03-11 interacciónes SVG y Transformaciones

Nombre		Curso	
Apellidos		Fecha	

1. interacciónes SVG

Hay tres formas de interactuar con un svg

- Atributo <a href> para insertar un enlace
- Los atributos Begin/end asociados a una animaciones
- Los atributos on____ asociados a eventos y JavaScripts

1. Atributo <a href> para insertar un enlace

Se coloca antes y despues de la forma a clicar

```
<!-- Ejemplo  1 href -->
<svg viewBox="0 0 200 100" xmlns="http://www.w3.org/2000/svg"
width="200" height="200" >

 <path
  fill="none"
  stroke="lightgrey"
  d="M20,50 C20,-50 180,150 180,50 C180-50 20,150 20,50 z" />

  <a href="https://es.wikipedia.org/wiki/Wikipedia:Portada">
  <circle r="5" fill="red">
  <animateMotion
   dur="10s"
   repeatCount="indefinite"
   path="M20,50 C20,-50 180,150 180,50 C180-50 20,150 20,50 z" />
  </circle>
  </a>
</svg>
```

2. Los Atributos Begin / End

El atributo de inicio define cuándo debe comenzar una animación o cuándo debe descartarse un elemento.

El valor del atributo es una lista de valores separados por punto y coma. La interpretación de una

lista de horas de inicio se detalla en la especificación SMIL en "Evaluación de listas de horas de inicio y finalización".

Cada valor individual puede ser uno de los siguientes: <offset-value>, <syncbase-value>, <event-value>, <repeat-value>, <accessKey-value>, <wallclock-sync-value> o la palabra clave indefinido.

Puede usar este atributo con los siguientes elementos SVG:

- <animate>
- <animateMotion>
- <animateTransform>

2.1 Valor de tiempo absoluto Begin="0s"

Este valor define un valor de reloj que representa un punto en el tiempo relativo al comienzo del documento SVG (generalmente el evento de carga o DOMContentLoaded). Los valores negativos son válidos.

Ejemplo

Begin="2s" empieza se ha cargado dos segundos el objeto en la web

2.2 Valor de tiempo sincronizado con otro elemento

La hora de inicio de la animación del elemento se define en relación con el inicio o el final activo de otra animación identificado por el id del otro elemento

Un valor de base de sincronización válido consiste en una referencia de ID a otro elemento de animación seguido de un punto y un comienzo o un final para identificar si sincronizar con el comienzo o el final activo del elemento de animación al que se hace referencia. Se puede agregar un valor de compensación opcional como se define en <offset-value>.

Ejemplo

Begin="animaciondelelementoconid1.end" empieza tras la animación del elemento con id1

OJO COMO UN PINO el id debe ir dentro del animation

```
<!-- Ejemplo  2 Begin -->
<svg width="120"  height="120"  viewBox="0 0 120 120"  xmlns="http://www.w3.org/2000/svg"  version="1.1">
        <rect x="10" y="35" height="15" width="0">
          <animate  attributeType="XML"      attributeName="width" id="rectangulo1"
            to="100"     dur="8s"      fill="freeze"      begin="2s"/>
        </rect>

        <rect x="35" y="60" height="15" width="0">
          <animate     attributeType="XML"     attributeName="width" id="rectangulo2"
            to="75"      dur="6s"     fill="freeze"
            begin="rectangulo1.end"   />
        </rect>
```

```
        <rect x="60" y="85" height="15" width="0">
          <animate      attributeType="XML"      attributeName="width" id="rectangulo3"
            to="50"      dur="4s"     fill="freeze"
           begin="rectangulo2.end"  />
        </rect>
</svg>
```

2.3 Begin="<valor-evento>"

Este valor define un evento y un desplazamiento opcional que determina el momento en que debe comenzar la animación del elemento. La hora de inicio de la animación se define en relación con la hora en que se activa el evento especificado.

Un valor de evento válido consta de un ID de elemento seguido de un punto y uno de los eventos admitidos para ese elemento. Todos los eventos válidos (no necesariamente admitidos por todos los elementos) están definidos por las especificaciones DOM y HTML. Esos son:

- **Eventos de Seleccion (focus):** focus,blur,focusin,focusout,
  Eventos de click al raton : auxclick,click,dblclick,
- **Eventos del raton:** mousedown,mouseenter, mouseleave,mousemove, mouseout, mouseover, mouseup, wheel,
  Eventos del teclado: beforeinput,input,keydown,keyup, compositionstart, compositionupdate , compositionend, load,unload, abort, error, select ,resize, scroll, beginEvent, endEvent, repeatEvent

Begin="wheel" cuando se mueva la rueda del raton
Begin="click" cuando se haga clic en el raton
Begin="mouseenter" cuando se pase por el raton encima
Begin="keydown(e) " cuando se pulsa la tecla "e"

<!-- Ejemplo 3 Eventos -->

```
<svg  width="120"  height="120"  viewBox="0 0 120 120"
xmlns="http://www.w3.org/2000/svg"  version="1.1"  xmlns:xlink="http://www.w3.org/1999/xlink">

 <rect x="10" y="35" height="15" width="0">
  <animate      attributeType="XML"      attributeName="width"
                  from="0"      to="100"      dur="8s"      fill="freeze"
                    begin="startButton.click"     />
 </rect>

 <rect   id="startButton"     style="cursor:pointer;"     x="19.5"     y="62.5"     rx="5"
          height="25"     width="80"
          fill="#EFEFEF"     stroke="black"     stroke-width="1" />
 <text x="60" y="80" text-anchor="middle" style="pointer-events:none;">   Click me.  </text>
</svg>
```

```
<!-- Ejemplo  4 Eventos -->
<svg width="270" height="290">

<rect x=10 y=10 width=80 height=60
   fill="yellow" stroke="navy" stroke-width="3">
   <animate  id="r1" attributeType="XML" attributeName="x"
     values="10;170;10" dur="2s" begin="0s;r4.end+2s"/>
</rect>
<rect  x=10 y=80 width=80 height=60
   fill="red" stroke="navy" stroke-width="3">
   <animate id="r2" attributeType="XML" attributeName="x"
     values="10;170;10" dur="2s" begin="r1.end"/>
</rect>
<rect  x=10 y=150 width=80 height=60
   fill="green" stroke="navy" stroke-width="3">
   <animate id="r3" attributeType="XML" attributeName="x"
     values="10;170;10" dur="2s" begin="r2.end"/>
</rect>
<rect  x=10 y=220 width=80 height=60
   fill="blue" stroke="navy" stroke-width="3">
   <animate id="r4" attributeType="XML" attributeName="x"
     values="10;170;10" dur="2s" begin="r3.end"/>
</rect>
</svg>
```

3. Los atributos de evento (svg + javascript)

Los atributos de evento siempre tienen su nombre que comienza con "on" seguido del nombre del evento al que están destinados. Especifican algún script para ejecutar cuando el evento del tipo dado se envía al elemento en el que se especifican los atributos.

Para cada tipo de evento que admite el navegador, SVG lo admite como un atributo de evento, siguiendo los mismos requisitos que para los atributos de evento HTML.

Los atributos de eventos globales están disponibles en todos los elementos SVG. Otros atributos de eventos están disponibles caso por caso para cada elemento.

Animation Event Attributes: onbegin, onend, onrepeat

Document Event Attributes : onabort, onerror, onresize, onscroll, onunload

Document Element Event Attributes : oncopy, oncut, onpaste

Global Event Attributes: oncancel, oncanplay, oncanplaythrough, onchange, onclick, onclose, oncuechange, ondblclick, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, ondurationchange, onemptied, onended, onerror, onfocus, oninput, oninvalid, onkeydown, onkeypress, onkeyup, onload, onloadeddata, onloadedmetadata, onloadstart, onmousedown, onmouseenter, onmouseleave, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel, onpause, onplay, onplaying, onprogress, onratechange, onreset, onresize, onscroll, onseeked, onseeking, onselect, onshow, onstalled, onsubmit, onsuspend, ontimeupdate, ontoggle, onvolumechange, onwaiting

Graphical Event Attributes : onactivate, onfocusin, onfocusout

3.1. Formas de insertar JavaScript en un SVG

Existen **tres formas principales**:

A) JavaScript dentro del mismo archivo SVG (inline)

Puedes usar la etiqueta <script> **dentro del SVG**:

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
 <circle id="c1" cx="100" cy="100" r="50" fill="red" />

 <script>
  // Esperar a que el SVG cargue
  <![CDATA[
   const circle = document.getElementById("c1");
   circle.addEventListener("click", () => {
    circle.setAttribute("fill", "blue");
   });
  ]]>
 </script>
</svg>
```

Nota: En algunos navegadores, el contenido del script debe ir dentro de <![CDATA[ ]]> para evitar conflictos con XML.

B) Cargar JavaScript desde un archivo externo

```
<svg width="200" height="200"
    xmlns="http://www.w3.org/2000/svg">
 <script xlink:href="script.js"></script>

 <circle id="c1" cx="100" cy="100" r="50" fill="red"/>
</svg>
```

En script.js:

```
document.getElementById("c1").setAttribute("fill", "green");
```

Funciona sólo si el SVG se carga como documento (<img> NO permite scripts).

C) Insertar un SVG en HTML y manipularlo desde JavaScript

```
<svg id="mysvg" width="200" height="200">
 <rect id="r1" x="10" y="10" width="100" height="100" fill="orange"/>
</svg>

<script>
 const rect = document.getElementById("r1");
 rect.setAttribute("fill", "purple");
</script>
```

Esta es la forma **más flexible**.

3.2. Cómo acceder y modificar propiedades de elementos SVG

Los elementos SVG se manipulan igual que en el DOM, pero con atributos especiales.

✔ Usar getElementById

```
const circle = document.getElementById("c1");
```

Cambiar atributos como cx, fill, etc.

```
circle.setAttribute("cx", "150");
circle.setAttribute("fill", "blue");
```

Leer atributos

```
let color = circle.getAttribute("fill");
console.log(color);
```

Usar propiedades especiales (en SVG compatible con SVG DOM)

Algunos elementos tienen propiedades JavaScript nativas:

```
circle.cx.baseVal.value = 120;
circle.r.baseVal.value = 40;
```

Resumen rápido

Acción	Cómo se hace
Insertar JS en el SVG	<script>...</script> dentro del SVG
Manipular desde fuera	<svg id="..."> y DOM normal
Cambiar atributos	setAttribute()
Acceder a propiedades SVG	element.cx.baseVal.value

```
<!-- Ejemplo  5 SVG JS-->

<svg viewBox="0 0 200 200" xmlns="http://www.w3.org/2000/svg">
  <circle   cx="100"    cy="100"    r="100"
    onclick="alert('You have clicked the circle.')" />
</svg>

<!-- Ejemplo  6  SVG JS-->
<svg xmlns='http://www.w3.org/2000/svg' version='1.1' height='600' width='820'>

 <script type="text/ecmascript"><![CDATA[
     function changerect(evt) {
```

```
    var svgobj=evt.target;
    svgobj.style.opacity= 0.3;
    svgobj.setAttribute ('x', 300);
   }
  ]]>
 </script>

<!-- Ejemplo 7 SVG JS-→

<svg width="300"  xmlns="http://www.w3.org/2000/svg"    xmlns:xlink="http://www.w3.org/1999/xlink">

<!-- Un círculo →  <circle id="c1" cx="80" cy="100" r="40" fill="tomato" />
<!-- Un rectángulo →  <rect id="r1" x="150" y="60" width="80" height="80" fill="steelblue" />

<script type="text/ecmascript">  <![CDATA[

  // Esperar a que exista el DOM del SVG
  window.addEventListener("load", () => {
            const circle = document.getElementById("c1");
            const rect   = document.getElementById("r1");

            // Cambiar color al hacer click
            circle.addEventListener("click", () => {
                  circle.setAttribute("fill", "gold");              });

            // Mover rectángulo al acercar el mouse
                  rect.addEventListener("mouseover", () => {
                        rect.setAttribute("x", parseInt(rect.getAttribute("x")) + 10);     });

    // Animación sencilla usando propiedades del SVG DOM
    let grow = true;
            setInterval(() => {
                  let r = circle.r.baseVal.value;
                  if (grow) r++; else r--;
                  if (r > 50) grow = false;
                  if (r < 30) grow = true;
                  circle.r.baseVal.value = r;
            }, 30);              });

          ]]>  </script>
</svg>
```

2. Funciones de movimiento & Transformaciones SVG

En SVG, existe un atributo denominado **transform** mediante el cuál se pueden hacer transformaciones a las formas, figuras y trazos dibujados. Las transformaciones SVG son muy similares a las transformaciones CSS, sin embargo tienen algunas diferencias sustanciales.

Estas son las funciones que podemos utilizar para realizar transformaciones:

Función	Descripción
translate(x y)	Traslada (mueve) el elemento x unidades en horizontal, y y unidades en vertical.
rotate(a x y)	Rota el elemento a grados, utilizando el punto de origen (x, y).
scale(x y)	Escala el elemento x unidades en horizontal y y unidades en vertical.
scale(s)	Escala el elemento s unidades en horizontal y vertical.
skewX(x)	Deforma el elemento x unidades en horizontal.
skewY(y)	Deforma el elemento y unidades en vertical.

> Una diferencia importante respecto a HTML/CSS, es que en SVG los valores por parámetro no se separan por comas, ni se utilizan unidades como px, ya que indicamos las unidades del viewbox.

Observa la siguiente imagen SVG con fondo gris, donde tenemos un cuadrado color violeta del tamaño de la mitad del lienzo, colocado al inicio.

```
<svg viewBox="0 0 100 100" height="300" style="background:grey">
  <rect x="0" y="0" width="50" height="50" fill="indigo">
</svg>
```

En los siguientes ejemplos, vamos a añadirle un atributo transform al elemento <rect> para realizar transformaciones. En las demostraciones interactivas puedes variar los valores de la función de transformación y observar la diferencia visual.

2.1 La función translate()

Mediante la transformación translate() podemos realizar movimientos o desplazamientos en horizontal y vertical. Observa que los parámetros, aunque pueden ir separados por comas, es más habitual separarlo únicamente con espacios.

```
<!-- Ejemplo 8 translate-->
<svg viewBox="0 0 100 100" height="300" style="background:grey">
  <rect x="0" y="0" width="50" height="50" fill="indigo" transform="translate(50 50)">
</svg>

<p>Valor de <code>X</code>: <input class="x" type="range" min="-25" max="75" value="50"></p>
<p>Valor de <code>Y</code>: <input class="y" type="range" min="-25" max="75" value="50"></p>
<script>
```

```
const x = document.querySelector(".x");
const y = document.querySelector(".y");
const rect = document.querySelector("rect");
const output = document.querySelector("output");

const update = () => {
  output.value = `${x.value} ${y.value}`;
  rect.setAttribute("transform", `translate(${x.value} ${y.value})`);
}

x.addEventListener("input", () => update());
y.addEventListener("input", () => update());

</script>
```

2.2 La función rotate()

Mediante la función rotate() debemos indicar tres parámetros: el ángulo (*de 0 a 360*), el punto de referencia de x y el punto de referencia de y. El único obligatorio es el ángulo, ya que si no indicamos valores x e y, ambos valdrán 0.

<!-- Ejemplo 9 rotate-->

```
<svg viewBox="0 0 100 100" height="300" style="background:grey">
  <rect x="0" y="0" width="50" height="50" fill="indigo" transform="rotate(45 50 50)">
</svg>

<p>Angulo de rotación:
<input id="a" type="range" min="0" max="360" value="20"></p>
<p>Valor de <code>X</code>: <input id="x" type="range" min="0" max="50" value="50"></p>
<p>Valor de <code>Y</code>: <input id="y" type="range" min="0" max="50" value="50"></p>
<output> </output>
<script>
let  a = document.getElementById("a");
let  x = document.getElementById("x");
let  y = document.getElementById("y");
const rect = document.querySelector("rect");
const output = document.querySelector("output");

const update = () => {
  output.value = `${a.value} ${x.value} ${y.value}`;
  rect.setAttribute("transform", `rotate(${a.value} ${x.value} ${y.value})`);
}

a.addEventListener("input", () => update());
x.addEventListener("input", () => update());
y.addEventListener("input", () => update());
</script>
```

2.3 La función scale()

La transformación scale() nos permite escalar nuestro objeto, y hacerlo más pequeño (*valores inferiores a 1*) o más grande (*valores superiores a 1*). Observa el siguiente ejemplo, que tiene un scale(1) por defecto. Esto significa que el elemento no estará alterado, sino que estará a su tamaño real.

Si indicamos un scale(2), el elemento tendrá el doble de tamaño, si indicamos un scale(0.5) tendrá la mitad de su tamaño real.

<!-- Ejemplo 10 scale-->

```
<svg viewBox="0 0 100 100" height="300" style="background:grey">
  <rect id="rect2" x="0" y="0" width="50" height="50" fill="indigo" transform="scale(1)">
</svg>

<p>Escala: <input class="s" type="range" min="0" max="2" step="0.1" value="1"></p>
<output id="o2"> </output>
<script>
const s = document.querySelector(".s");
const rect2 = document.getElementById("rect2");;
const output2 = document.getElementById("o2");

const update2 = () => {
  output2.value = `${s.value}`;
  rect2.setAttribute("transform", `scale(${s.value})`);
}

s.addEventListener("input", () => update2());
</script>
```

También es posible indicar dos parámetros, de modo que si indicamos scale(1 2), significa que el elemento tendrá el mismo tamaño en horizontal, pero el doble en vertical.

2.4 La función skewX() / skewY()

las funciones de transformación skewX() y skewY() permiten deformar o torcer un elemento. Juega un poco con los controles de la demo y observarás el resultado de deformar elementos en los diferentes ejes.

<!-- Ejemplo 11 skewXskewY -->

```
<svg viewBox="0 0 300 300" height="300" style="background:grey">
  <rect id="srect" x="50" y="50" width="150" height="150" fill="indigo" >
</svg>

<p>Valor de <code> SkewX</code>: <input id="sx" type="range" min="0" max="50" value="0"></p>
<p>Valor de <code>SkewY</code>: <input id="sy" type="range" min="0" max="50" value="0"></p>
<output id="sout">  </output>
<script>
let  sx = document.getElementById("sx");
let  sy = document.getElementById("sy");
let  srect = document.getElementById("srect");
let  soutput = document.getElementById("sout");
```

```
soutput.value = `${sx.value} ${sy.value}`;
const update4 = () => {
   soutput.value = `${sx.value} ${sy.value}`;
   srect.setAttribute("transform", `skewX(${sx.value}) skewY(${sy.value})`);
}

sx.addEventListener("input", () => update4());
sy.addEventListener("input", () => update4());
</script>
```

2.5 Transformaciones múltiples

Por último,Como habrás visto, ten en cuenta que es posible aplicar múltiples transformaciones simplemente separando por espacio las funciones de transformación.

Practica

Ejercicio 1. Crea cada uno de los ejemplos que aparecen en la practica

Ejercicio 2 . Crea un svg con 1 ellipse con los mismos focos que cuando se pulsen en ellas cree una animación que intercambie el rx y el ry

Ejercicio 2bis . Crea un svg con 5 ellipses con los mismos focos que cuando se pulsen en ellas crezcan de manera diferente. animate + radios

Ejercicio 3. Modifica el anterior svg para que al pasar el ratón encima de cada forma se mueva hacia la izquierda

Ejercicio 4. Modifica el anterior svg para que al hacerle click cada forma rote sobre si misma

Ejercicio 5. crea 4 círculos que se empiecen a mover en los tiempos 0,2s, 4s, 6s solo se deben estar moviendo cada uno de ellos por lo que debes usar end y begin

para borrarlos al final debes usar

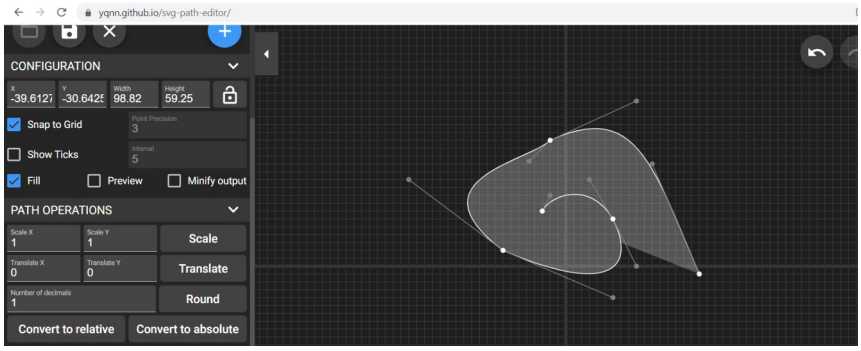
```
<set begin="4s" attributeName="visibility" to="hidden"/>
```

Ejercicio 6. Crea un rectángulo con un javascript asociado a el que cuando se pulse se ponga medio transparente y salte un alert.

Ejercicio 7. Crea dos rectangulos verdes y morados y un texto en enmedio que al pulsar sobre el se mueva girando sobre el mismo

Ejercicio 8. Usa https://yqnn.github.io/svg-path-editor/ para generar una ruta chula y modifica el ejemplo de la pelota para que la siga Se debe iniciar al hacer click y parar al hacer doble clik. Ejemplo de path

M -3 -7 C -2 -9 3 -11 6 -6 C 9 0 6 4 -8 -2 C -20 -11 -4.6667 -13.3333 -2 -16 C 9 -21 11 -13 17 1



Ejercicio 9. Modifica los ejemplos 9,10 y 11 y para tener todos los controles (rotate, transform, skew) de la caja en un mismo ejemplo