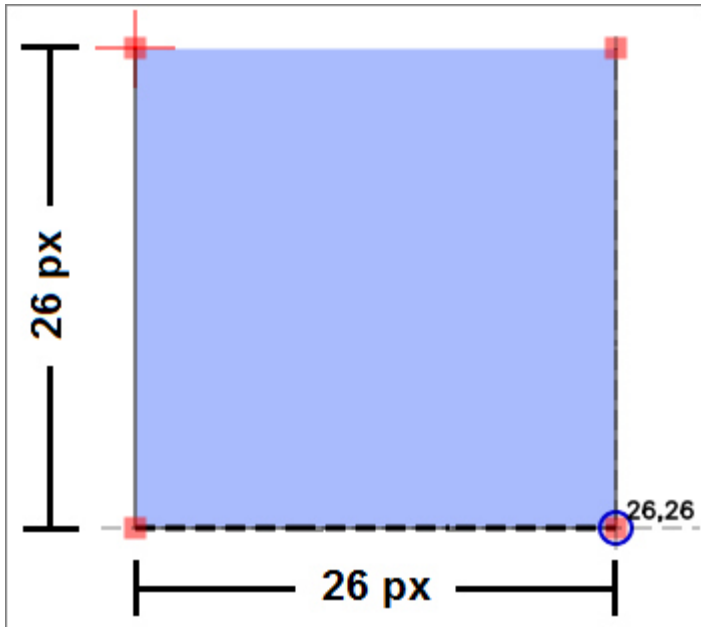


Librería Shape [KE] – Parte 3

shape.length(Shape) : Esta función retorna la medida en pixeles de la longitud total del perímetro de la **shape** ingresada.

Ejemplo:

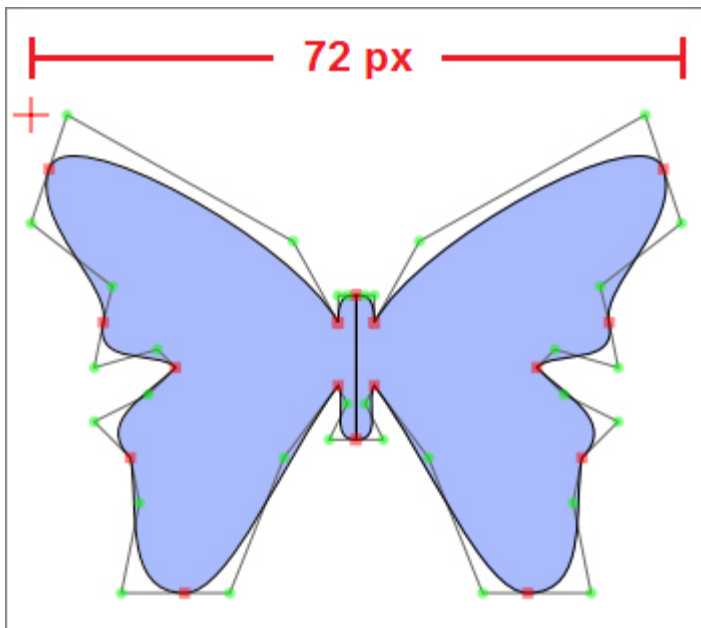
Shape.length(“m 0 0 1 0 26 1 26 26 1 26 0 1 0 0 “)



Entonces la función calculará la medida del perímetro de la **shape**, o sea: $26 + 26 + 26 + 26 = 104$ px.

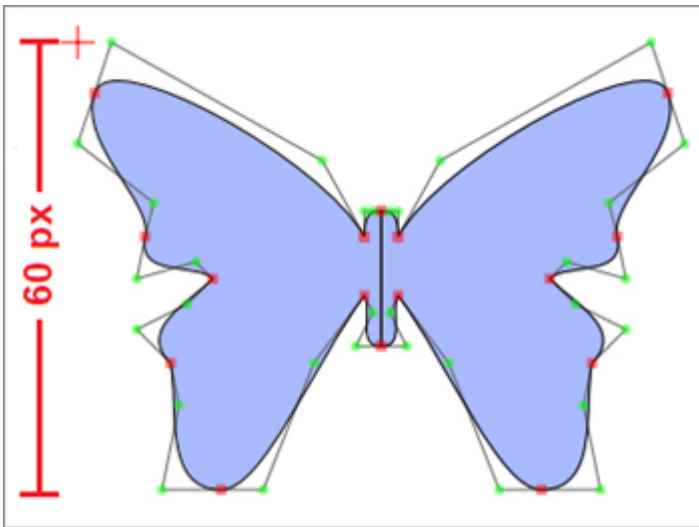
shape.width(Shape): Esta función retorna la medida en pixeles del ancho total de la **shape** ingresada.

Ejemplo:



shape.height(Shape) : Esta función retorna la medida en pixeles de la altura total de la **shape** ingresada.

Ejemplo:

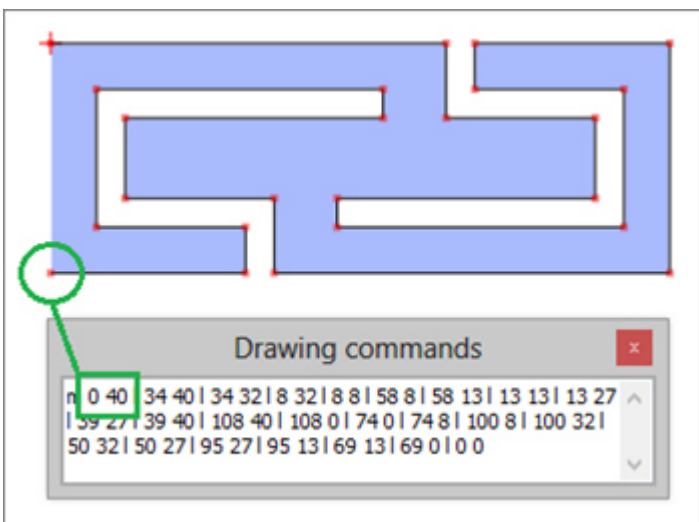


Las tres anteriores funciones están diseñadas para arrojar información básica de una **shape**, información como su longitud, su ancho y su altura. Esta información podrá ser usada en los efectos sin la necesidad de hacer los cálculos.

shape.firstpos(Shape, Px, Py) : Esta función es muy similar a **shape.displace** con la particularidad que mueve a todos los puntos de la **shape** ingresada teniendo en cuenta al primer punto de la misma, como referencia para hacerlo.

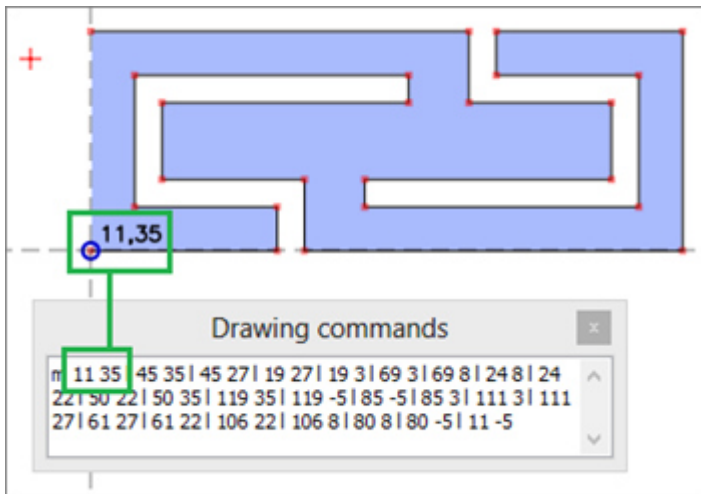
Ejemplo:

En la siguiente imagen vemos el primer punto de una shape, y con referencia a ese punto es que la función la moverá a la nueva posición que le indiquemos con los parámetros **Px** y **Py**:



Px es la coordenada respecto al eje “x” que tendrá el primer punto de la **shape**, y **Py** es la coordenada respecto al eje “y” de dicho punto:

shape.firstpos(mi_shape, 11, 35)

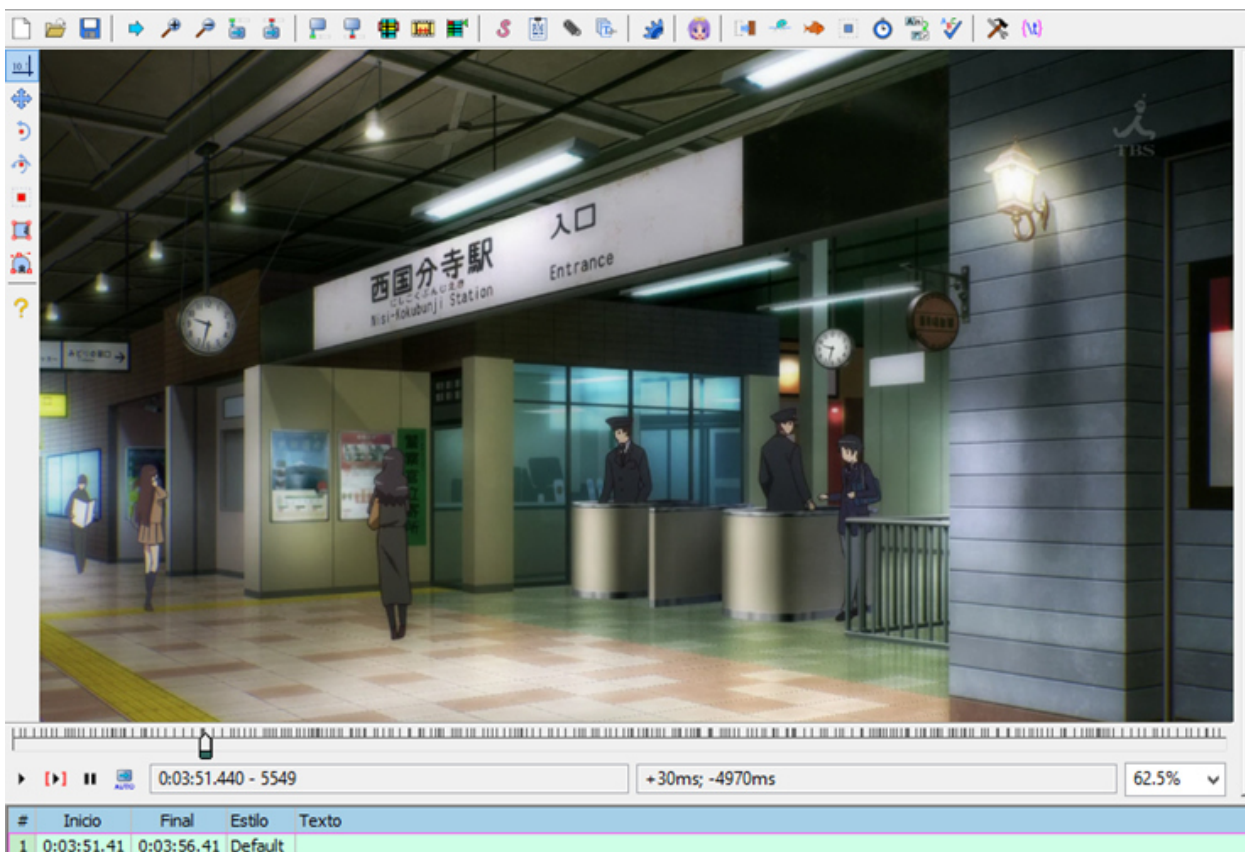


En conclusión, lo que hacemos al darles valores a los parámetros **Px** y **Py**, es asignar el punto exacto en donde quedará el primer punto de nuestra **shape** luego de ser desplazada. Esta función es ideal para reubicar la shape respecto al centro de la **Syl** y luego aplicar una de las funciones de **shape** de movimiento, como **shape.Smove**.

shape.from_clip() : Esta función está pensada para ser usada en la edición, más que para hacer fx karaokes. Esta función ayuda a crear carteles y lo que hace es convertir un clip dibujado en la pantalla en un shape.

Ejemplo:

Ubicamos el frame en dónde está el cartel que queremos hacer y le damos los tiempos de inicio y final a una línea en la que posteriormente dibujaremos el clip:



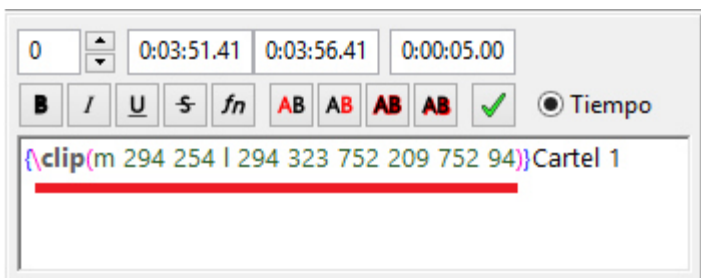
En esa línea creada, escribimos algo, lo que sea, por ejemplo: “cartel 1” y seleccionamos la herramienta para recortar subtítulos en un área vectorial:



Y dibujamos el contorno del cartel:



Hecho esto, en nuestra línea que previamente habíamos creado, se debe ver el código del clip, algo como esto:



Y por último, en **Return [fx]** llamamos a la función:



Y al aplicar ya podemos ver la **shape** justo en el lugar en donde está el cartel:



#	Inicio	Final	Estilo	Texto
1	0:03:51.41	0:03:56.41	Default	*Cartel 1
2	0:03:51.41	0:03:56.41	Default	*m 294 254 294 323 752 209 752 94

Recordemos que los colores y transparencias de una shape los podemos modificar con las herramientas de la **Ventana de Modificación del KE** destinadas para ello:

Shape Primary Color	Shape Border Color	Shape Shadow Color
<input type="color" value="#FFFFFF"/>	<input type="color" value="#4682B4"/>	<input type="color" value="#00008B"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

shape.redraw(Shape, tract) : Esta función cumple con la tarea de redibujar la shape que ingresemos, pero dividiendo a las partes que la componen en tramos muchos más cortos.

El parámetro **tract** es la medida en pixeles en la que será dividido cada segmento de la **shape** y debe ser un número mayor a cero. Su valor por default es 2.

Ejemplo:

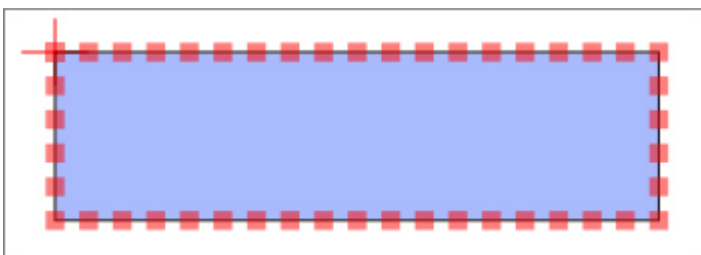


Entonces ponemos el código de la anterior **shape** dentro de la función y aplicamos:

➤ Return [fx]:

```
shape.redraw( "m 0 0 | 0 10 | 38 10 | 38 0 | 0 0 " )
```

Y obtendremos esto:

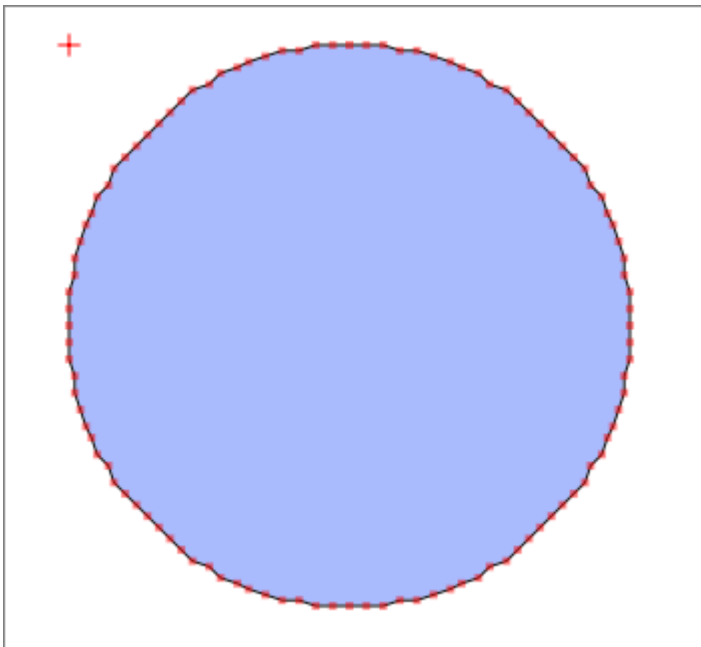


La **shape** que genera la función es la misma, pero dibujada con segmentos cortos de 2 pixeles de largo, ya que no pusimos el parámetro **tract**, entonces la función lo asume por default, es decir: 2 px.

Ejemplo:

➤ Return [fx]:

```
shape.redraw( shape.circle, 3 )
```



Entonces la función redibuja el círculo con tramos rectos de 3 píxeles de longitud.

Puesto en práctica estos dos ejemplos, la pregunta sería: ¿por qué redibujar una **shape** en segmentos rectos más cortos?

Una de las varias respuestas a esa pregunta la puede responder la siguiente función de la **librería shape**:

shape.modify(Shape, modify): Esta función modifica a los puntos de una shape por medio de una función ingresada en el parámetro **modify**, que debe tener la siguiente estructura:

modify:

```
function( x, y )
  -- Instrucciones --
  return format( "%d %d", x, y )
end
```

Ejemplo:

Aplicamos el **shape.redraw** al círculo y la definimos como una variable en la celda de texto “**Variables**”:

```
>> Variables:
mi_shape = shape.redraw( shape.circle, 1 )
```

Seguido de esto, definimos la función que modificará a la anterior **shape**:

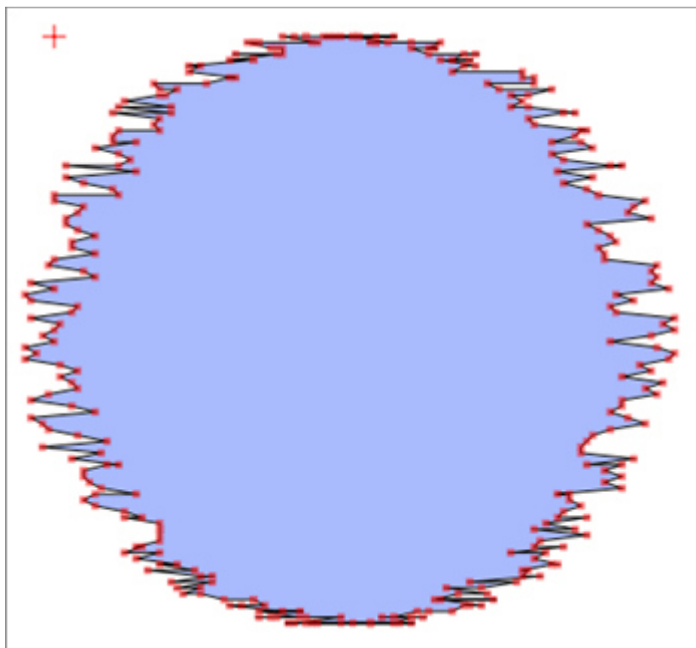
```
>> Variables:
mi_shape = shape.redraw( shape.circle, 1 );
mi_modify = function( x, y )
    x = x + R(-6,6)
    return format( "%d %d", x, y )
end
```

La instrucción de la anterior función declarada es que tome las coordenadas respecto al eje “x” de la **shape** y las modifique en una cantidad aleatoria entre -6 y 6 px.

Ahora llamamos a la función **shape.modify** con las dos variables que acabamos que declarar:

```
>> Return [fx]:
shape.modify( mi_shape, mi_modify )
```

Lo que dará por resultado algo como esto:



Y lo que antes era un simple círculo, ahora es uno que está distorsionado, como la estática de una TV sin señal.

Otra característica de la función **shape.modify** es que contiene internamente a la función **shape.info**, que como recordaremos, nos da la siguiente información de la **shape**:

- **minx** ← mínima coordenada en “x”
- **maxx** ← máxima coordenada en “x”
- **miny** ← mínima coordenada en “y”
- **maxy** ← máxima coordenada en “y”
- **w_shape** ← ancho de la **shape**
- **h_shape** ← alto de la **shape**

Y esta información la podemos usar dentro de la función que usaremos para modificar a la **shape**. Ejemplo:

```
1 mi_modify = function( x, y )
2     mod = (y - miny)/h_shape
3     x = x + 20*sin( 2*pi*mod )
4     y = 2*y
5     return format( "%d %d", x, y )
6 end
```

Generalmente uso el programa **Notepad++** para hacer las funciones, pero no es obligatorio hacerlo, ya que se pueden hacer directamente en el **Kara Effector** o hasta usar el Bloc de Notas es suficiente para ello.

La anterior función de modificación, declarada en la celda de texto “**Variable**” se vería así:

» Variables:

```
mi_shape = shape.redraw( shape.circle, 1 );
mi_modify = function( x, y )
    mod = (y - miny)/h_shape
    x = x + 20*sin( 2*pi*mod )
    y = 2*y
    return format( "%d %d", x, y )
end
```

Lo que quieren decir las instrucciones de la anterior función es que a cada coordenada “x” de la **shape** ingresada, se le sume esta función trigonométrica:

$$20*\text{math.sin}(s) \rightarrow 0 \leq s \leq 2*\pi$$

Y que a cada coordenada en “y” se multiplique por 2:

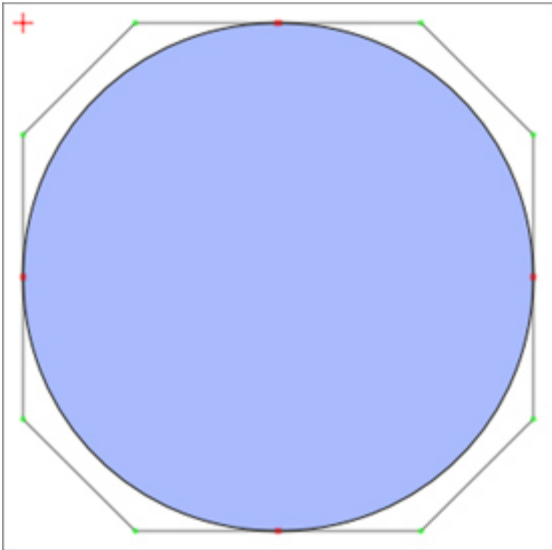
$$y = 2*y$$

Y ya sabemos lo que debemos poner en **Return [fx]**:

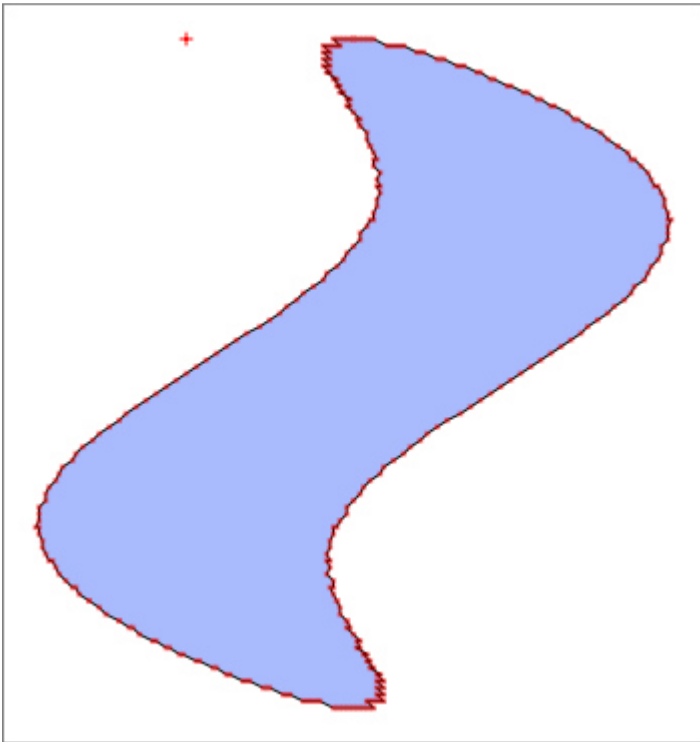
» Return [fx]:

```
shape.modify( mi_shape, mi_modify )
```


Y convertiremos esta **shape**:



En esto:



Imaginen hacerle estas deformaciones al texto, sería algo genial para hacer nuestros fx. Pues es posible hacerlo y más adelante les mostraremos la forma de lograrlo, pero primero es importante que se vayan familiarizando con las funciones necesarias para ello.

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**

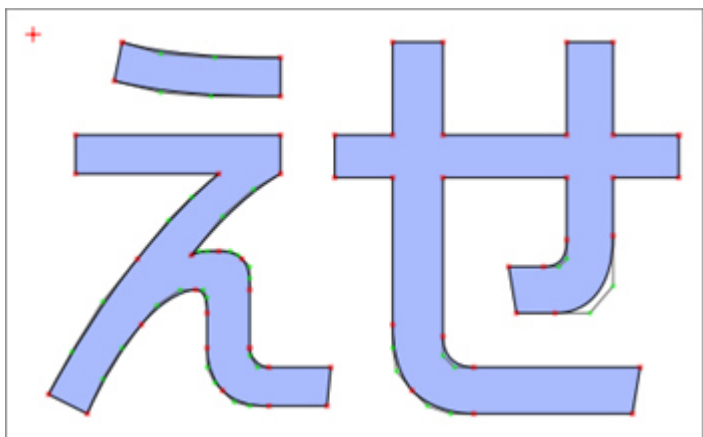
Ejemplo:

```
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

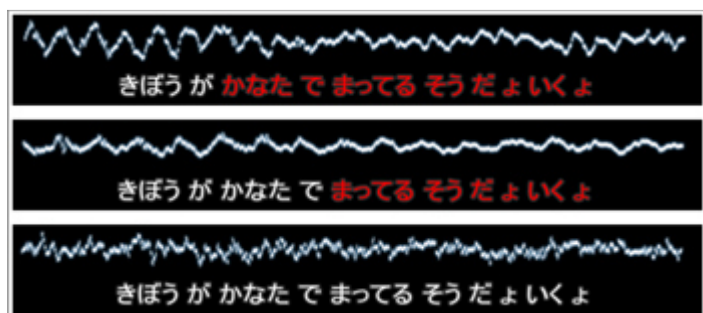
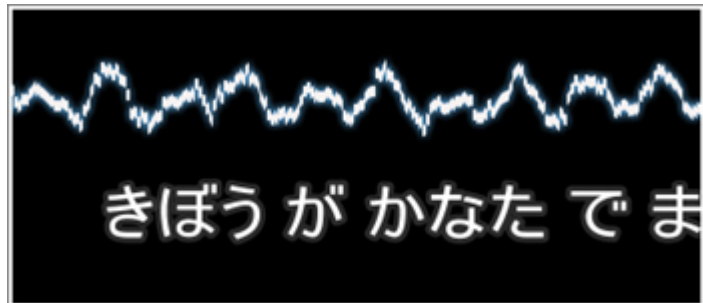
Ahora cambiamos el modo de nuestro fx a **Line**:

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.

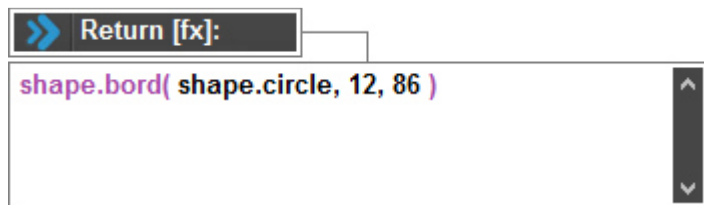
El parámetro **Size** puede ser o un número o una **tabla** con dos valores numéricos. Ejemplo:

Size = 120

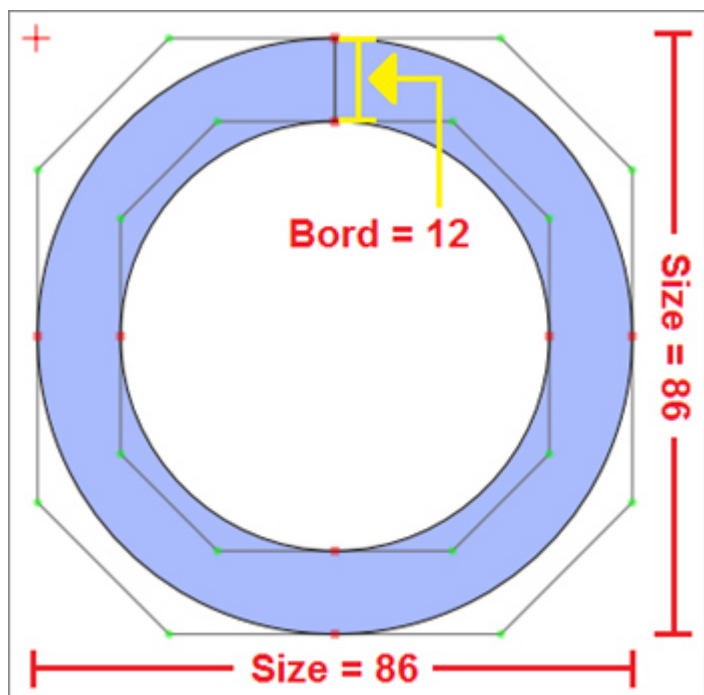
Size = {80, 100}

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:



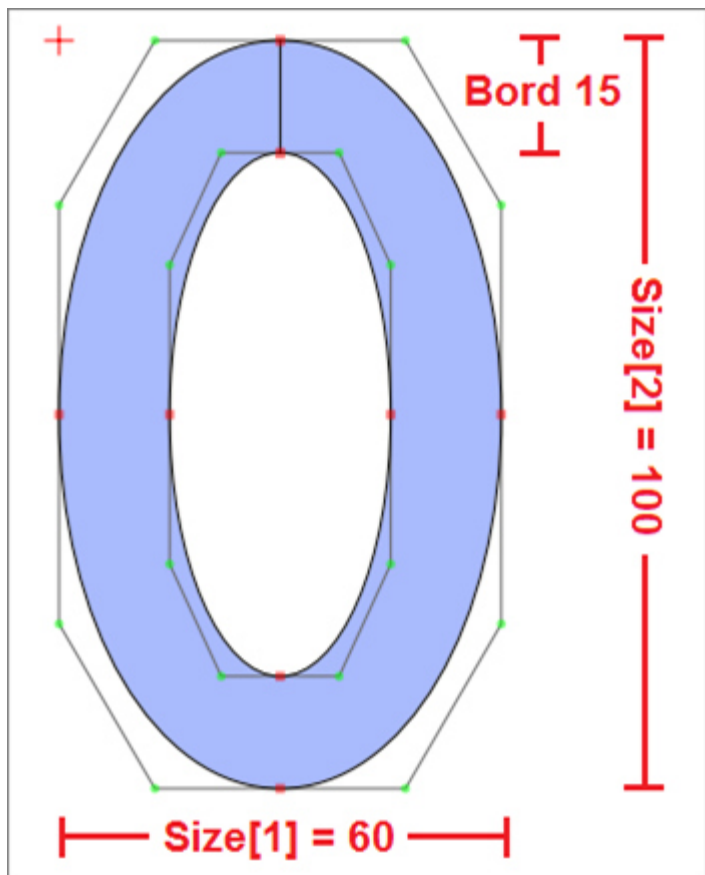
Lo que nos dará la **shape** del borde del círculo, de 86 px, tanto de ancho como de alto y con un borde de 12 px. Lo que en geometría se conoce como corona circular:



Ejemplo:

```
>> Return [fx]:  
shape.bord( shape.circle, 15, {60, 100} )
```

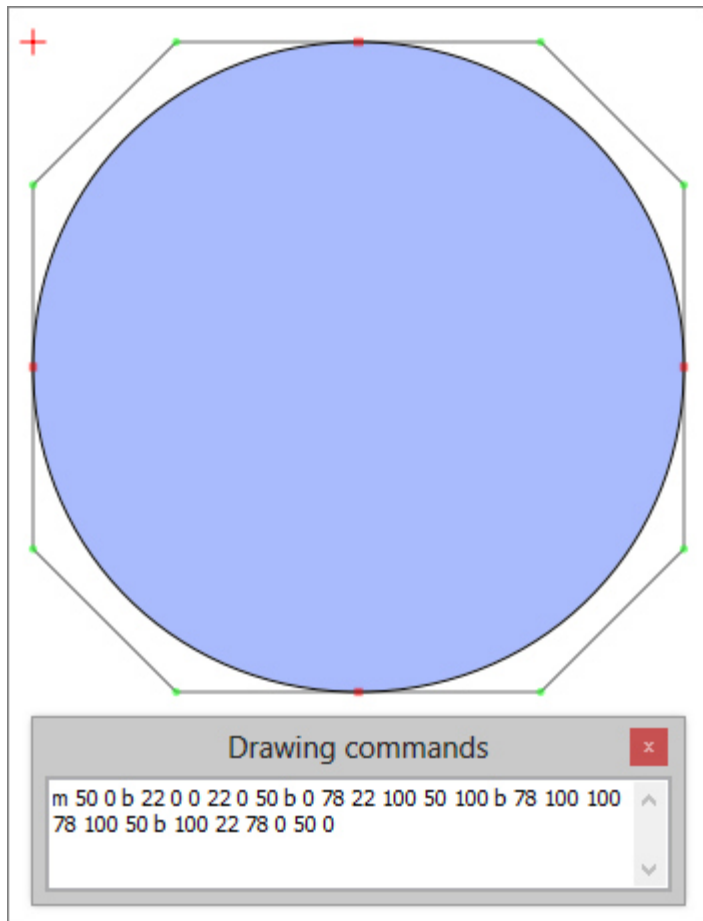
En este ejemplo, **Size** es una tabla, en donde el 60 indica el ancho de la **shape** y 100 será la altura medida también en pixeles:

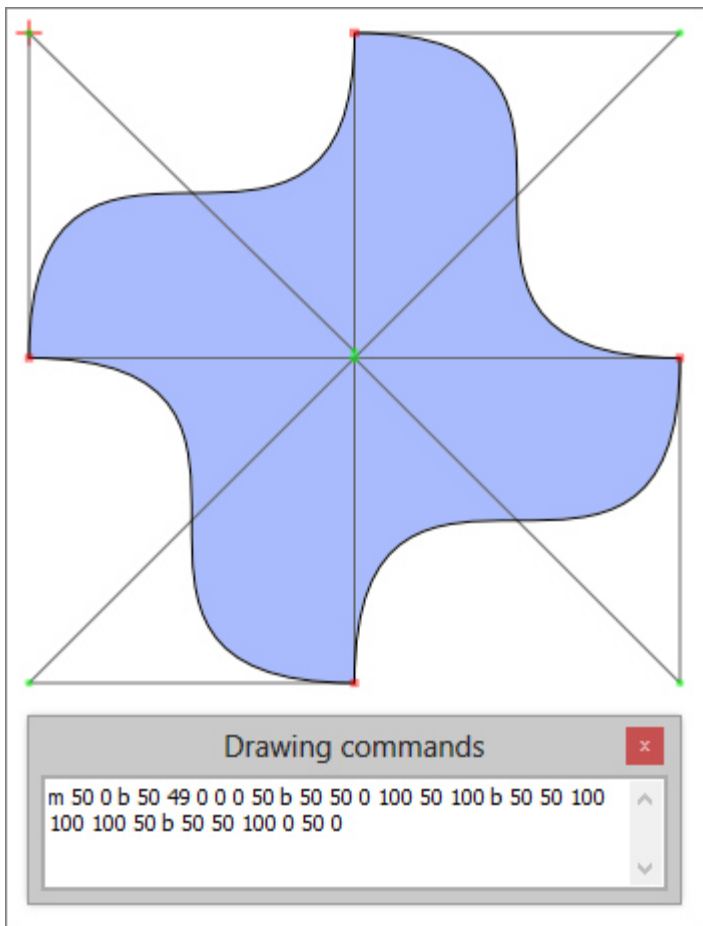


shape.morphism(Size, Shape1, Shape2) :

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 →

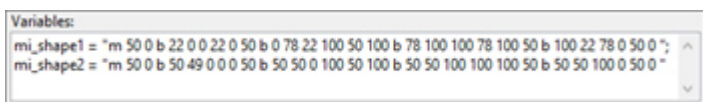
El parámetro **Size** debe ser un número entero mayor a 2 que indicará el tamaño de la tabla que se retornará y por ende indica la cantidad de Shapes que se crearán en la interpolación de una shape a la otra.



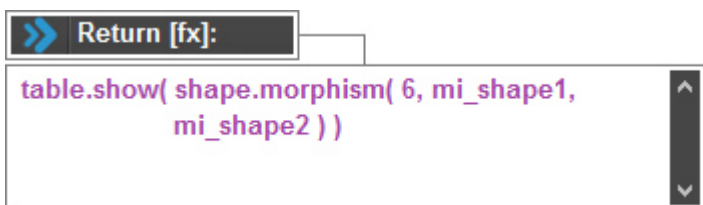


Ejemplo:

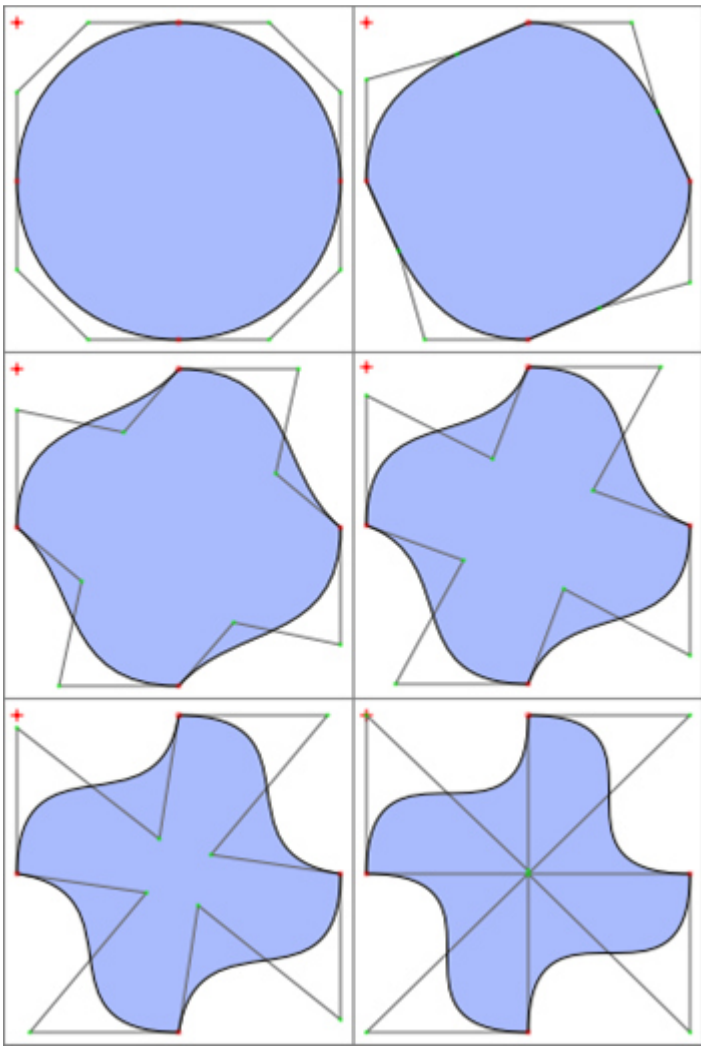
Definimos las dos Shapes anteriores en **Variables**:



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**:



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



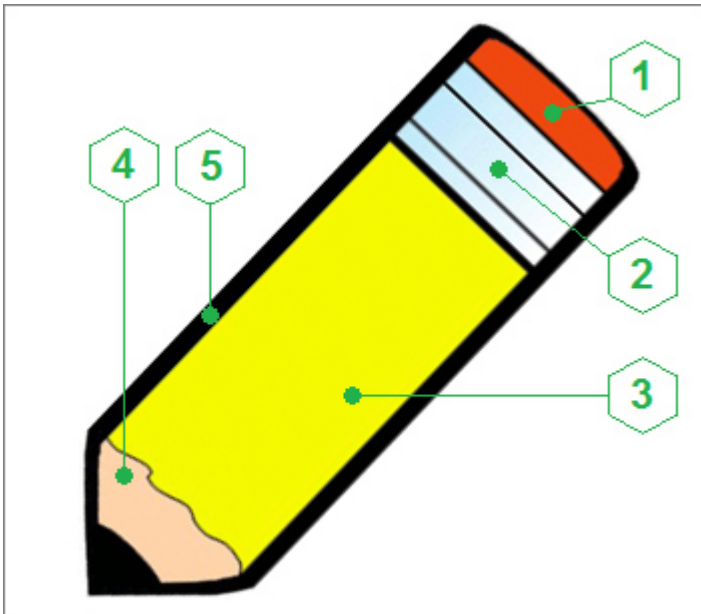
Las Shapes contenidas en este tipo de tabla es ideal para ser usada en la función **shape.animated2**, para que la transición de una **shape** a la otra se vea una animación, una transformación de la una a la otra.

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.

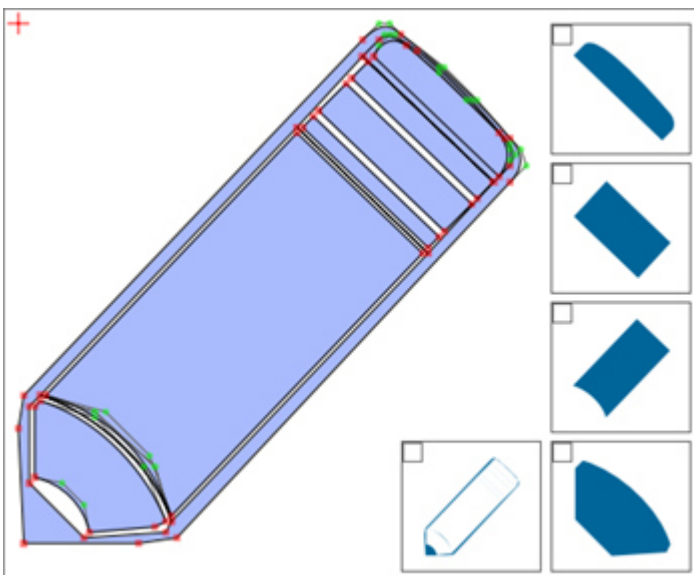
El parámetro **Mark** es opcional y hace referencia a 2 líneas paralelas que “enmarcan” a cada una de las Shapes que componen a la shape ingresada.

Ejemplo:

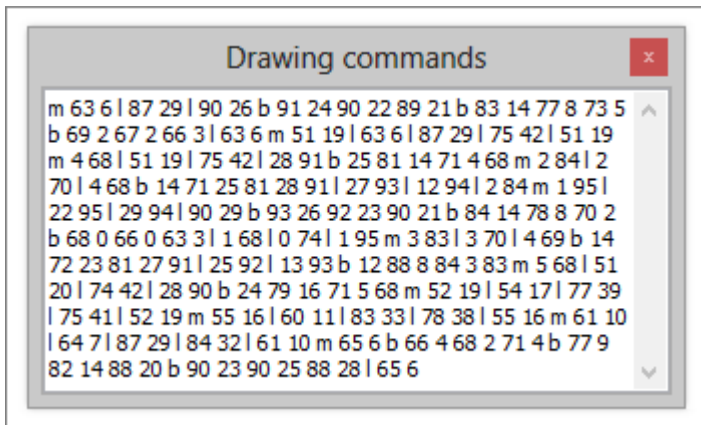
Para el siguiente ejemplo usaré la imagen de este lápiz que está compuesto por cinco Shapes individuales y tiene cinco colores diferentes:



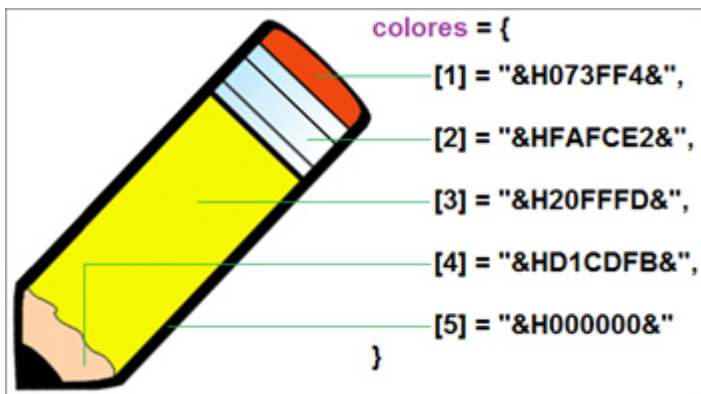
Dibujamos las cinco Shapes individuales en el mismo plano en el **ASSDraw3**. A la derecha de la imagen vemos lo que serían al dibujarlas en planos diferentes:



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:



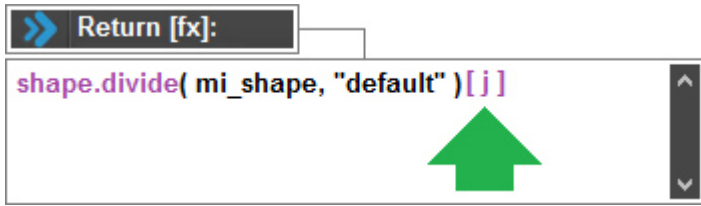
El siguiente paso es modificar el **loop** de nuestro efecto. El valor que debemos colocar será el de la cantidad de Shapes individuales que conforman a nuestra **shape**, en este caso será 5:



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:

