SVG, Transiciones y Animaciones (Parte I)

Bienvenidos a la experiencia

Como hemos hablado durante todo el curso, la experiencia del usuario es un componente muy importante dentro de cualquier proyecto web, ya que tomar en serio estas necesidades y objetivos nos permitirá crear maquetas que, en primera instancia, cumplan con las expectativas del consumidor y, que en segunda instancia, produzcan beneficio monetario a nuestro cliente.

Teniendo en cuenta lo anterior, en esta sección revisaremos algunas tecnologías y propiedades que nos ayudarán a mejorar la usabilidad e interacción que tiene el usuario con la interfaz.

Para ello comenzaremos conociendo un archivo de imagen muy particular llamada SVG que nos permitirá escalar y transformar su resolución sin afectar a la nitidez de esta, aún cuándo el tamaño del dispositivo sea muy grande o pequeño.

Después aprenderemos a mejorar la interacción de los elementos usando transiciones de CSS a modo de definir los estados que tendrá un elemento al interactuar con un usuario.

Y finalmente, conoceremos y crearemos animaciones usando CSS.

En resumen, el conocer y poner en práctica estas tecnologías nos permitirán dos cosas:

- Primero, podremos hacer que nuestras maquetas sean interactivas y usables a modo asegurar de que el usuario pueda lograr su objetivos y nuestro cliente también.
- Y segundo, por que estás herramientas nos permitirán conocer aspectos que pasábamos desapercibidos y que ahora se transformarán en competencias profesionales importantes.

¿Qué es SVG?

Uno de los primeros temas que abordaremos en este módulo serán las imágenes SVG.

Comencemos conociendo qué es una imagen SVG:

¿Qué es una imagen SVG?

Las imágenes SVG, son literalmente imágenes vectorizadas que se pueden escalar. En palabras simples las imágenes vectorizadas son imágenes que contienen diferentes objetos geométricos que en su conjunto forman un diseño.

Los SVG aunque suenen como un archivo difícil de usar o trabajar es común y de hecho es probable que lo hayamos visto sin darnos cuenta, en un sitio web, en una taza o hasta en la portada de un libro, debido a que es uno de los formatos más usados por diseñadores gráficos para crear diferentes piezas de diseño por su facilidad y manejo.

¿Por qué usar un SVG?

Entonces, ¿por qué nos sería útil este formato? Pues bien, SVG es una es una tecnología, diferente a otras como `.png` o `.jpg`, debido a que:

- Podremos escalar estos archivos sin perder nitidez
    - Esto quiere decir que podremos definir un tamaño de `5000px` por `4500px` y aún así mantener la misma resolución en la imagen
- SVG no tiene problemas de resolución
    - Por lo tanto si vemos una imagen SVG en diferentes dispositivos esta se verá igual
- Su peso es menor que formatos similares como `.png`
- Podemos personalizar su contenido usando CSS
    - Este punto es el más interesante para nosotros, con un SVG podremos hacer muchas cosas como cambiar su color, tamaño, animarlos o definir filtros con CSS.

Estructura de un SVG

Ahora bien, para poder lograr lo mencionado anteriormente deberemos conocer la estructura de un SVG.

Hasta ahora sabemos que un SVG es comúnmente usado por diseñadores gráficos los cuales a su vez utilizan programas especializados editar de manera *"visual"* vectores como **Ilustrator**, **Inkscape**, **Sketch**, entre otras opciones.

Sin embargo, nosotros como desarrolladores también podremos editar, crear y cambiar elementos dentro de un svg utilizando nuestro viejo y confiable editor de texto.

Para hacerlo usaremos como ejemplo el isotipo anterior de Desafío Latam.

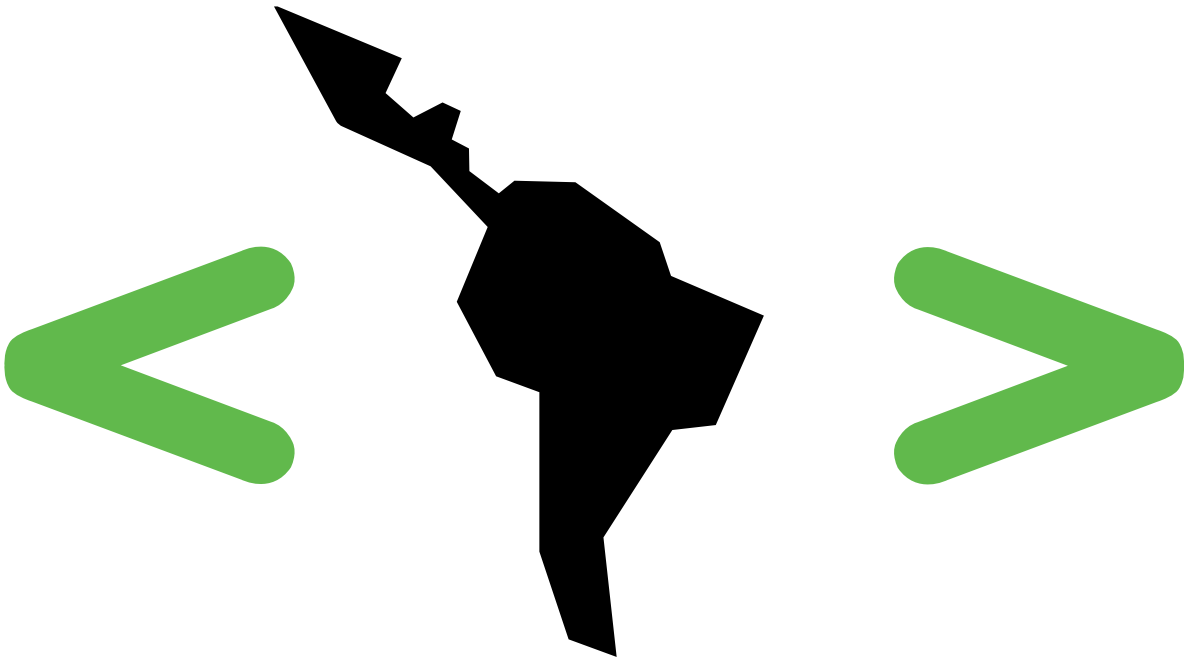

Imagen 1. Logo anterior de Desafío Latam.

> Presiona el botón derecho de tu mouse y presiona la opción "guardar como" para descargar el archivo

Luego de descargar el isotipo nos moveremos este hacia nuestro editor favorito.

Al entrar veremos lo siguiente:

```
<svg xmlns="http://www.w3.org/2000/svg" width="300px" height="400px"
viewBox="0 0 368 203">
  <defs>
    <style>
      .cls-1 {
        fill: #000;
      }

      .cls-2 {
        fill: #61b94c;
      }
    </style>
  </defs>
  <g id="Group_113" data-name="Group 113" transform="translate(-3057.509
-3952)">
    <path id="Path_1" data-name="Path 1" class="cls-1"
d="M392.07,217.823v49.758L401.2,294.96l15,5.5-4.126-37.315,21.5-33.521,13.566-
1.548,15.018-34.162-28.739-12.282-.257-.14L429.64,171l-26.317-18.712-
19.033-.479-4.877,3.941-9.188-6.94-.126-7.064-5.377-2.811,2.815-8.9-5.688-
2.655-9.1,4.69-8.689-7.612,5.064-10.891-38.77-16.152h-
1.127l19.408,35.833a4.473,4.473,0,0,0,2.062,1.688l27.425,12.377,17.817,18.942L
366.285,189.6l12.283,23.254Z" transform="translate(2832.419 3854.582)"/>
    <path id="Path_2" data-name="Path 2" class="cls-2" d="M571.7,194.435s-
1.126-2-6.813-3.876l-65.416-24.41s-9.468-4.782-15.344,3.845c0,0-
2.189,4.218-.594,7.782,0,0,1.874,4.969,6.75,6.47l46.93,17.627L490.282,219.5c-
4.876,1.5-6.75,6.47-6.75,6.47-
1.6,3.563.594,7.781.594,7.781,5.875,8.627,15.344,3.845,15.344,3.845l65.416-
24.41c5.688-1.875,6.813-3.876,6.813-3.876,2.187-3.126,1.8-7.439,1.8-
7.439S573.885,197.561,571.7,194.435Z" transform="translate(2852.258
3862.28)"/>
```

```
    <path id="Path_3" data-name="Path 3" class="cls-2" d="M316.955,219.375l-
46.928-17.628,46.928-17.628c4.875-1.5,6.753-6.47,6.753-6.47,1.593-3.561-.6-
7.781-.6-7.781-5.875-8.627-15.344-3.845-15.344-3.845l-65.416,24.41c-
5.687,1.875-6.814,3.876-6.814,3.876-2.188,3.127-1.794,7.438-
1.794,7.438s-.394,4.313,1.794,7.442c0,0,1.129,2,6.814,3.873l65.416,24.41s9.469
,4.782,15.344-3.843c0,0,2.191-4.221.6-
7.784C323.707,225.846,321.83,220.876,316.955,219.375Z"
transform="translate(2823.794 3862.266)"/>
  </g>
</svg>
```

Todo este código que vemos es literalmente el mismo isotipo pero en código XML.

¿Qué es un XML?

XML o *"Extensible Markup Language"* es una herramienta para guardar y transportar datos mediante etiquetas y valores definidos por los desarrolladores.

Visualmente se parece HTML, especialmente por su que los dos son leguajes de marcas, pero su gran diferencia radica en que HTML esta diseñado para mostrar datos y en que las etiquetas se encuentran definidas por la 3WC, mientras que las etiquetas XML deben ser definidas por el desarrollador.

En concreto, nosotros debemos saber que la estructura de un código está SVG está escrito en XML, es importante, debido a que todas las definiciones que encontremos dentro de un código tendrán un significado definido por la etiqueta.

Conociendo la estructura de un código SVG

Ahora bien, la estructura de código SVG se definen por algunos elementos los cuales pueden aparecer o no dependiendo del editor gráfico del cual se haya exportado:

Declaración XML

En primer lugar, nos podremos encontrar con una declaración XML. Esta declaración permite definir la versión, el tipo de codificación y si el documento se basa en la información de un elemento externo o no.

```
<?xml version = '1.0' encoding = 'UTF-8' standalone = 'no' ?>
```

Declaración DOCTYPE

Asimismo, un SVG también podría contener una declaración DOCTYPE, tal como la tiene HTML, enunciando la versión de SVG estamos usando. Por lo general la versión recomendada a usar en esta declaración es la versión 1.1.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Etiqueta SVG

Después del DOCTYPE SVG nos encontraremos con una etiqueta llamada `<svg>` la cual es muy importante, pues define en base a diversos atributos las declaraciones que vimos anteriormente.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" xml:space="preserve">

    <!-- Elementos de que componen a un SVG -->

</svg>
```

Las definiciones que encontraremos serán:

- **version="":** El primer atributo que veremos llamado `version=""`. Este atributo contiene la versión SVG del archivo.
- **xmlns="":** El segundo atributo hace referencia al *XML namespace*, el cual define un conjunto de nombres de elementos y atributos que podremos usar al interior de la etiqueta `<svg>`.
- **xmlns:xlink="":** El tercer atributo significa *"XML Linking Language"*. Este al igual que `xmlns` permite agregar algunas características relacionadas a la vinculación de elementos, como por ejemplo conectando elementos o usando enlaces externos a otros sitios web.
- **xlmns: space="":** Finalmente nos encontraremos con un último elemento que al igual que los dos últimos atributos nos permite que al interior de una etiqueta `<text>`

Por otra parte, es importante acotar que algunas declaraciones y atributos mencionados anteriormente pueden que no se encuentren en todos los archivos SVG que revises, puesto que algunos editores gráficos como **Sketch** o **Illustrator** están dejando de usar estos para dar paso a la nueva versión de SVG (2.0).

Conociendo los elementos básicos que componen un SVG

Ahora que conocemos las directrices que provocan que un archivo SVG pueda funcionar, cambiaremos de tema haciendo algunos experimentos con el isotipo anterior de Desafio Latam.

En primer lugar, comenzaremos conociendo algunos atributos importantes para cualquier archivo SVG, puesto que definen el tamaño del lienzo, así como también definir el contenido visualizado.

Alto, ancho y viewBox

En la etiqueta podemos agregar atributos que definan el tamaño del lienzo o viewport SVG con `width=""` y `height=""`, pero también podemos definir el contenido del SVG a visualizar usando el atributo `viewBox=""`.

Estas dos acciones aunque parezcan similares tienen diferencias que son importantes de identificar.

Por ejemplo, si borramos el atributo `viewBox` del SVG y cambiamos el alto por `100px`, podremos ver que la imagen ahora se encuentra recortada. Esto sucede debido a que el tamaño de los vectores es mayor al tamaño nominal que agregamos nosotros, tal como si vieramos un paisaje desde una ventana.

```
<svg xmlns="http://www.w3.org/2000/svg" width="100px" height="400px">
  <!-- Código SVG -->
</svg>
```

Imagen 2. Isotipo sin viewBox.

Por otra parte, si mantenemos el mismo los atributos de alto y ancho, y luego devolvemos el atributo `viewBox` a su lugar, veremos que la imagen volvió a verse de manera normal.

Esto sucede debido a que el atributo `viewBox` permite definir los parámetros de visión, así como también el tamaño que tendrá esta visión, o sea que `viewBox` es similar a ver el mismo paisaje pero usando unos binoculares, con estos podremos definir el lugar donde ver y el tamaño con el cual lo deseamos mirar.

```
<svg xmlns="http://www.w3.org/2000/svg" width="100px" height="400px"
viewBox="0 0 368 203">
 <!-- Código SVG -->
</svg>
```

Imagen 3. Isotipo con variación de tamaño.

En definitiva, conocer la estructura base de un código SVG nos permitirá transformar y definir las características que tendrá de manera conscientes y podremos tomar mejores decisiones al momento de usarlos al interior de nuestros proyectos.

Lecturas complementarias

- SVG Tutorial - MDN Web Docs
- Stuff at the Top of an SVG - Peter Nowell

Elementos de un SVG

Ahora que conocemos la estructura y elementos principales de un SVG, revisaremos algunos de los elementos más comunes que podremos encontrar al interior de un SVG.

Para hacerlo utilizaremos como referencia el logo anterior de Desafío Latam que vimos anteriormente.

Bien si movemos esta imagen a nuestro editor de texto preferido podremos ver el código que constituye a este SVG.

```
<svg xmlns="http://www.w3.org/2000/svg" width="300px" height="400px"
viewBox="0 0 368 203">
  <defs>
    <style>
      .cls-1 {
        fill: #000;
      }

      .cls-2 {
        fill: #61b94c;
      }
    </style>
  </defs>
  <g id="Group_113" data-name="Group 113" transform="translate(-3057.509
-3952)">
    <!-- Elementos gráficos -->
  </g>
</svg>
```

Etiquetas básicas de un archivo SVG

El primer elemento que veremos al interior del logo de desafío es una etiqueta `<svg>`. Etiqueta es la encargada de contener, definir y estructurar los elementos que constituyen a una imagen SVG.

```
<svg xmlns="http://www.w3.org/2000/svg" width="300px" height="400px"
viewBox="0 0 368 203">
    <!-- Código SVG -->
</svg>
```

Sin ir más lejos, esto lo podremos ver en el elemento `<svg>` del logo de Desafío. Aquí encontraremos el atributo namespace XML el cual define a todos los elementos al interior de la etiqueta `<svg>`, también veremos el tamaño del viewport SVG definido por los atributos `width` y `height`, y finalmente el atributo `viewBox` que nos permitirá definir la posición y dimensión de la visión que tendrá el usuario del viewport SVG.

Así mismo, al interior de `svg` podremos encontrar otras etiquetas las cuales nos permitirán agrupar, reusar y ordenar los elementos de un SVG.

Agrupando elementos con la etiqueta `<g>`

En primer lugar, hablaremos de la etiqueta `<g>` la cual nos permitirá agrupar varios elementos gráficos en uno.

Si nos fijamos en la etiqueta `<g>` que se encuentra en el logo podremos ver que está tiene varios atributos que son interesantes de conocer.

Por ejemplo, el primero es un `id=""`. Este nos permitirá definir un `id` específico a cualquier elemento, y en general cumple con las mismas reglas de especificidad de un `id` en CSS y HTML. De hecho este `id` lo podremos llamar desde CSS usando estos selectores.

Cambiemos el nombre del `id` por uno que describa mejor el contenido gráfico del grupo. Probemos con `id="logo-desafio"`.

```
<g id="logo-desafio" data-name="Group 113" transform="translate(-3057.509
-3952)">
  <!-- Elementos Gráficos -->
</g>
```

El segundo atributo que veremos es llamado `data-name=""` y nos permitirá describir el contenido del grupo. En este caso el grupo de elementos contiene tres `path`, o sea, tres elementos gráfico que componen al logo, lo cuales podríamos llamar como `Logo Desafio Latam`.

```
<g id="logo-desafio" data-name="Logo Desafio Latam"
transform="translate(-3057.509 -3952)">
  <!-- Elementos Gráficos -->
</g>
```

Finalmente, nos encontraremos con el atributo `transform` que nos permite manipular diferentes aspectos como el sesgar, rotar o escalar el contenido del grupo.

Es importante acotar que en este módulo se centrará en la manipulación de algunos atributos que nos permitan manipular un SVG desde CSS, de modo que sólo nombraremos a los elementos que forman parte de los elementos gráficos.

Elementos de forma

Elemento `<path>`

Ahora bien, si bajamos veremos los elementos `<path>`. El elemento es el elemento más poderoso en la biblioteca SVG de formas básicas. Se puede usar para crear líneas, curvas, arcos y más.

Los `<path>` crean formas complejas combinando múltiples líneas rectas o líneas curvas. Las formas complejas compuestas solo de líneas rectas se pueden crear como polilíneas. Si bien las polilíneas y los trazados pueden crear formas de aspecto similar, las polilíneas requieren muchas líneas rectas pequeñas para simular curvas y no se escalan bien a tamaños más grandes. Una buena comprensión de las rutas es importante al dibujar SVG. Si bien no se recomienda crear rutas complejas utilizando un editor XML o un editor de texto, comprender cómo funcionan permitirá identificar y reparar problemas de visualización en SVG.

El elemento se usa para definir una ruta.

Los siguientes comandos están disponibles para datos de ruta:

```
M = moveto
L = lineto
H = horizontal lineto
V = vertical lineto
C = curveto
S = smooth curveto
Q = quadratic Bézier curve
T = smooth quadratic Bézier curveto
A = elliptical Arc
Z = closepath
```

Nota: Todos los comandos anteriores también se pueden expresar con letras más bajas. Las letras mayúsculas significan una posición absoluta, las minúsculas significan una posición relativamente

```
<path id="Path_1" data-name="Path 1" class="cls-1"
d="M392.07,217.823v49.758L401.2,294.96l15,5.5-4.126-37.315,21.5-33.521,13.566-
1.548,15.018-34.162-28.739-12.282-.257-.14L429.64,171l-26.317-18.712-
19.033-.479-4.877,3.941-9.188-6.94-.126-7.064-5.377-2.811,2.815-8.9-5.688-
2.655-9.1,4.69-8.689-7.612,5.064-10.891-38.77-16.152h-
1.127l19.408,35.833a4.473,4.473,0,0,0,2.062,1.688l27.425,12.377,17.817,18.942L
366.285,189.6l12.283,23.254Z" transform="translate(2832.419 3854.582)"/>
```

Estos elementos se definen con el atributo `d=""`, el cual contiene una serie de instrucciones y parámetros que permiten definir la forma que tendrá el elemento gráfico.

Además, en estos elementos también podemos agregar selectores como `id` o `class` para poder modificar los estilos que componen a esta forma, lo que es muy útil si deseamos cambiar un color o un tamaño dentro de una forma específica.

Cambiemos las clases e id's de las formas a modo de hacer más fácil de entender el código SVG. Cambiemos el valor de los atributos `id` y `data-name` por `latam-map` y la clase como `shape-latam`, ya que esta la usaremos luego para cambiar el estilo de nuestras formas.

```
<path id="latam-map" data-name="Latam Map" class="shape-latam"
d="M392.07,217.823v49.758L401.2,294.96l15,5.5-4.126-37.315,21.5-33.521,13.566-
1.548,15.018-34.162-28.739-12.282-.257-.14L429.64,171l-26.317-18.712-
19.033-.479-4.877,3.941-9.188-6.94-.126-7.064-5.377-2.811,2.815-8.9-5.688-
2.655-9.1,4.69-8.689-7.612,5.064-10.891-38.77-16.152h-
1.127l19.408,35.833a4.473,4.473,0,0,0,2.062,1.688l27.425,12.377,17.817,18.942L
366.285,189.6l12.283,23.254Z" transform="translate(2832.419 3854.582)"/>
```

Haremos lo mismo pero esta vez con los otros dos elementos, los cuales forman los dos chevrones que encierran al mapa de américa latina. Para poder saber cual es el derecho y el izquierdo usaremos la táctica de comentar el elemento.

Comentemos la segunda forma para ver cual es:

Imagen 4. Probando el chevron.

En este caso, el segundo elemento gráfico es el chevron del lado derecho.

Bien, ahora movamos el chevron derecho hacia abajo y después de eso cambiemos el `id` y `data-name` de las dos formas por un nombre que defina el posición del chevron dentro del logo, como por ejemplo `left-chevro` y `right-chevron`. Asimismo podemos definir el tamaño una clase en conjunto con nombre `shape-chevron`.

```
<g id="logo-desafio" data-name="Logo Desafio Latam"
transform="translate(-3057.509 -3952)">
    <path id="latam-map" data-name="Latam Map" class="shape-latam" d="muchos
comandos" transform="translate(2832.419 3854.582)"/>
    <path id="left-chevron" data-name="Left chevron" class="shape-chevron"
d="muchos comandos" transform="translate(2823.794 3862.266)"/>
    <path id="right-chevron" data-name="Right chevron" class="shape-chevron"
d="muchos comandos" transform="translate(2852.258 3862.28)"/>
  </g>
```

Si miramos el código ahora podremos entender de manera simple el contenido de cada forma que compone a esta imagen SVG.

Elemento `<rect>`

Además del elemento `<path>`, tenemos a nuestra disposición también el elemento `<rect>`, el cual nos permite representar un rectángulo (o alguna variación de éstos) dentro de SVG.

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 400 200" width="400"
height="200">
    <defs>
        <style>
            .rectangulo{
                fill: rgb(0,0,255);
            }
        </style>
    </defs>
    <rect width="400" height="200" class="rectangulo" />
</svg>
```

Como podemos ver, para definir el ancho y alto del rectángulo, hacemos uso de las propiedades **width** y **height** respectivamente. También es posible asociar una clase para definir elementos de estilo.

El código anterior, nos generará un rectángulo de color azul como el siguiente:

Imagen 5. Rectángulo de color azul.

Elemento `<circle>`

Con el elemento `<circle>`, podemos definir una circunferencia dentro de SVG. Para ésto, `<circle>` necesita recibir como entrada las coordenadas y un radio.

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 200 200" width="200"
height="200">
    <defs>
        <style>
            .circulo{
                fill: purple;
                stroke-width: 5px;
                stroke: red;
            }
        </style>
    </defs>
    <circle class="circulo" cx="100" cy="100" r="96"/>
</svg>
```

Como podemos ver, las coordenadas están definidas por **cx** (con el valor 100), **cy** (con el valor de 100 también) y un radio **r** (con el valor de 96). Junto con definir un color de fondo púrpura a través de la clase **circulo**, también definimos un borde de color rojo con la propiedad **stroke** y el ancho del borde de 5 píxeles mediante **stroke-width**.

El código anterior, nos entregará el siguiente resultado:

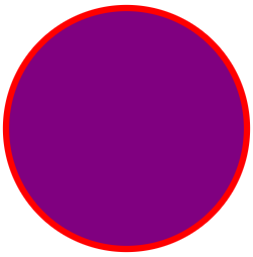

Imagen 6. Resultado de código anterior - Cículo.

Elemento `<polygon>`

Con el elemento `<polygon>`, podemos definir figuras que al menos tengan 3 lados, como por ejemplo un triángulo, un cuadrado, un hexágono, etc., mediante la definición de coordenadas.

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 210 500" width="210"
height="500">
    <defs>
        <style>
            .estrella{
                fill: lime;
                stroke: blue;
                stroke-width: 5;
            }
        </style>
    </defs>
  <polygon points="100,10 40,198 190,78 10,78 160,198" class="estrella"/>
</svg>
```

Para definir las coordenadas del polígono, nos valemos de la propiedad **points**, en donde definimos las mismas a través de pares (**x,y**). En este caso, la primera coordenada es **100,10**, la segunda es **40,198** y así con el resto.

Con el código anterior y la definición de 5 coordenadas, podemos desplegar una figura con forma de estrella, como la siguiente:

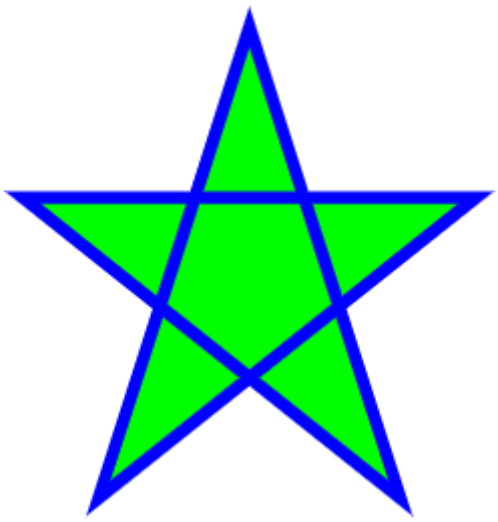

Imagen 7. Resultado de código anterior - Estrella.

Elemento `<defs>`

Ahora si miramos hacia arriba veremos a dos etiquetas, la primera aún no la conocemos y la segunda nos puede parecer muy similar y de hecho lo es, ya que aquí podremos agregar estilos CSS en nuestro SVG.

De hecho si vemos las clases dentro de esta etiqueta podremos reconocer que estas son las mismas que usaban las formas al interior de `<g>`.

Si queremos devolver los estilos hacia las formas simplemente debemos modificar la clase de los estilos por las nuevas clases que definimos anteriormente.

```
<defs>
  <style>
    .shape-latam {
      fill: #000;
    }

    .shape-chevron {
      fill: #61b94c;
    }
  </style>
</defs>
```

Reusando elementos con `<defs>`

Ahora bien, el elemento que contiene a `<style>` llamado `<defs>` permite alojar contenido que no queremos que sea mostrado directamente en el viewport SVG. De igual manera, podremos usar los elementos guardados acá referenciándolos con un selector o con el atributo `xlink:href`.

En este caso, al agregar las clases `.shape-latam` y `.shape-chevron` referenciamos estos estilos hacia las formas que se encontraban al interior del grupo `#logo-desafio`.

Probemos cambiando los colores de relleno agregados con la propiedad `fill` a un color `green` para la clase `.shape-latam` y un color `orange` para `shape-chevron`.

```
<style>
  .shape-latam {
    fill: green;
  }

  .shape-chevron {
    fill: orange;
  }
</style>
```

Si revisamos el logo veremos que este cambio de color a verde para latam y el chevron a color naranjo.

Imagen 8. Cambio de color de isotipo.

Por último, algo importante a tener en consideración cuando trabajamos con SVG es el orden que tienen estos dentro de un lienzo de SVG.

Agreguemos debajo del grupo `#logo-desafio` un elemento `<rect>` a modo de generar un cuadrado. Pongamos un ancho de `200` y un alto de `200` y un relleno color `black`.

```
<svg>
  <rect width="200" height="200" fill="black"/>
</svg>
```

Si revisamos la imagen veremos que el cuadrado se superpuso sobre el logo.

Imagen 9. Recuadro sobre isotipo.

Ahora, si cambiamos de posición de el cuadrado al principio del logo podremos identificar que el cuadrado está ahora debajo del logo.

Imagen 10. Recuadro tras isotipo.

En general esto sucede debido a que los elementos SVG tienen un orden el cual dependerá del tipo de diseño que tenga la imagen, pues como vimos si el cuadrado queda debajo del logo este se superpondrá, pero si lo dejamos arriba esta moverá hacia atrás.

En resumen, nos encontraremos a menudo con estos elementos al interior de un SVG, por lo que es importante entender el significado de estas etiquetas y atributos, así como también el orden que las formas deben tener al interior de un lienzo SVG a modo de poder manipularlos de mejor manera.

Lectura complementaria

- Document Organization (Pocket Guide to Writing SVG) - Joni Trythall

Integrando un SVG

Anteriormente conocimos algunos elementos básicos que nos serán de ayuda para poder manipular un SVG dentro de un proyecto web.

Ahora, conoceremos las formas en la cual podremos integrar una imagen SVG dentro de una interfaz de usuario.

Para hacerlo crearemos un documento HTML al interior de nuestro escritorio usando nuestra terminal con el comando `touch`.

```
cd Desktop
touch index.html
```

Luego de crear el archivo moveremos este hacia nuestro editor de texto. Cuando estemos dentro agregaremos la estructura base de un documento HTML y después, debajo de `<title>` pondremos una etiqueta `<style>`.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Agregando un SVG</title>
</head>
<body>
  <!-- Código HTML -->
</body>
</html>
```

Bien, con ello listo comenzaremos a ver las formas en que podemos integrar un SVG al interior de un proyecto web:

Formas de integrar una imagen SVG

Usando etiqueta `<img>`

La primera forma de integrar un SVG es usando una etiqueta `<img>`. Para hacerlo debemos agregar en el atributo `src` la dirección relativa del archivo SVG a integrar.

Probemos esto agregando una etiqueta `<img>` en el que agregaremos la dirección relativa la imagen.

```
<!-- Integrar una imagen SVG con etiqueta img -->
<img src="iso-desafio-latam.svg" alt="Logo Desafío Latam">
```

Al guardar y abrir la página en nuestro navegador veremos que la imagen aparece sin ningún problema.

Usando una imagen de fondo

La segunda forma que veremos será usando una imagen de fondo. En este caso lo que debemos hacer es agregar un elemento en el cual definir una imagen de fondo, por ejemplo usando un `<div>` con una clase `logo`.

```
<!-- Integrar una imagen SVG con un imagen de fondo -->
<div class="logo"></div>
```

Después deberemos agregar la imagen de fondo usando la propiedad `background-image` al interior de la clase `.logo`. En esta propiedad pondremos la dirección relativa de la imagen.

```
.logo {
  background-image: url(iso-desafio-latam.svg);
}
```

Si recargamos veremos que la imagen no se verá. Esto es debido a que el contenedor aún no tiene un ancho, por lo tanto agregaremos un ancho de `300px` para mostrar la imagen.

```
.logo {
  height: 300px;
  background-image: url(iso-desafio-latam.svg);
}
```

Ahora si recargamos veremos el logo pero repetido varias veces. Para solventar esto usaremos la propiedad `background-repeat` con un valor `no-repeat`.

```
.logo {
  height: 300px;
  background-image: url(iso-desafio-latam.svg);
  background-repeat: no-repeat;
}
```

Si recargamos una última vez podremos sin ningún problema el logo en nuestro sitio.

Asimismo, para evitar algunos problemas relacionados a la posición del SVG podemos agregar la propiedad `background-size` con un valor `contain` a modo reducir el tamaño de la imagen y asegurarnos que esta sea totalmente visible.

Otra cosa importante relacionada a la integración de imágenes SVG es que si queremos modificar los estilos de las formas incluidas en estos archivos no podremos hacerlo con ellas ya que con estas sólo podremos definir sólo el tamaño, de modo que modificar el estilo por completo deberemos integrar estas imágenes haciéndolo de manera inline o usando una etiqueta llamada `<object>`.

Usando SVG de manera inline

La primera opción que tendremos para poder cambiar los estilos de un SVG es pegando directamente el código SVG en nuestro documento HTML.

De hecho, si vamos al código del archivo SVG, copiamos su contenido y finalmente, lo pegamos al interior de nuestro archivo HTML, veremos que la imagen imagen se muestra sin ningún problema.

```
<!-- Integrar una imagen SVG de manera inline -->
<svg xmlns="http://www.w3.org/2000/svg" width="300px" height="400px"
viewBox="0 0 368 203">
  <defs>
    <style>
      .shape-latam {
        fill: green;
      }

      .shape-chevron {
        fill: orange;
      }
    </style>
  </defs>
  <!-- Forma cuadrado -->
  <rect width="200" height="200" fill="black" />
    <!-- Logo Desafío Latam -->
  <g id="logo-desafio" data-name="Logo Desafio Latam"
transform="translate(-3057.509 -3952)">
    <path id="latam-map" data-name="Latam Map" class="shape-latam" d="comandos
de forma"
          transform="translate(2832.419 3854.582)" />
    <path id="left-chevron" data-name="Left chevron" class="shape-chevron"
d="comandos de forma" />
    <path id="right-chevron" data-name="Right chevron" class="shape-chevron"
d="comandos de forma" transform="translate(2852.258 3862.28)" />
  </g>
</svg>
```

Ahora probemos que se puede cambiar los estilos de nuestra imagen moviendo las clases `.shape-latam` y `.shape-chevron` al interior de la etiqueta `<style>`.

```
<style>
  .logo {
    height: 300px;
    background-image: url(iso-desafio-latam.svg);
    background-repeat: no-repeat;
  }

  .shape-latam {
    fill: green;
  }

  .shape-chevron {
    fill: orange;
  }
</style>
```

Cambiemos el relleno de los elementos de `.shape-latam` a `gold` y de `.shape-chevron` a `salmon`.

```
.shape-latam {
  fill: gold;
}

.shape-chevron {
  fill: salmon;
}
```

Si recargamos veremos que los elementos de la imagen cambiaron de color sin ningún problema.

Imagen 11. Cambio de relleno de isotipo.

Hasta el momento esta es indiscutiblemente la mejor manera de integrar SVG, ya que no es necesario cargar de manera externa las imágenes y podremos editar vía CSS todas las formas SVG de manera fácil, pero aunque suene todo genial este método tiene un gran problema relacionado a la cantidad de código extra que tendrá el documento HTML, por lo tanto es aconsejable optimizar el código SVG o evitar sobre usar este método.

Usando SVG con `<object>`

La última forma que veremos, será la integración de imágenes SVG usando el elemento `<object>`. Esta etiqueta permite agregar diversos elementos multimedia como audio, video, SVG, PDF, entre otros a nuestro documento HTML.

Para nuestro caso la forma en que debemos agregar dos atributos al interior de la etiqueta. La primera es el origen del archivo usando `data=""`. Aquí agregaremos la ruta de la imagen y luego, agregando el atributo `type=""` deberemos agregar el tipo de elemento multimedia que es. En este caso usaremos un tipo `image/svg+xml`.

```
<!-- Integrar un SVG usando etiqueta object -->
<object data="iso-desafio-latam.svg" type="image/svg+xml"></object>
```

Es importante agregar dentro de este elemento un `fallback` a modo de evitar problemas de compatibilidad con entre navegadores.

Un `fallback` en CSS, es cualquier regla que sirva como apoyo en caso de que una propiedad experimental no tenga compatibilidad con un navegador específico.

En este caso, podemos hacer lo mismo utilizando al interior de `<object>` la misma imagen pero con otro formato.

```
<!-- Integrar un SVG usando etiqueta object -->
<object data="iso-desafio-latam.svg" type="image/svg+xml">
  <img src="isotipo-desafio-latam.png" alt="Logo desafío Latam">
</object>
```

Si revisamos la imagen se mostrará sin problemas, pero si borramos la extensión del archivo se mostrará otra imagen la cual será el `fallback`.

Algo importante a tener en cuenta sobre este método es que para cambiar los estilos de estos debemos hacerlo al interior del archivo SVG, tal como lo vimos en unidades anteriores.

En definitiva agregar un archivo SVG a un proyecto web es un gran idea debido al escazo peso de las imágenes, la posibilidad de modificar sus estilos y por supuesto por su potencial de escalar sin perder calidad.

Lectura Complementaria

- Using SVG - Chris Coyier (CSS Tricks)

Optimizando un SVG

Como vimos anteriormente, integrar un archivo dentro de un SVG es relativamente fácil si conocemos las maneras que existen para agregarlos y si entendemos las características de las cuales podremos aprovecharnos.

Estas características, como darle estilos a los SVG con CSS, son una manera muy útil de transformar imágenes usando como base un código, lo cual es muy cómodo si trabajamos en un proyecto web.

Ahora bien, con respecto a este último punto, podemos pensar que todas las imágenes SVG tienen el mismo orden y organización que vimos en el código del logo de Desafío Latam, pero no es así.

El orden y organización de un SVG

De hecho, la orden y estructura que vemos en al interior de un SVG dependerá de dos factores cruciales relacionados al programa de edición gráfico y de la organización del creador de la imagen SVG. Estos dos elementos combinados provocarán que al revisar un código SVG sea fácil de entender o que simplemente no se pueda trabajar con el.

Para ver esto de mejor manera descargaremos un icono SVG desde la página **Flaticon**, que es un repositorio gigante de iconos gratuitos. Ingresemos a este escribiendo https://www.flaticon.com en nuestro navegador.

Cuando estemos dentro busquemos un icono, como por ejemplo idea. Elijamos la primera opción. al presionar nos aparecerá un modal con diferentes opciones. Presionemos el botón `SVG`, y luego en el siguiente modal presionaremos la opción `Free Download`. Guardamos la imagen y esperamos a que descargue. Luego de descargar moveremos esta a nuestro editor de texto favorito.

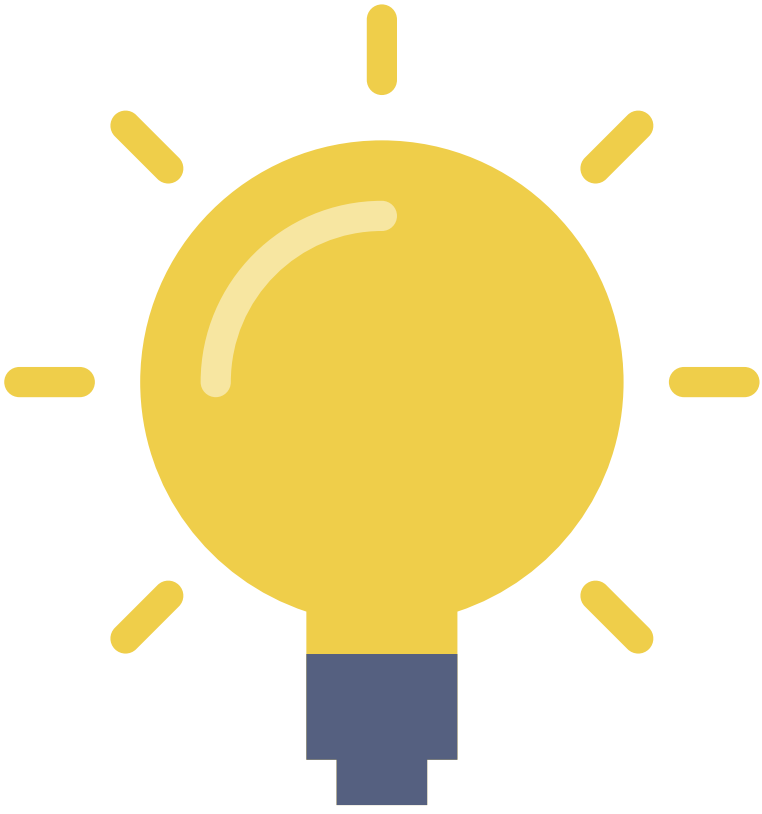

Imagen 12. Ícono IDEA que se utilizará de ejemplo.

Al entrar podremos ver el código fuente de la imagen, junto además de la versión SVG, el generador usado por la imagen, que en este caso es de Ilustrator y debajo las formas que componen a la imagen, junto también con una gran cantidad de grupos vacíos.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Generator: Adobe Illustrator 19.0.0, SVG Export Plug-In . SVG Version:
6.00 Build 0)  -->
<svg version="1.1" id="Capa_1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
     viewBox="0 0 53 53" style="enable-background:new 0 0 53 53;"
xml:space="preserve">
    <path style="fill:#EFCE4A;" d="M26.5,9c-8.837,0-16,7.164-
16,16c0,7.089,4.615,13.091,11,15.192V50h2v3h6v-3h2v-9.808
    c6.385-2.101,11-8.103,11-15.192C42.5,16.164,35.337,9,26.5,9z"/>
  <g>
    <!-- Formas SVG-->
  <g/>
  <!-- Grupos sin uso -->
  <g></g><g></g><g></g><g></g><g></g><g></g><g></g><g></g><g></g><g></g><g></g><g></g><g>
</g><g></g><g></g><g></g>
</svg>
```

Estos grupos sin formas representan elementos que ensuciarán nuestro código al usarlos de manera inline en un documento HTML, por lo que es una buena idea eliminarlos.

Para organizar y optimizar nuestras imágenes SVG tendremos dos opciones para realizarlos: Una es hacerlo de maneral manual y la otra es utilizando una herramienta especializada.

En nuestro caso y a modo de hacer más fácil nuestro trabajo utilizaremos una herramienta que nos ayudará en la optimización de nuestros SVG llamada "*SVG Optimiser*".

Optimizar código SVG

Esta herramienta creada por *Peter Collingridge* nos ayudará a optimizar nuestro código SVG dándonos la facultad de decidir qué elementos queremos dejar y como queremos que sea la organización del código. Estos dos puntos hacen de esta una gran herramienta para reducir y organizar una imagen SVG.

Dicho esto, ingresemos a la página principal de SVG Optimiser ingresando a: http://petercollingridge.appspot.com/svg-optimiser

Cuando ingresemos veremos un `input` de texto donde debemos pegar el código a optimizar. Copiemos el código de nuestra imagen SVG y peguemos esta en el `input`.

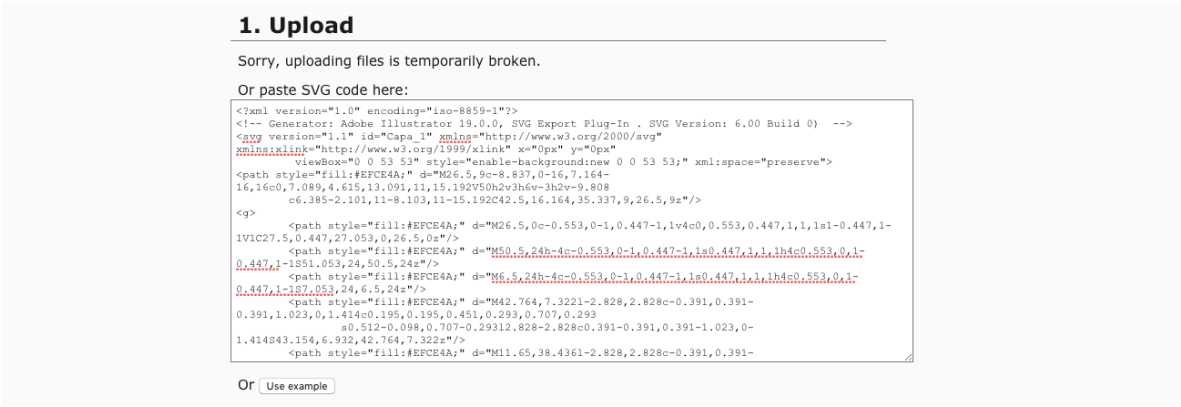

Imagen 13. Optimización de código.

Luego, debemos bajar hasta encontrar la opción `Optimise`. Desde aquí podremos definir los parámetros que quitaremos u optimizaremos del código.

Las opciones que tendremos serán:

- **Whitespace:** En dónde podremos definir si queremos comprimir o reducir el código SVG borrando todos los espacios en blanco con la opción `remove` u organizar el código con la opción `pretty`. En nuestro caso, como deseamos organizar nuestro código usaremos la opción `pretty`.
- **Style type:** Con esta opción podremos definir si nuestro estilo será una mezcla entre estilos inline con los atributos `fill` y `stroke` y estilos con clases al interior del elemento `<style>` usando `optimal`, o sólo los estilos al interior de `<style>` con `CSS`, o solamente los elementos inline con `styleString`.

Nosotros elegiremos la opción `CSS` debido a que podremos usar las clases creadas para agregar nuestros propios estilos.

- **Truncate atributte, SVG size & style:** Estos tres elementos nos permitirán reducir los decimales asociados a los atributos de la imagen como `d`, al tamaño del SVG y de limitar la cantidad de estilos en formas importantes dentro de una imagen SVG.

Mantengamos estos valores por defecto, puesto que no requerimos que nuestro código pese menos sino que sea más fácil trabajar con el, de modo que mantendremos esta configuración.

- **Remove:** Las opciones `remove` nos permitirán seleccionar qué elementos queremos remover desde nuestro código SVG.

El remover elementos dependerá de la estructura y la dependencia que tiene la forma con un atributo o id específico.

Si miramos nuestro código veremos que no existe muchas dependencias a otros atributos en esta imagen SVG, de modo que podremos remover id's, estilos innecesarios, elementos vacíos, formas redundantes, y aplicar transformaciones si es que las tuviera.

Por otra parte, deseleccionaremos remover atributos y estilos por defecto, y remover grupos.

Si bajamos y revisamos la sección `analisys` veremos que el peso de nuestro SVG se redujo significativamente, sin afectar la organización de este.

Bajemos un poco más, ahí veremos el resultado final de nuestra imagen optimizada. Si deseamos usarla podemos descargar la imagen o simplemente copiar el código SVG para luego usarlo en otro lugar.

Realicemos la segunda opción copiando el código y luego, pegándolo en la imagen SVG sin optimizar.

```
<svg xmlns="http://www.w3.org/2000/svg" x="0" y="0" viewBox="0 0 53 53"
class="undefined">
<style>
  .a{
    fill:#EFCE4A;
  }
  .b{
    fill:#F7E6A1;
  }
  .c{
    fill:#556080;
  }
</style>
  <!-- Formas SVG -->
</svg>
```

Si revisamos la imagen podremos ver que esta no cambió en ningún aspecto visual, lo que es genial.

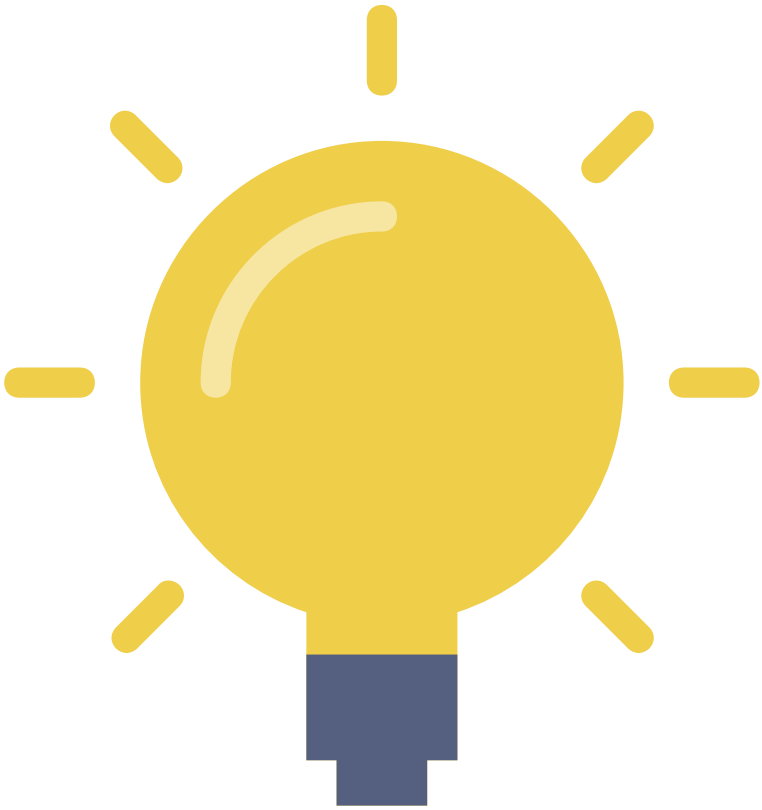

Imagen 14. Visualización de ícono IDEA al incorporar cambios.

Sin embargo, es importante a tener en cuenta sobre esta herramienta es que puede ocurrir que al optimizar una imagen SVG con muchas formas este pueda desformar la imagen final, de modo que es importante usar esta herramienta a conciencia.

En definitiva, el optimizar una imagen SVG nos ayudará a disminuir el tamaño de nuestro SVG y además nos permitirá organizar y reducir las líneas de código usadas en un código SVG, lo que nos permitirá trabajar de mejor manera cuándo insertamos SVG de manera online.