

Kara Effector 3.2:

Effector Book

Vol. II [Tomo XXIV]



Kara Effector 3.2:

El **Tomo XXIV** es otro más dedicado a la librería **shape**, que como ya habrán notado, es la más extensa hasta ahora vista en el **Kara Effector**. El tamaño de esta librería nos da una idea de la importancia de las Shapes en un efecto karaoke, y es por ello que debemos tomarnos un tiempo en ver y conocer a cada una de las funciones y recursos disponibles para poder dominarlas.

Librería Shape [KE]:

» **shape.filter2(Shape, Filter, Split)**

Esta función es la combinación de las funciones **shape.redraw** y **shape.modify** y tiene una modificación en la estructura de la función modificadora **Filter**.

El parámetro **Filter** es la función que modifica los puntos de la **shape** ingresada.

El parámetro **Split** es la longitud de los segmentos en los que se dividirá la **shape** ingresada.

La estructura de la función **Filter** es:

```
function( x, y )
```

```
-- Instrucciones --
```

```
return x, y
```

```
end
```

Noten que la estructura de la función es más simple que la de la función **shape.modify**, ya que la parte que retorna es simplemente: **return x, y**

> Ejemplo:

```

1 mi_modify = function( x, y )
2     y = y + R(4,8)*(-1)^R(2)
3     return x, y
4 end

```

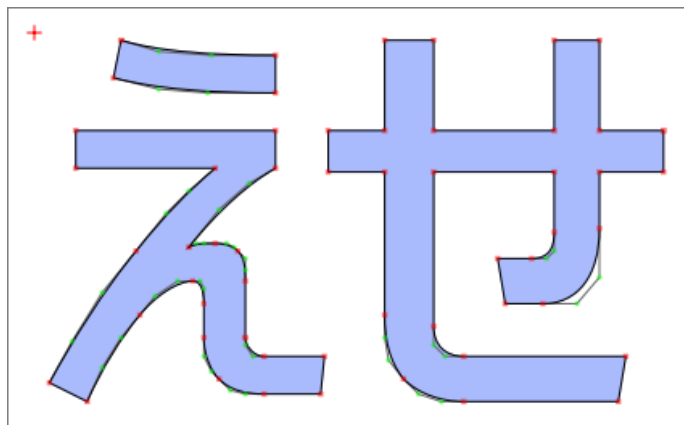
>> Variables:

```

mi_modify = function( x, y )
    y = y + R(4,8)*(-1)^R(2)
    return x, y
end

```

Y usaremos esta shape para modificarla. Ya sabemos el procedimiento para definirla como una variable en la celda de texto “Variables”:



Y en Return [fx] ponemos:

>> Return [fx]:

```
shape.filter2( mi_shape, mi_modify, 2 )
```

Y veremos cómo se ha modificado la **shape**:



Las coordenadas respecto al eje “y” se han modificado de manera considerable y hacen que la **shape** tenga un efecto distorsionado verticalmente.

Más adelante veremos cómo esta función nos ayudará a modificar el texto de nuestros karaokes de la misma manera que lo hace con la Shapes.

```
shape.from_audio( Audio, Width, Height_scale,
                  Thickness, Offset_time )
```

Esta función convierte un archivo de audio en formato **.wav** en una animación a base de Shapes.

Hay una diversidad de programas para convertir un archivo de vídeo a **.wav**, lo mismo que archivos de audio. En lo personal uso **Format Factory**:



Y una vez tengamos nuestro archivo de audio en formato **.wav**, lo guardamos en la misma carpeta en donde esté nuestro archivo **Effector-utils-lib-3.2.lua**

El parámetro **Audio** es el nombre entre comillas, simples o dobles, de nuestro archivo **.wav**

Width es el ancho de la **shape** que simulará la frecuencia del audio. Su valor por default es **line.width**

Height_scale es la escala vertical de la **shape**. Su valor por default es 1/220







Thickness es el espesor de la **shape** y su valor por default es de 6 pixeles.

Offset_time es el tiempo adicional a la duración de la **shape** en pantalla, tanto al tiempo de inicio y final.

Esta función está pensada para ser usada en modo **Line**, es por ello que la duración en pantalla de las Shapes es la duración de la línea karaoke: **line.dur**

Ejemplo:

Como les mencionaba anteriormente, el archivo de audio de nuestro karaoke debe estar en la misma carpeta en la que se encuentra la librería principal del **Kara Effector**:

	Yutils	126 KB	Archivo LUA
	Effector-utils-lib-3.2	387 KB	Archivo LUA
	Effector-newfx-3.2	367 KB	Archivo LUA
	Effector-3.2	371 KB	Archivo LUA
	SAOIIOP	16.869 KB	Archivo de sonido
	Effector-3.2-test	6 KB	Archivo ASS

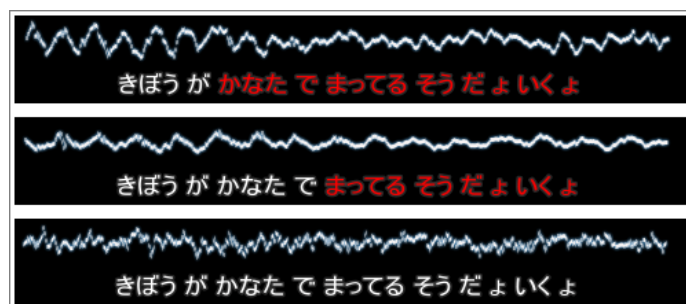
Ahora cambiamos el modo de nuestro fx a **Line**:

Template Type [fx]: **Line**

Return [fx]:

```
shape.from_audio( "SAOIIOP", l.width + 50, 1/160 )
```

Y ya en pantalla podemos ver cómo las Shapes hacen la animación de la frecuencia del audio de nuestro karaoke:



La implementación de esta función demanda cierta cantidad de recursos de nuestra PC, lo que hará que tarde cierto tiempo en aplicarse por completo. Esta función genera aproximadamente 120 líneas fx por cada línea karaoke y dependiendo de cada computadora, puede tardar desde 10 segundos hasta 2 minutos en aplicar completamente el efecto, así que no se preocupen si se tarda un poco en ello, ya que pueden ver el progreso de la aplicación al mismo tiempo que esperan que termine.

shape.bord(Shape, Bord, Size)

Esta función crea el borde de una shape ingresada con el tamaño del borde especificado y con el tamaño de la shape también seleccionado por el usuario.

El parámetro **Bord** es la medida en pixeles del tamaño del borde y su valor por default es 4px.

El parámetro **Size** es la medida en pixeles del tamaño de la **shape** que retornará la función y su valor por default son las dimensiones de la **shape** ingresada.

El parámetro **Size** puede ser o un número o una **tabla** con dos valores numéricos. Ejemplo:

Size = 120

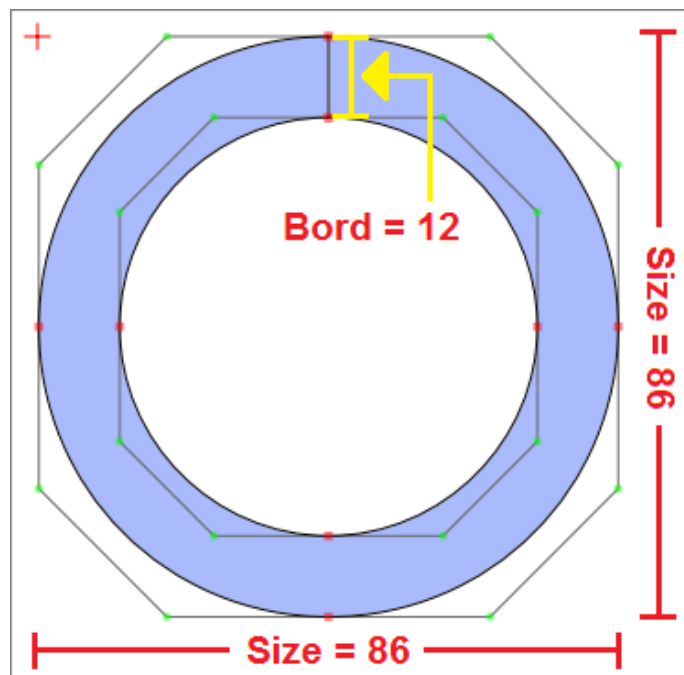
Size = {80, 100}

Para el caso cuando es un número, ambas dimensiones de la **shape** retornada, serán el mismo. Para el caso de ser una **tabla**, en ella podemos especificar el ancho y el alto a nuestro acomodo.

Ejemplo:**Return [fx]:**

```
shape.bord( shape.circle, 12, 86 )
```

Lo que nos dará la **shape** del borde del círculo, de 86 px, tanto de ancho como de alto y con un borde de 12 px. Lo que en geometría se conoce como corona circular:

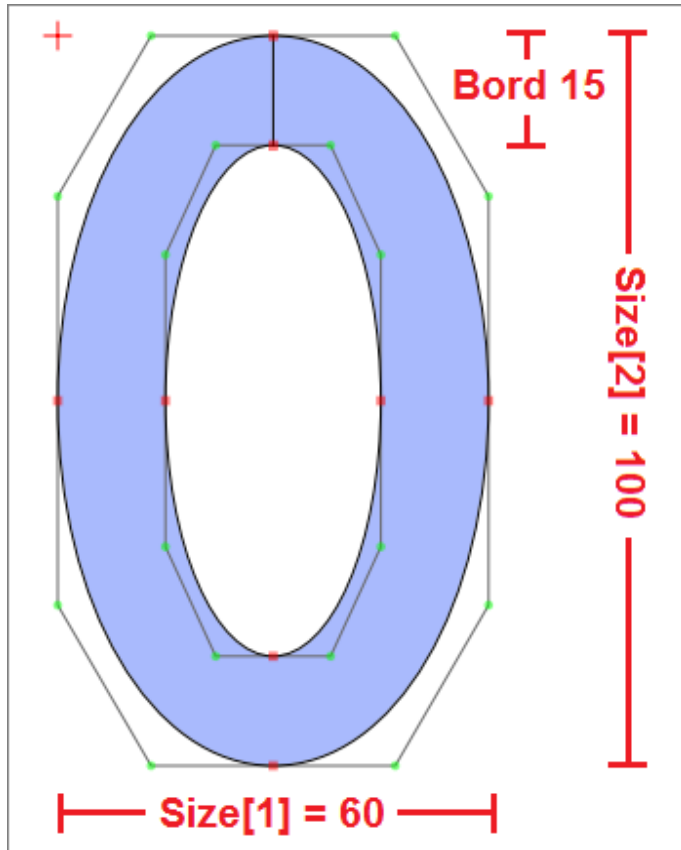


> Ejemplo:

>> Return [fx]:

```
shape.bord( shape.circle, 15, {60, 100} )
```

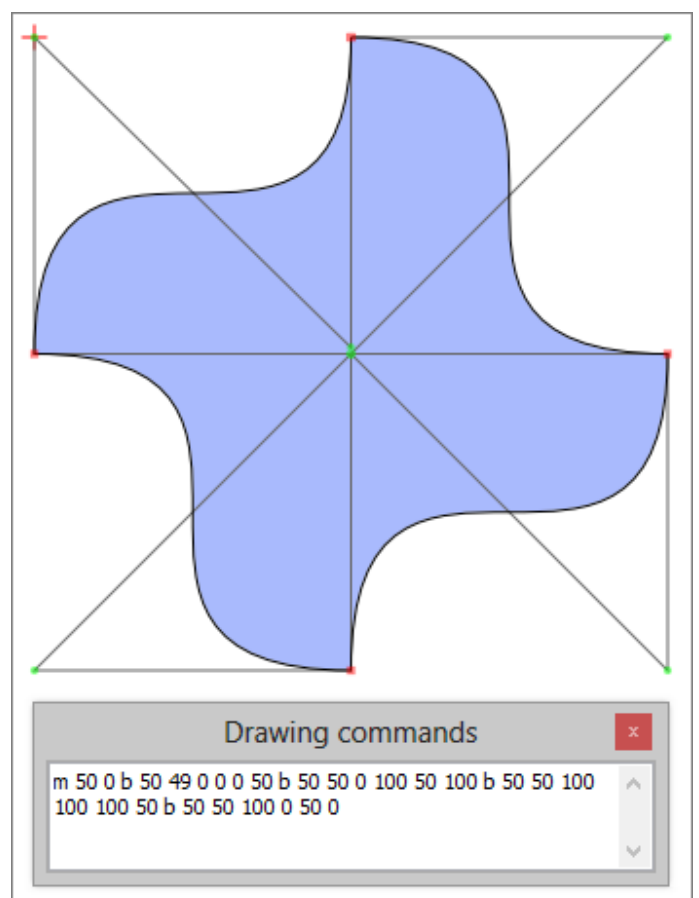
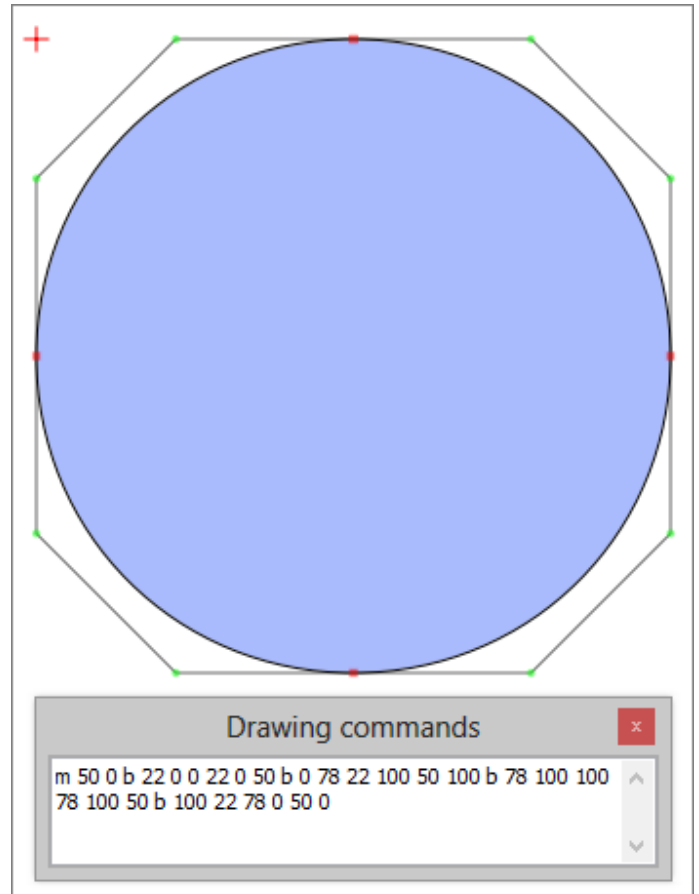
En este ejemplo, **Size** es una tabla, en donde el 60 indica el ancho de la **shape** y 100 será la altura medida también en pixeles:



>> shape.morphism(Size, Shape1, Shape2)

Esta función retorna una tabla con la interpolación de las Shapes ingresadas. Los dos parámetros **Shape1** y **Shape2** deben cumplir con una muy especial particularidad para que la función cumpla con su objetivo, y es que una de esas Shapes debe ser una modificación en la posición de los puntos de la otra, es decir que no funcionará con dos Shapes que tengan distinta cantidad de puntos. En nuestro ejemplo usaremos las Shapes de nuestra derecha →

El parámetro **Size** debe ser un número entero mayor a 2 que indicará el tamaño de la tabla que se retornará y por ende indica la cantidad de Shapes que se crearán en la interpolación de una shape a la otra.



Ejemplo:

Definimos las dos Shapes anteriores en **Variables**:

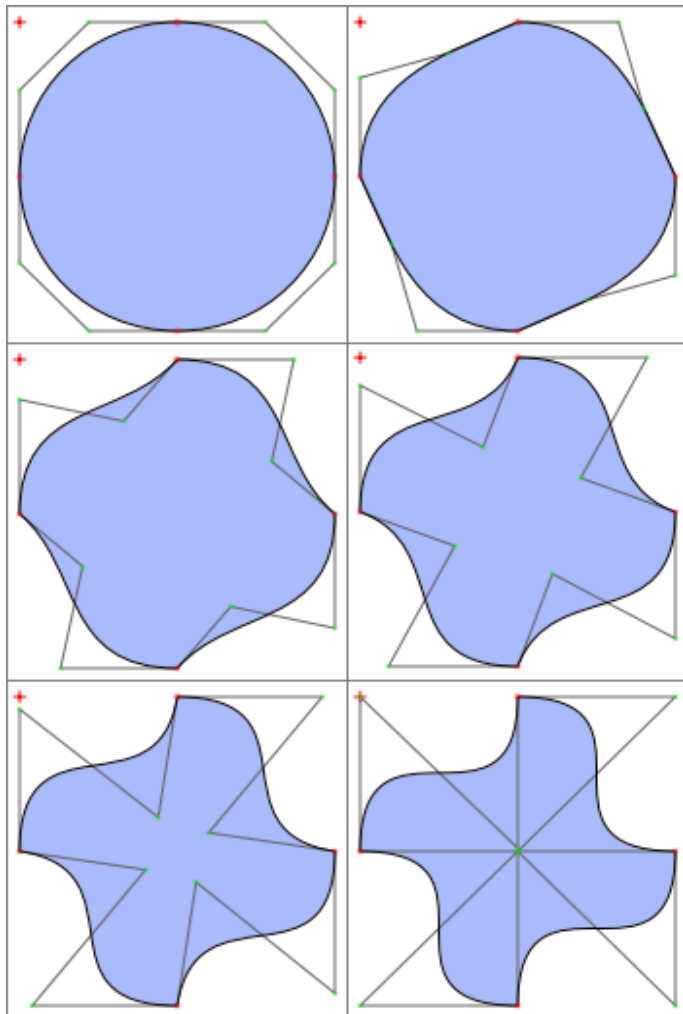
```
Variables:
mi_shape1 = "m 50 0 b 22 0 0 22 0 50 b 0 78 22 100 50 100 b 78 100 100 78 100 50 b 100 22 78 0 50 0";
mi_shape2 = "m 50 0 b 50 49 0 0 0 50 b 50 50 0 100 50 100 b 50 50 100 100 50 b 50 50 100 0 50 0";
```

Como la función retorna una **tabla** de Shapes, no podemos hacer un ejemplo directo, ya que no la podríamos visualizar, entonces nos apoyamos de la función **table.show** para ver el contenido de la **tabla**:

Return [fx]:

```
table.show( shape.morphism( 6, mi_shape1,
                             mi_shape2 ) )
```

Entonces la función hará en 6 Shapes, la transición de una a la otra:



Las Shapes contenidas en este tipo de tabla es ideal para ser usada en la función **shape.animated2**, para que la transición de una **shape** a la otra se vea una animación, una transformación de la una a la otra.

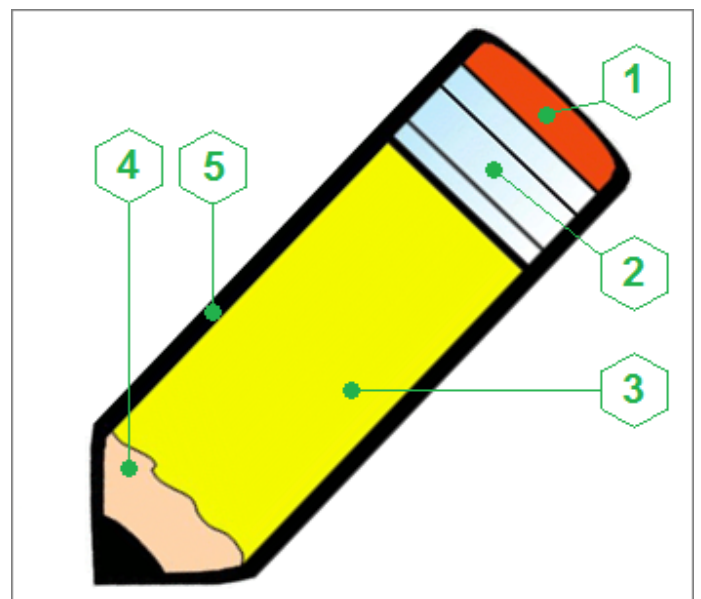
shape.divide(Shape, Mark)

Esta función divide las partes en las que se componen una shape más compleja, para que sea más simple el darle los diferentes colores de la misma y que queden ubicadas en su posición de manera automática.

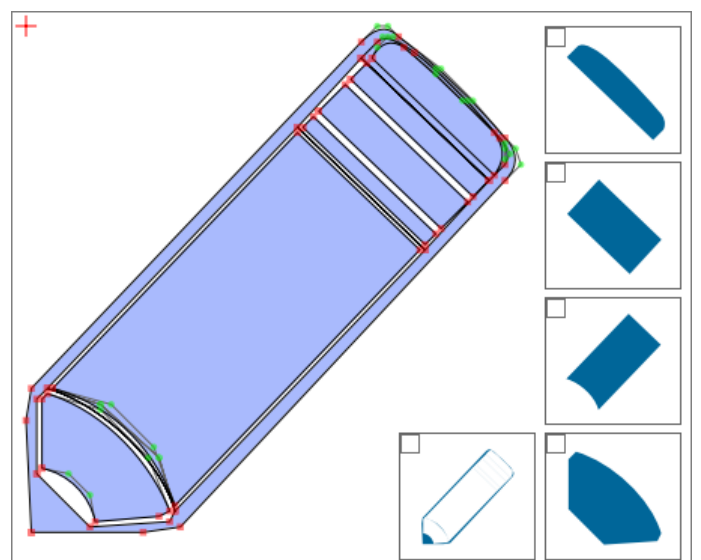
El parámetro **Mark** es opcional y hace referencia a 2 líneas paralelas que “enmarcan” a cada una de las Shapes que componen a la shape ingresada.

Ejemplo:

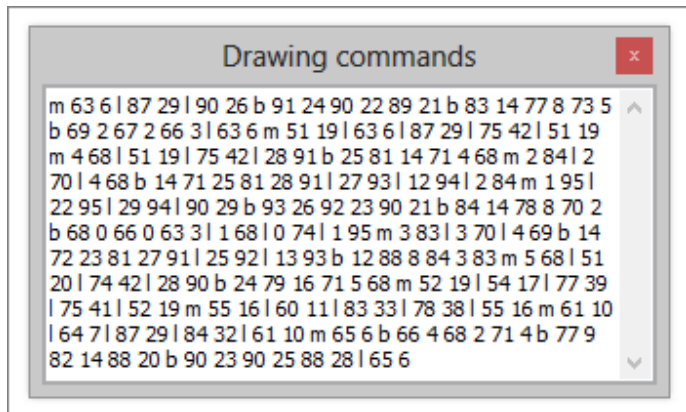
Para el siguiente ejemplo usaré la imagen de este lápiz que está compuesto por cinco Shapes individuales y tiene cinco colores diferentes:



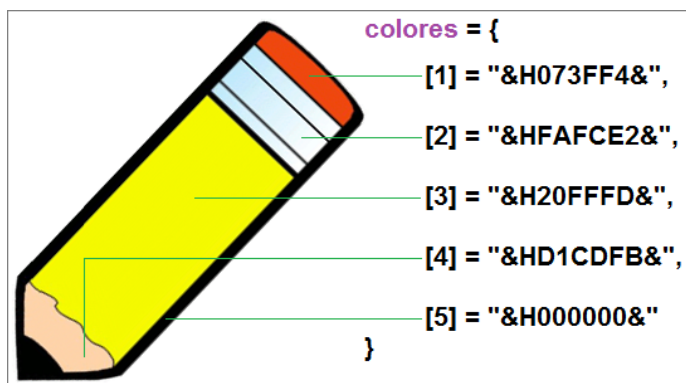
Dibujamos las cinco Shapes individuales en el mismo plano en el **ASSDraw3**. A la derecha de la imagen vemos lo que serían al dibujarlas en planos diferentes:



Este sería el código completo de toda la **shape** del lápiz al ser dibujado por partes, y con este código declaramos una variable: **mi_shape**



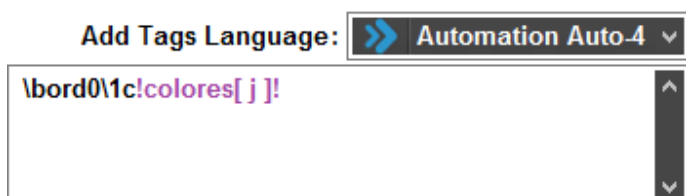
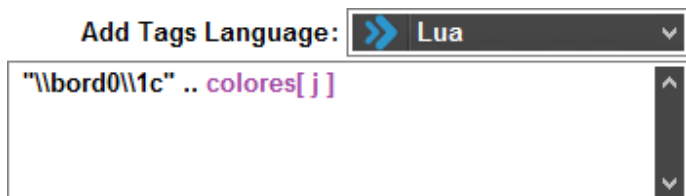
Junto con la variable de la **shape**, declaramos otra variable, una tabla que contenga los colores de cada una de las Shapes individuales, de tal manera que hagamos coincidir los colores con el orden en que dibujamos las Shapes:



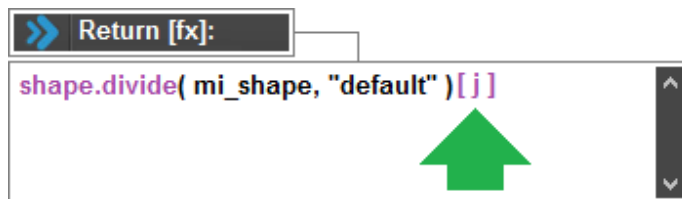
El siguiente paso es modificar el **loop** de nuestro efecto. El valor que debemos colocar será el de la cantidad de Shapes individuales que conforman a nuestra **shape**, en este caso será 5:



Hecho esto, asignamos los colores a las Shapes, y para ello ponemos lo siguiente en **Add Tags**. Recordemos las dos formas de hacerlo según el lenguaje:



Y por último, en **Return [fx]** llamamos a nuestra función, colocando la palabra **"default"** en el parámetro **Mark**:



Al aplicar, si seguimos los pasos correctamente, veremos cómo las 5 Shapes conforman al lápiz, con sus respectivos colores asignados:



Entonces, el parámetro **Mark** tiene las siguientes opciones:

- Si lo omitimos, la función no agregará nada a las Shapes individuales de la shape.
- Si ponemos la palabra **"default"**, la función le pondrá dos líneas paralelas a cada una de las Shapes, con las dimensiones por default de la **shape** ingresada.
- Puede ser una tabla con cuatro valores numéricos, de manera que los dos primeros corresponden a la coordenada superior izquierda del marco y los dos siguientes, a la coordenada inferior derecha del mismo:



Es todo por ahora para el **Tomo XXIV**. Intenten poner en práctica todos los ejemplos vistos y no olviden descargar la última actualización disponible del **Kara Effector 3.2** y visitarnos en el **Blog Oficial**, lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

- www.karaeffector.blogspot.com
- www.facebook.com/karaeffector
- www.youtube.com/user/victor8607
- www.youtube.com/user/Natsuoke
- www.youtube.com/user/karalaura2012