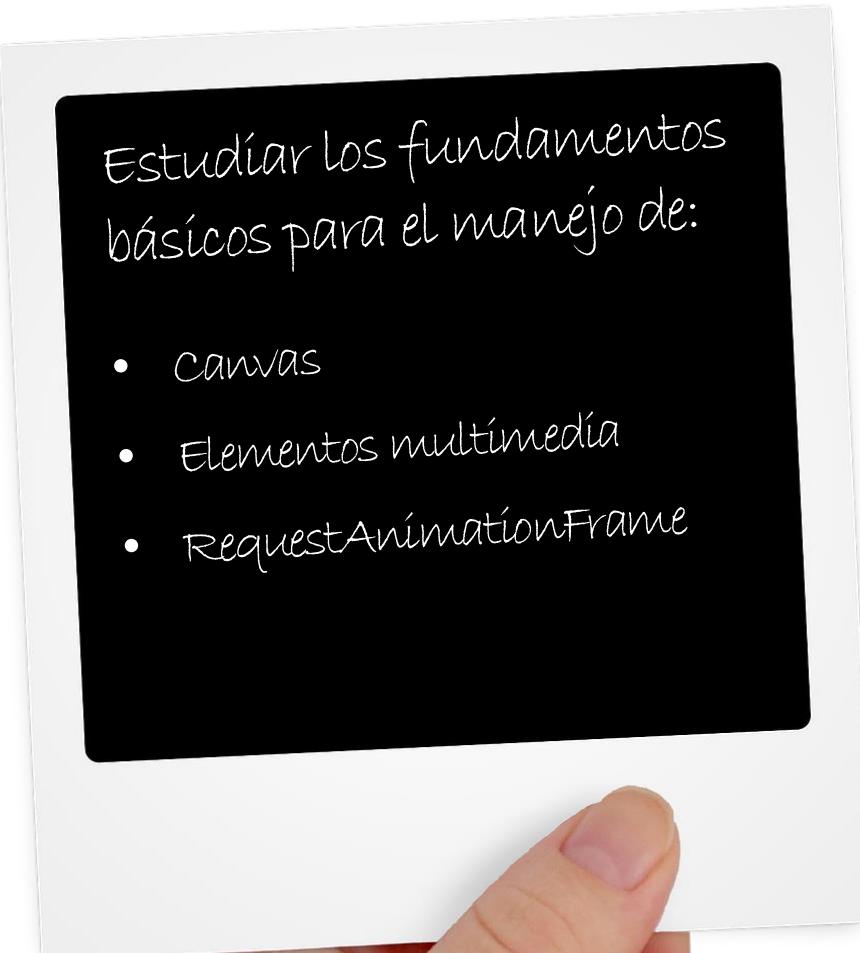
Liliana Patricia Santacruz Valencia



Contenido

Estudiar los fundamentos básicos para el manejo de:

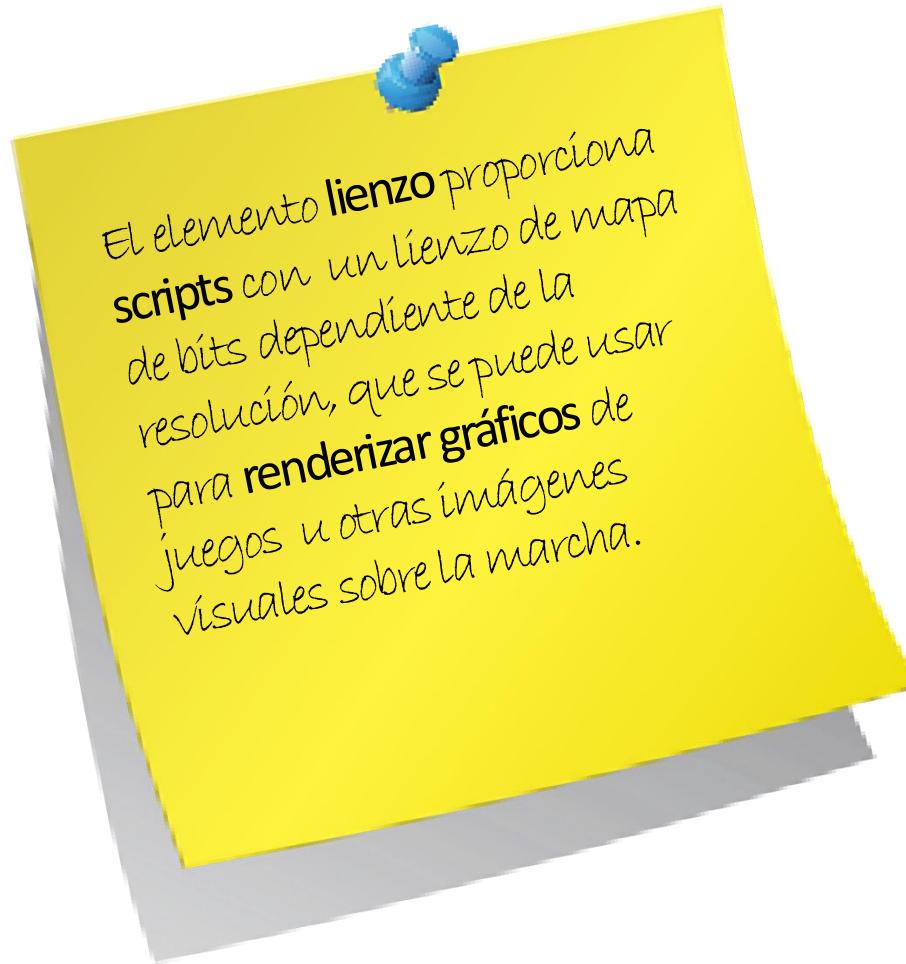
- canvas
- Elementos multimedia
- RequestAnimationFrame





Canvas

Canvas (1)

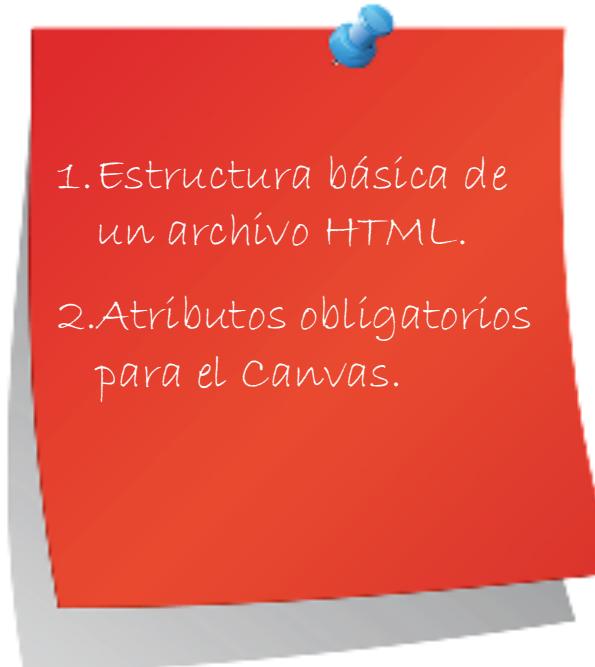


Canvas (2)



Canvas (3)

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-type" content="text/html; charset=utf-8">
5     <title>Ejemplo de página HTML</title>
6     </script>
7   </head>
8   <body onload="pageLoaded();">
9     2 <canvas width="600" height="400" id="testcanvas" style="border:1px solid black;">
10       Tu navegador no soporta la etiqueta Canvas de HTML5. Por favor, utilice otro navegador.
11     </canvas>
12   </body>
13 </html>
```

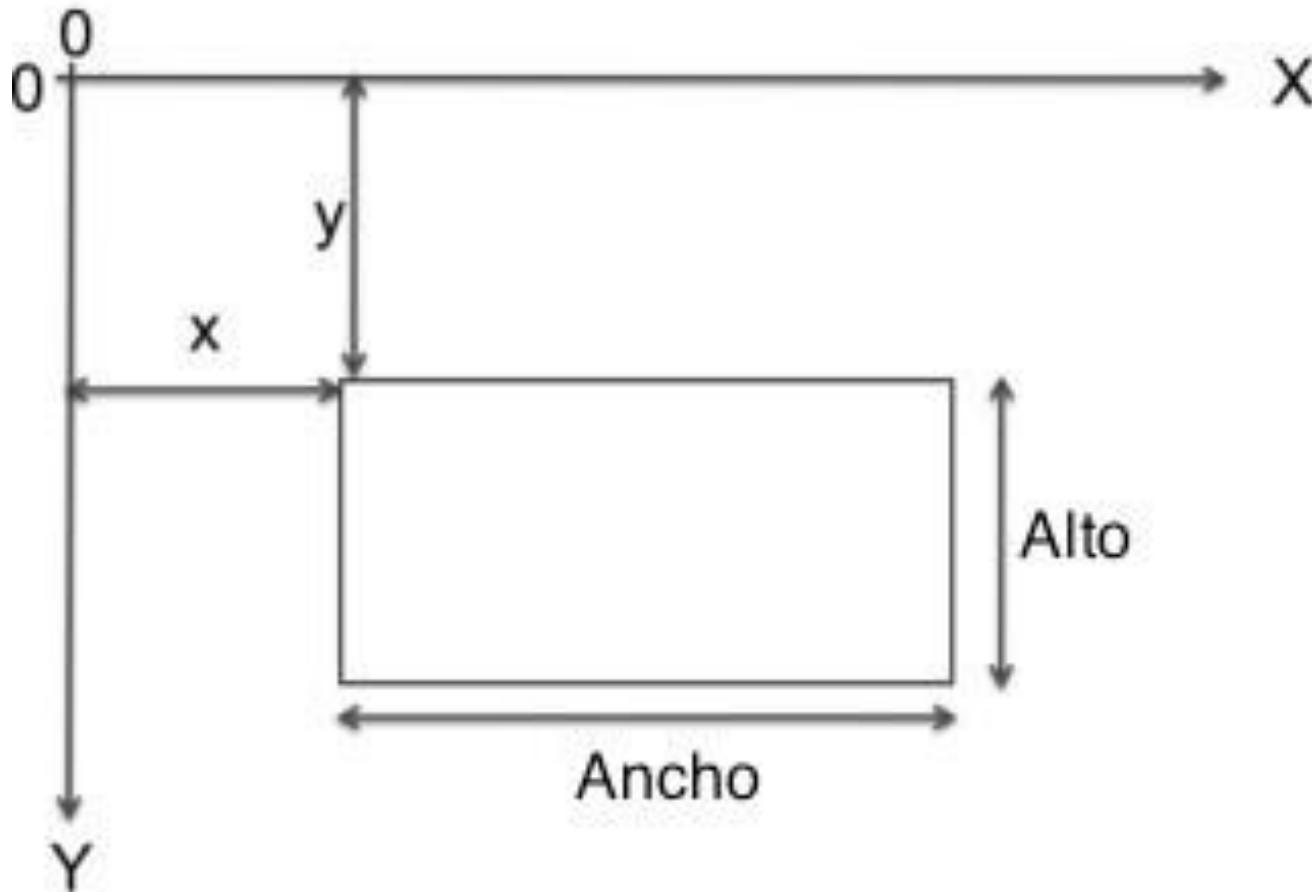


Canvas (4)

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta http-equiv="Content-type" content="text/html; charset=utf-8">
5         <title>Ejemplo de página HTML</title>
6         <script type="text/javascript" charset="utf-8">
7             function pageLoaded(){
8                 // Obtener el manejador del objeto canvas
9                 var canvas = document.getElementById('testcanvas');
10
11                 // Obtener el contexto 2D para este canvas
12                 var context = canvas.getContext('2d');
13
14                 // Nuestro código de los dibujos ...
15             </script>
16         </head>
17         <body onload="pageLoaded();">
18             1 <canvas width="600" height="400" id="testcanvas" style="border:1px solid black;">
19                 Tu navegador no soporta la etiqueta Canvas de HTML5. Por favor, utilice otro navegador.
20             </canvas>
21         </body>
22     </html>
```

1. Sistema de
coordenadas
para Canvas.

Canvas (5)

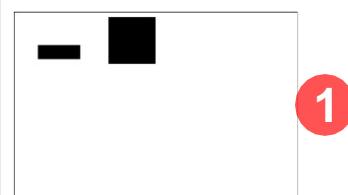


Sistema de coordenadas para Canvas

```

16 // DIBUJAR RECTÁNGULOS
17
18 // Rectángulos llenos
19 // Dibujar un cuadrado sólido con ancho y alto de 100 pixels en (200,10)
20 context.fillRect (200,10,100,100);
21 // Dibujar un cuadrado sólido con ancho de 90 y alto 30 pixels en (50,70)
22 context.fillRect (50,70,90,30);
23
24 // Contornos rectangulares
25 // Dibujar un contorno rectangular se ancho y alto de 50 pixels en (110,10)
26 context.strokeRect(110,10,50,50);
27 // Dibujar un contorno rectangular se ancho y alto de 50 pixels en (30,10)
28 context.strokeRect(30,10,50,50);
29
30 // Otros rectángulos
31 // Rectángulo de 30 pixeles de ancho y 20 de alto en (210,20)
32 context.clearRect(210,20,30,20);
33 // Rectángulo de 30 pixeles de ancho y 20 de alto en (260,20)
34 context.clearRect(260,20,30,20);
35
36 // DIBUJAR FORMAS COMPLEJAS
37
38 // Triángulo lleno
39 context.beginPath();
40 context.moveTo(10,120); // Comenzar a dibujar en 10,120
41 context.lineTo(10,180);
42 context.lineTo(110,150);
43 context.fill();           // Cerrar la forma y rellenarla
44
45 // Contornos triangulares
46 context.beginPath();
47 context.moveTo(140,160); // comenzar a dibujar en 140,160
48 context.lineTo(140,220);
49 context.lineTo(40,190);
50 context.closePath();
51 context.stroke();
52
53 // OTRAS FORMAS COMPLEJAS
54
55 context.beginPath();
56 context.moveTo(160,160); // comenzar a dibujar en 160,160
57 context.lineTo(170,220);
58 context.lineTo(240,210);
59 context.lineTo(260,170);
60 context.lineTo(190,140);
61 context.closePath();
62 context.stroke();

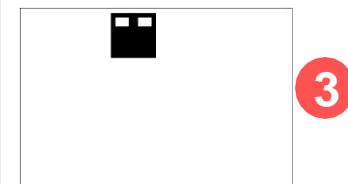
```



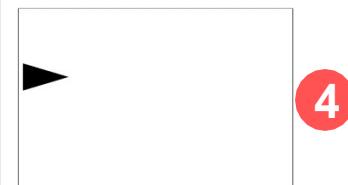
1



2



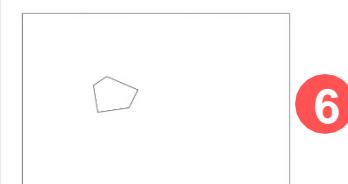
3



4



5



6



Canvas (6)

- `beginPath()`: Comienza a dibujar una nueva forma.
- `closePath()`: Cierra la ruta dibujando una línea desde el punto de dibujo actual hasta el punto de inicio.
- `Fill(), stroke()`: Llena o dibuja un contorno de la forma dibujada.
- `moveTo(x,y)`: Mueve el punto de dibujo a la coordenada (x,y).
- `lineTo(x,y)`: Dibuja una línea desde el punto de dibujo actual a la coordenada (x,y).
- `arc(x,y, radio, startAngle, endAngle, anticlockwise)`: Dibuja un arco en la coordenada (x,y) con radio específico.



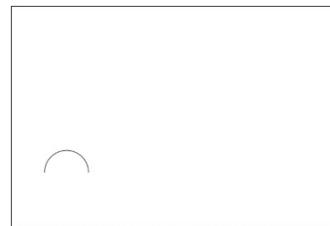
Canvas (7)

utilizar estos métodos para dibujar una ruta compleja implica los siguientes pasos:

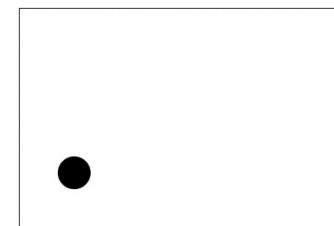
- utilizar **beginPath()**: Para comenzar a grabar la nueva forma.
- utilizar **moveTo()**, **lineTo()** y **arc()** para crear la forma.
- Opcionalmente, cierra la forma utilizando **closePath()**.
- utilizar el **stroke()** o **fill()** para dibujar un contorno o una forma rellena. El uso de **fill()** cierra automáticamente cualquier ruta abierta.

Canvas (8)

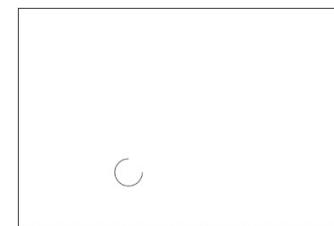
```
64          // Dibujar arcos  
65  
66          // Dibujar semicírculos  
67          context.beginPath();  
68  1 // Dibujar un arco en (100,300) con radio 40 de 0 a 180 grados, contrareloj  
69          context.arc(100,300,40,0,Math.PI,true);      //(PI radianes = 180 grados)  
70          context.stroke();  
71  
72          // Dibujar un círculo completo  
73          context.beginPath();  
74  2 // Dibujar un arco en (100,300) con radio de 30 de 0 a 360 grados, contrareloj  
75          context.arc(100,300,30,0,2*Math.PI,true); // (2*PI radianes = 360 grados)  
76          context.fill();  
77  
78          // Dibujar un arco de tres cuartos  
79          context.beginPath();  
80  3 // Dibujar un arco en (200,300) con radio de 25 de 0 a 270 grados, sentido reloj  
81          context.arc(200,300,25,0,3/2*Math.PI,false); // (3/2*PI radianes = 270 grados)  
82          context.stroke();  
--
```



1



2



3

Canvas (9)

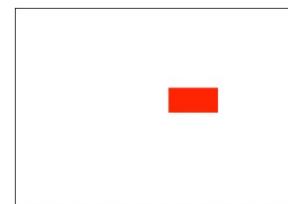
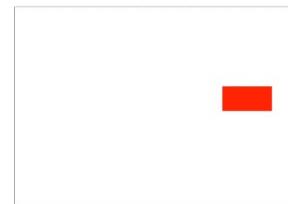
```
84 // DIBUJAR TEXTO  
85  
86 context.fillText('This is some text...',330,40);  
87  
88 // Modificar fuentes  
89 context.font = '10pt Arial';  
90 context.fillText('This is in 10pt Arial...',330,60);  
91  
92 // Dibujar contorno de texto  
93 context.font = '16pt Arial';  
94 context.strokeText('This is stroked in 16pt Arial...',330,80);
```

This is some text.
This is in 10pt Arial.
This is stroked in 16pt Arial.

1. El contexto también proporciona dos métodos para dibujar texto en el lienzo.
2. `strokeText(text,x,y)`: dibuja un contorno del texto en (x,y) .
3. `fillText(text,x,y)`: Rellena el texto en (x,y) .

Canvas (10)

```
96 //DIBUJAR COLORES Y TRANSPARENCIAS  
97  
98 // Fijar el color a rojo  
99 1 context.fillStyle = "red";  
100 // Dibujar un rectángulo relleno de color rojo  
101 context.fillRect (310,160,100,50);  
102  
103 // Fijar el color del contorno a verde  
104 2 context.strokeStyle = "green";  
105 // Dibujar un rectángulo relleno con verde  
106 context.strokeRect (310,240,100,50);  
107  
108 // Fijar el color de relleno a rojo utilizando rgb()  
109 3 context.fillStyle = "rgb(255,0,0)";  
110 // Dibujar un rectángulo relleno rojo  
111 context.fillRect (420,160,100,50);  
112  
113 // Fijar el color a verde con transparencia a 0.5  
114 4 context.fillStyle = "rgba(0,255,0,0.6)";  
115 // Dibujar un rectángulo verde semitransparente  
116 context.fillRect (450,180,100,50);
```

**1****2****3****4**

Para aplicar colores existen dos **propiedades** importantes que se pueden utilizar:

- **fillStyle**: Establece el color predeterminado para todas las operaciones futuras de relleno.
- **strokeStyle**: establece el color predeterminado para todas las futuras operaciones de trazo.

Canvas (11)

```
118 // DIBUJAR IMÁGENES  
119  
120 // Obtener un manejador para el objeto imagen  
121 var image = document.getElementById('spaceship');  
122  
123 // Dibujar la imagen en (0,350)  
124 context.drawImage(image,0,350);  
125  
126 // Escalar la imagen a la mitad del tamaño original  
127 context.drawImage(image,0,400,100,25);  
128  
129 // Dibujar parte de la imagen  
130 context.drawImage(image,0,0,60,50,0,420,60,50);
```





Canvas (12)

Para dibujar imágenes y sprites en el canvas se utiliza el método `drawImage()`. El contexto proporciona tres versiones diferentes de este método:

- `drawImage(image,x,y)`: Dibuja la imagen en el canvas en (x,y) .
- `drawImage(image,x,y,width,height)`: Escala la imagen al ancho especificado y la altura y luego dibuja en (x,y) .
- `drawImage(image,sourceX,sourceY,sourceWidth,sourceHeight,x,y,width,height)`: Recorta un rectángulo de la imagen (`sourceX, sourceY, sourceWidth, sourceHeight`), lo escala al ancho y alto especificados y lo dibuja en el canvas en (x,y) .

Canvas (13)

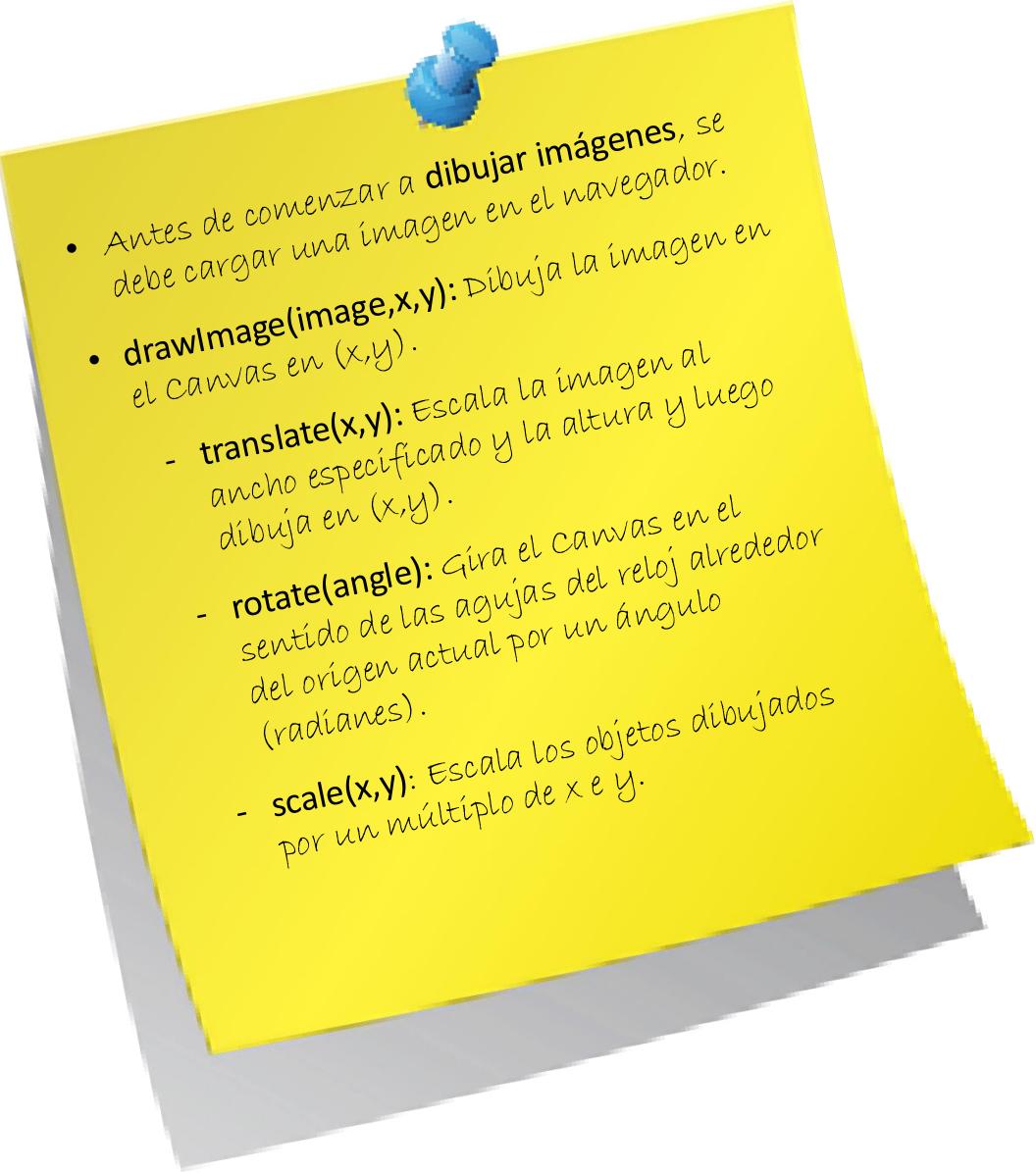
```
132 //TRANSFORMACIÓN  
133  
134 //Transladar el origen a la ubicación del objeto  
135 context.translate(250,370);  
136 //Rotar al rededor del origen 60 grados  
137 context.rotate(Math.PI/3);  
138 context.drawImage(image,0,0,60,50,-30,-25,60,50);  
139 //Restablecer el estado original  
140 context.rotate(-Math.PI/3);  
141 context.translate(-240,-370);  
142  
143 //Transladar el origen a la ubicación del objeto  
144 context.translate(300,370);  
145 //Rotar al rededor del nuevo origen  
146 context.rotate(3*Math.PI/4);  
147 context.drawImage(image,0,0,60,50,-30,-25,60,50);  
148 //Restablecer el estado original  
149 context.rotate(-3*Math.PI/4);  
150 context.translate(-300,-370);
```

1 2





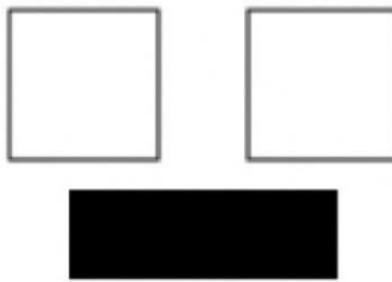
Canvas (14)

- 
- Antes de comenzar a dibujar imágenes, se debe cargar una imagen en el navegador.
 - `drawImage(image,x,y)`: Dibuja la imagen en el canvas en (x,y).
 - `translate(x,y)`: Escala la imagen al ancho especificado y la altura y luego dibuja en (x,y).
 - `rotate(angle)`: Gira el canvas en el sentido de las agujas del reloj alrededor del origen actual por un ángulo (radianes).
 - `scale(x,y)`: Escala los objetos dibujados por un múltiplo de x e y.

Canvas (15)

- Un uso común de estos métodos es **rotar objetos o sprites** al dibujarlos, mediante el siguiente procedimiento:
 1. Trasladar el origen del canvas a la ubicación del objeto.
 2. Girar el canvas por el ángulo deseado.
 3. Dibujar el objeto.
 4. Restaurar el canvas a su estado original.

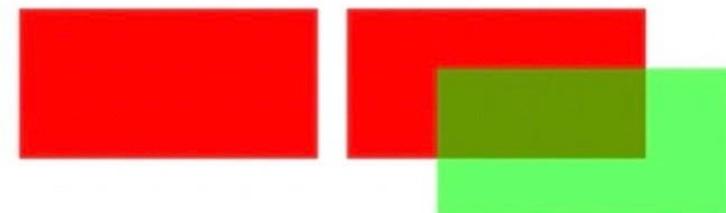
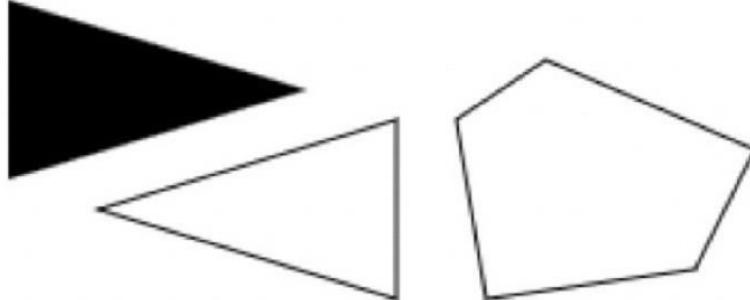
Visualización



This is some text...

This is in 10pt Arial...

This is stroked in 16pt Arial..





Elementos Multimedia

Elemento de Audio

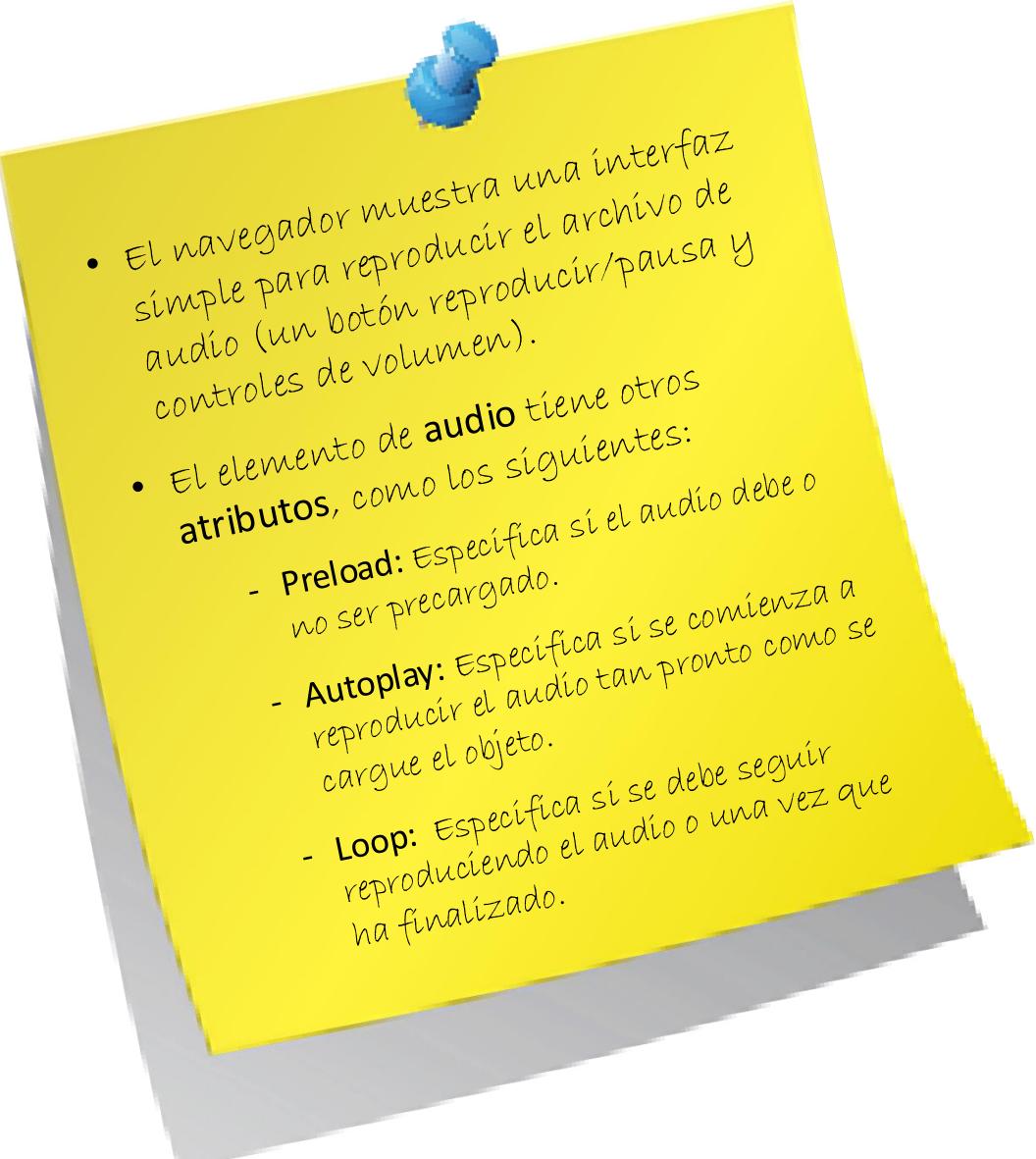


Elemento audio (1)

```
1 <html>
2 <audio src = "music.mp3" controls = "controls">
3 | Tu navegador no soporta la etiqueta Canvas de HTML5. Por favor, utilice otro navegador.
4 </audio>
5 </html>
```

- Para ejecutar audio, muchas páginas solían utilizar **plug-ins** embebidos (Flash).
- El elemento audio se crea en HTML utilizando la etiqueta **<audio>** o en JavaScript utilizando el objeto Audio.

Elemento audio (2)

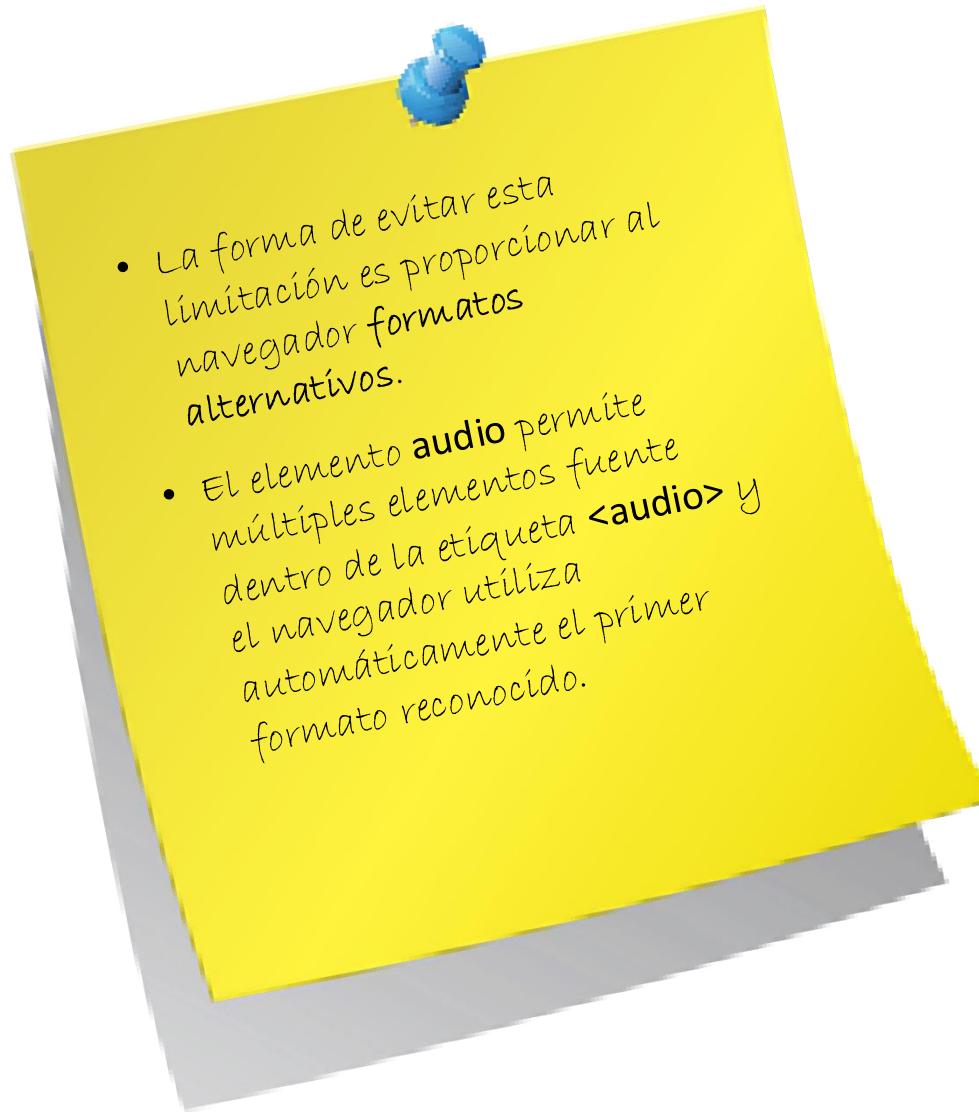
- 
- El navegador muestra una interfaz simple para reproducir el archivo de audio (un botón reproducir/pausa y controles de volumen).
 - El elemento de audio tiene otros atributos, como los siguientes:
 - **preload:** Especifica si el audio debe o no ser precargado.
 - **autoplay:** Especifica si se comienza a reproducir el audio tan pronto como se cargue el objeto.
 - **loop:** Especifica si se debe seguir reproduciendo el audio o una vez que ha finalizado.

Elemento audio (3)

Navegadores	Firefox	Opera	Chrome	Safari	Microsoft Edge
Formatos					
MP3	Sí	Sí	Sí	Sí	Sí
AAC	Sí	Sí	Sí	Sí	Sí
Ogg Vorbis	Sí	Sí	Sí	No	Sí
Opus	Sí	Sí	Sí	Sí	Sí
FLAC	Sí	Sí	Sí	No	Sí
WAV	Sí	Sí	Sí	Sí	Sí

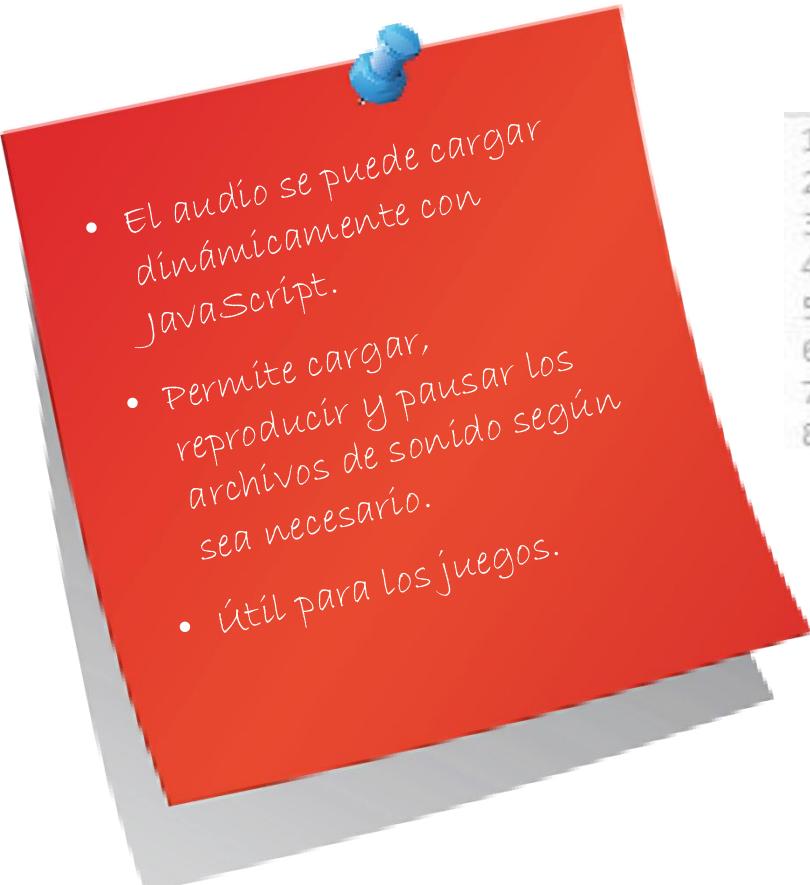
- 
- Formatos compatibles con navegadores:
 - **MP3** (MPEG Audio Layer).
 - **WAV** (Waveform Audio).
 - **OGG** (Ogg Vorbis).

Elemento audio (4)



Elemento audio (5)

```
1 <audio controls = "controls">
2   <source src = "music.ogg" type = "audio/ogg" />
3   <source src = "music.mp3" type = "audio/mpeg" />
4   | Tu navegador no soporta la etiqueta Canvas de HTML5. Por favor, utilice otro navegador.
5 </audio>
```

- 
- El audio se puede cargar dinámicamente con JavaScript.
 - Permite cargar, reproducir y pausar los archivos de sonido según sea necesario.
 - útil para los juegos.

```
1 <script>
2 //Crear un nuevo object Audio
3 var sound = new Audio();
4 // Seleccionar la fuente del sonido
5 sound.src = "music.ogg";
6 // Ejecutar el sonido
7 sound.play();
8 </script>
```

Elemento audio (6)

- Al igual que con la etiqueta HTML `<audio>`, se necesita una forma de detectar qué formato admite el navegador y cargar el formato apropiado.
- El objeto `Audio` proporciona un método llamado `canPlayType()` que devuelve valores de "", "tal vez" o "probablemente" para indicar compatibilidad con un códec específico.
- Podemos usar esto para crear una verificación simple y cargar el formato de audio apropiado.

Elemento audio (7)

```
1 <script>
2 var audio = document.createElement('audio');
3 var mp3Support,oggSupport;
4 if (audio.canPlayType) {
5 // Actualmente canPlayType () devuelve: "", "tal vez" o "probablemente"
6 mp3Support = "" != myAudio.canPlayType('audio/mpeg');
7 oggSupport = "" != myAudio.canPlayType('audio/ogg; codecs = "vorbis"');
8 } else {
9 //La etiqueta de audio no es compatible
10 mp3Support = false;
11 oggSupport = false;
12 }
13 // Comprueba si hay ogg, luego mp3, y finalmente configura soundFileExtn como indefinido
14 var soundFileExtn = oggSupport?"ogg":mp3Support?"mp3":undefined;
15 if(soundFileExtn) {
16 var sound = new Audio();
17 // Carga el archivo de sonido con la extensión detectada
18 sound.src = "bounce" + soundFileExtn;
19 sound.play();
20 }
21 </script>
```

Elemento audio (8)

```
1 <script>
2     if(soundFileExtn) {
3         var sound = new Audio();
4         sound.addEventListener('canplaythrough', function(){
5             alert('loaded');
6             sound.play();
7         });
8         // Cargar el archivo de sonido con la extensión detectada
9         sound.src = "bounce"+soundFileExtn;
10    }
11 </script>
```

- 
- El objeto Audio desencadena un evento llamado **canplaythrough** cuando el archivo está listo para ser reproducido.
 - Se puede utilizar esto para diseñar un **preload** de audio que cargará todos los recursos del juego antes de que comience.

Elemento Imagen



Elemento imagen (1)

- El elemento `image` permite mostrar imágenes dentro de un archivo HTML. La forma más sencilla de hacerlo es mediante el uso de la etiqueta `<image>` y la especificación de un atributo `src`.

```
<img src = 'spaceship.png' id = 'spaceship' >
```

- También puede cargar una imagen de forma dinámica utilizando JavaScript instanciando un nuevo objeto de `Image` y configurando su propiedad `src`.

```
var image = new Image();
image.src = 'spaceship.png';
```

- Se puede utilizar cualquiera de estos métodos para obtener una imagen para dibujar en un Canvas.

Elemento imagen (2)

- Cargar todas las imágenes antes de que comience el juego.
- Barra de progreso que muestra el porcentaje de imágenes cargadas.

```
image.onload = function() {
    alert('La imagen terminó de cargarse');
};
```

Elemento imagen (3)

- Usando el evento onload, podemos crear un cargador de imágenes simple que rastrea imágenes cargadas hasta el momento.

```
var imageLoader = {
    loaded:true,
    loadedImages:0,
    totalImages:0,
    load:function(url){
        this.totalImages++;
        this.loaded = false;
        var image = new Image();
        image.src = url;
        image.onload = function(){
            imageLoader.loadedImages++;
            if(imageLoader.loadedImages == imageLoader.totalImages){
                imageLoader.loaded = true;
            }
        }
        return image;
    }
}
```

Elemento imagen (4)

```
// Dibujar una imagen cargada de forma individual  
//Primero (Cargar imágenes individuales y almacenarlas en un array grande.  
//Argumentos: el elemento, las coordenadas (x,y) del destino.  
  
var image = imageArray[imageNumber];  
context.drawImage(image,x,y);  
  
//Dibujar una imagen cargada en un Sprite Sheet  
// Primero: (Cargar la imagen sprite sheet)  
// Argumentos: el elemento, las coordenadas (x,y) de la fuente,  
// El ancho y alto (para recortarla),  
// Las coordenadas (x,y) del destino, y  
// El alto y ancho del destino (para ajustar el tamaño).  
  
context.drawImage (this.spriteImage, this.imageWidth*(imageNumber), 0, this.imageWidth,  
this.imageHeight, x, y, this.imageWidth, this.imageHeight);
```

- Las **Sprite sheets** almacenan todos los **sprites** (imágenes) para un objeto en una sola imagen.
- Al mostrar las imágenes, se calcula el inicio del Sprite que se quiere mostrar y se utiliza el método **drawImage()** para dibujar solo una parte de una imagen, en este caso la imagen **spaceship.png**.

Elemento imagen (5)



Animación

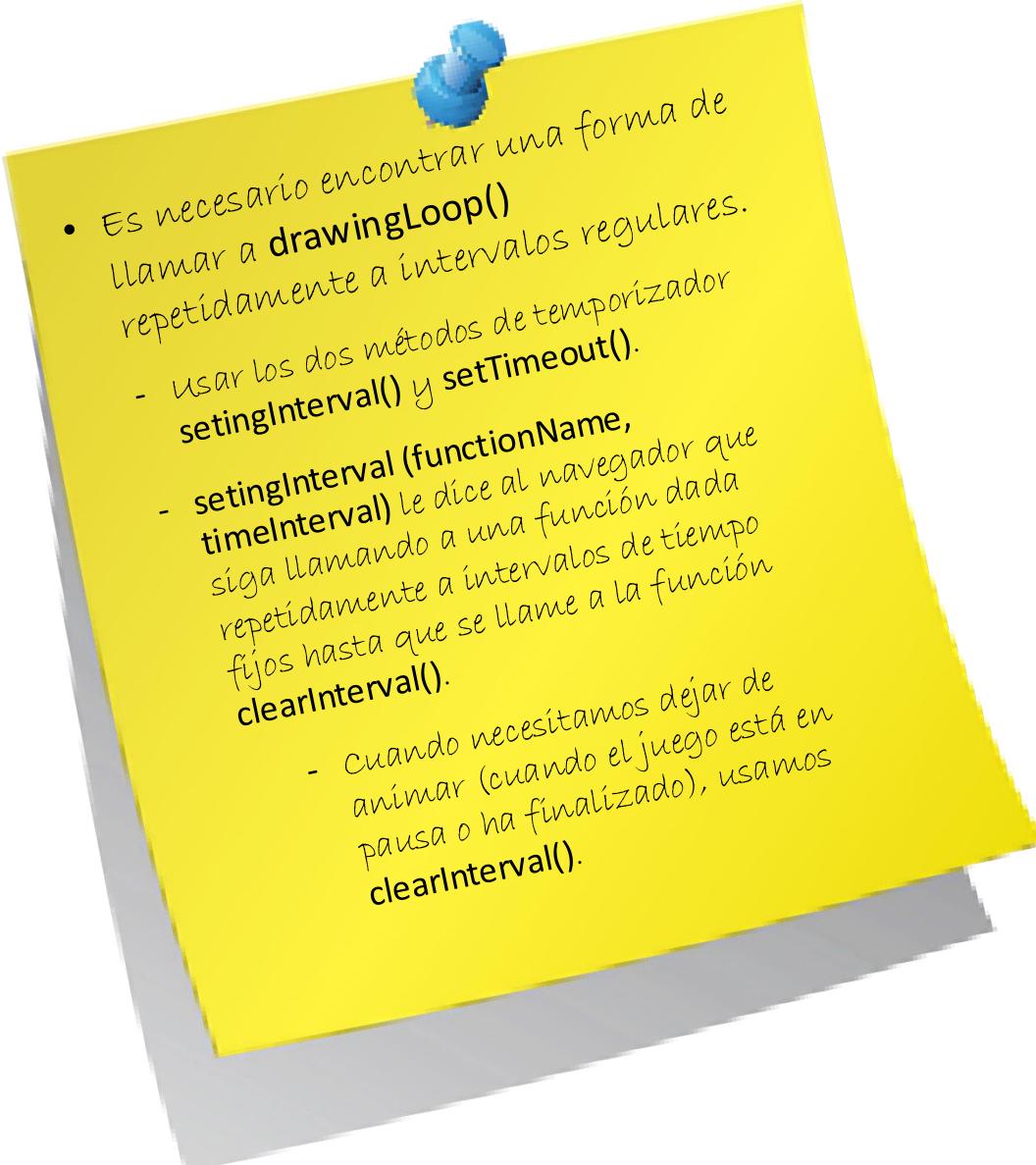


Animación (1)

```
// Animación típica y bucle de animación
function animationLoop(){
    // Itera a través de todos los elementos del juego y los mueve
}
function drawingLoop(){
    //1. Limpia el Canvas
    //2. Itera a través de todos los elementos
    //3. Y dibuja cada elemento
}
```

- **Animar** es simplemente una cuestión de dibujar un objeto, borrarlo y dibujarlo nuevamente en una nueva posición.
 - Manteniendo una función de dibujos que **se llama varias veces por segundo**.
 - En algunos juegos, también hay una función de control/animación.

Animación (2)

- 
- Es necesario encontrar una forma de llamar a `drawingLoop()` repetidamente a intervalos regulares.
 - usar los dos métodos de temporizador `setInterval()` y `setTimeout()`.
 - `setInterval(functionName, timeInterval)` le dice al navegador que siga llamando a una función dada repetidamente a intervalos de tiempo fijos hasta que se llame a la función `clearInterval()`.
 - cuando necesitamos dejar de animar (cuando el juego está en pausa o ha finalizado), usamos `clearInterval()`.

Animación (3)

```
//Llamando al bucle de dibujo con el intervalo establecido
// Llamar a drawingLoop() cada 20 milisegundos
var gameLoop = setInterval(drawingLoop,20);

// Parar la llamada a drawingLoop() y limpiar la variable gameLoop
clearInterval(gameLoop);
setTimeout(functionName, timeInterval)

// Le dice al navegador que llame a una función determinada una vez después de un
intervalo de tiempo determinado
//Llamando al bucle de dibujo con el tiempo de espera configurado
function drawingLoop(){
    //1. llamar al método drawingLoop una vez después de 20 milisegundos
    var gameLoop = setTimeout(drawingLoop,20);
    //2. Limpiar el canvas
    //3. Iterar a través de todos los elementos
    //4. Y dibujarlos
}
```

- Para dejar de animar (cuando el juego está en pausa o ha finalizado), se puede utilizar clearTimeout():

```
// Dejar de llamar a drawingLoop() y borre la variable gameLoop
clearTimeout(gameLoop);
```

requestAnimationFrame





requestAnimationFrame (1)

- API para manejar la animación.
- Optimizar el código de la animación en un solo ciclo de recarga y repintado, lo que da como resultado una animación más suave.
- Pausar la animación cuando la pestaña no está visible, lo que reduce el uso de CPU y GPU.
- Limitar automáticamente la velocidad de fotogramas.

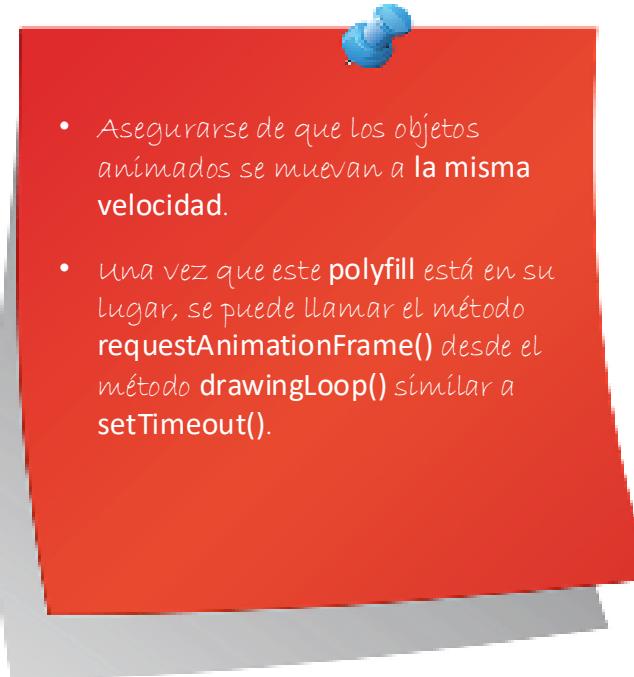
requestAnimationFrame (2)

```
//Una solicitud simple AnimationFramePolyfill
(function() {
var lastTime = 0;
var vendors = ['ms', 'moz', 'webkit', 'o'];
for(var x = 0; x<vendors.length && !window.requestAnimationFrame; ++x) {
    window.requestAnimationFrame = window[vendors[x] + 'RequestAnimationFrame'];
    window.cancelAnimationFrame =
        window[vendors[x] + 'CancelAnimationFrame'] ||
        window[vendors[x] + 'CancelRequestAnimationFrame'];
}
if (!window.requestAnimationFrame)
    window.requestAnimationFrame = function(callback, element) {
        var currTime = new Date().getTime();
        var timeToCall = Math.max(0, 16 - (currTime - lastTime));
        var id = window.setTimeout(function() { callback(currTime+timeToCall); }, timeToCall);
        lastTime = currTime+timeToCall;
        return id;
};
if (!window.cancelAnimationFrame)
    window.cancelAnimationFrame = function(id) {
        clearTimeout(id);
    };
}());
```

- Los navegadores tienen sus propios nombres para los métodos en la API:
 - msRequestAnimationFrame (Microsoft).
 - mozRequestAnimationFrame (Mozilla).

requestAnimationFrame (3)

```
//Llamar a Drawing Loop con requestAnimationFrame
function drawingLoop(nowTime){
    //1. Llama al método drawingLoop cada vez que el navegador está listo para dibujar de nuevo
var gameLoop = requestAnimationFrame(drawingLoop);
    //2. Limpiar el Canvas
    //3. Iterar a través de todos los elementos
    //4. Opcionalmente utilizar nowTime y last nowTime para interpolar frames
    //5. Y dibujarlos
}
```

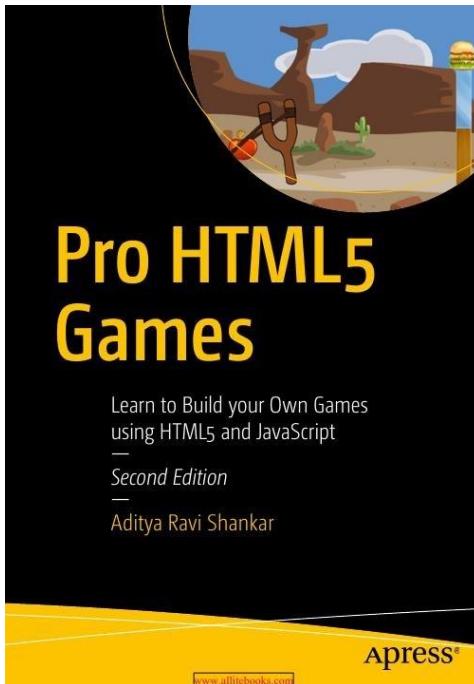
- 
- Asegurarse de que los objetos animados se muevan a la misma velocidad.
 - Una vez que este **polyfill** está en su lugar, se puede llamar el método **requestAnimationFrame()** desde el método **drawingLoop()** similar a **setTimeout()**.

requestAnimationFrame (4)

- Para dejar de animar (cuando el juego está en pausa o ha finalizado), se utiliza el método cancelAnimationFrame()

```
// Detener la llamada a drawingLoop() y Limpiar la variable gameLoop  
cancelAnimationFrame(gameLoop);
```

Bibliografía



- Shankar, A. (2017). *Pro HTML5 Games: Learn to Build your Own Games using HTML5 and JavaScript.* 10.1007/978-1-4842-2910-1.
- MDN Web Docs. (2005-2021). Canvas API. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
- Pixabay. (2025, 10 de enero). Pixabay. <https://pixabay.com/es/>
- Genially. (2025, 10 de enero). Genially. <https://genially.education/>