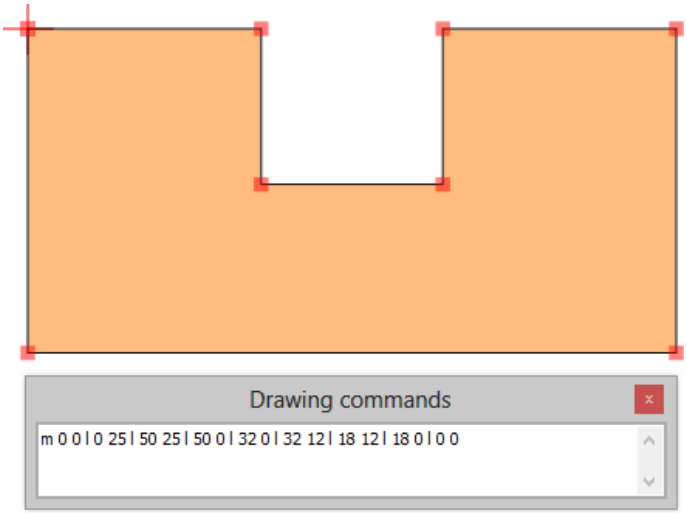
Kara Effector 3.2:

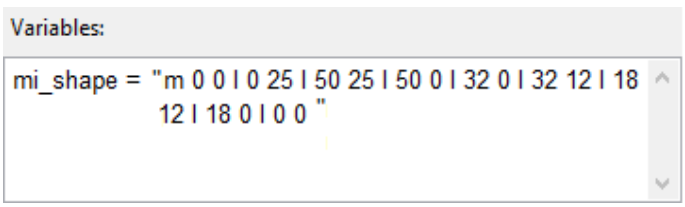
El **Tomo XV** es la continuación de la librería **shape**, ya que en el **Tomo** anterior vimos el listado de las Shapes que trae por default el **Kara Effector**, más un par de funciones de la Librería.

Librería Shape [KE]:

Para los siguientes ejemplos en las definiciones de las funciones de esta Librería, usaré esta simple **shape** que mide 50 X 25 px, pero ustedes pueden usar la que quieran y aun así los resultados deben ser visibles:



Y para mayor comodidad, la declaro en "**Variables**":

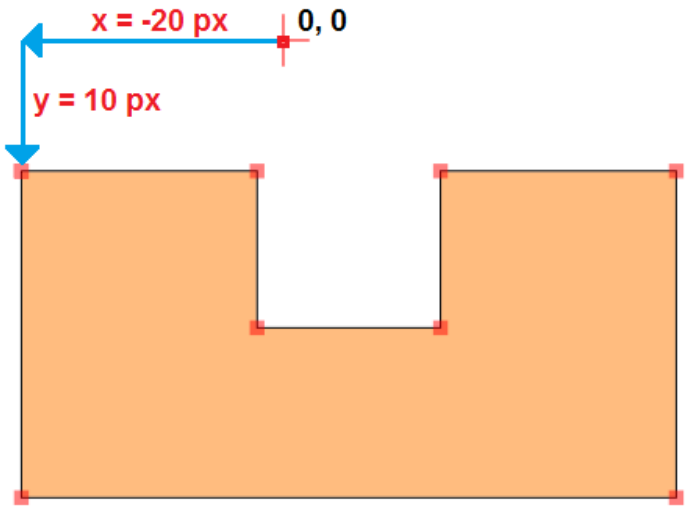


shape.displace(shape, x, y): desplaza la **shape** tantos pixeles respecto a ambos ejes cartesianos, como le indiquemos en los parámetros **x** e **y**. Ejemplo:

Return [fx]:

```
shape.displace( mi_shape, -20, 10 )
```

Retorna la misma **shape**, pero desplazada 20 pixeles a la izquierda (x = -20) y 10 pixeles hacia abajo (y = 10):



Drawing commands

```
m -20 10 l -20 35 l 30 35 l 30 10 l 12 10 l 12 22 l -2 22 l -2 10 l -20 10
```

Recordemos que en el **AssDraw3** y en el formato .ass, el eje positivo de “y” es hacia abajo del eje “x” y el negativo, hacia arriba del mismo.

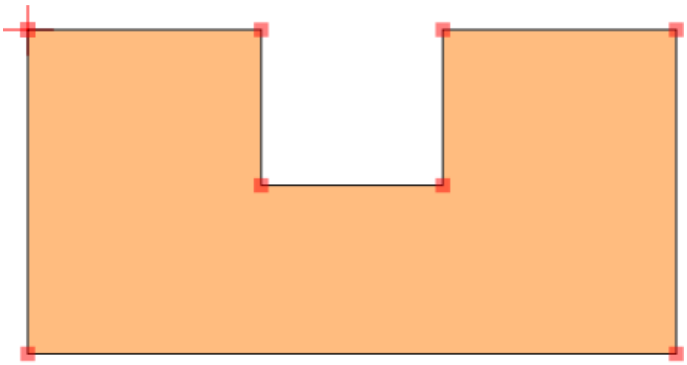
---

shape.origin(shape): desplaza la **shape** tantos pixeles respecto a ambos ejes cartesianos, hasta ubicarla en el **Cuadrante IV** del plano en el **AssDraw3**, respecto al punto de origen **P = (0, 0)**. Para el siguiente ejemplo usaré la **shape** desplazada del ejemplo anterior:

Return [fx]:

```
shape.origin( "m -20 10 l -20 35 l 30 35 l 30 10 l 12 10 l 12 22 l -2 22 l -2 10 l -20 10 " )
```

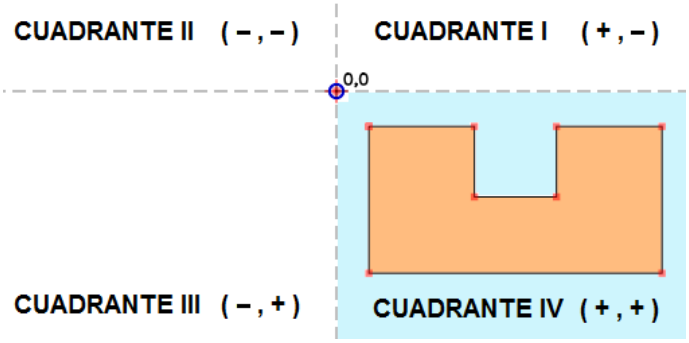
Entonces la función la ubicará en el **Cuadrante IV**:



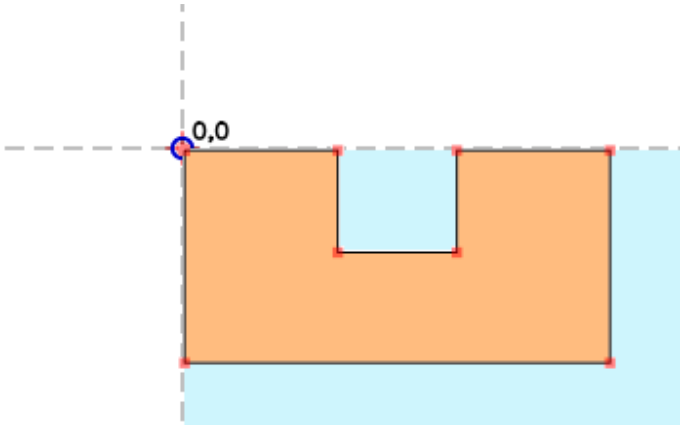
Drawing commands

```
m 0 0 l 0 25 l 50 25 l 50 0 l 32 0 l 32 12 l 18 12 l 18 0 l 0 0
```

En la siguiente imagen se muestran los **Cuadrantes** del **AssDraw3** y los signos que tienen ambas coordenadas en dichos Cuadrantes:



Y la función **shape.origin** desplaza la **shape**, en donde quiera que esté en el plano, al origen del **Cuadrante IV**, que es el Cuadrante en donde ambas coordenadas son positivas. Cuando una **shape** está ubicada en el origen del **Cuadrante IV**, se hace más sencillo aplicarle los tags de modificación y los resultados serán los esperados:



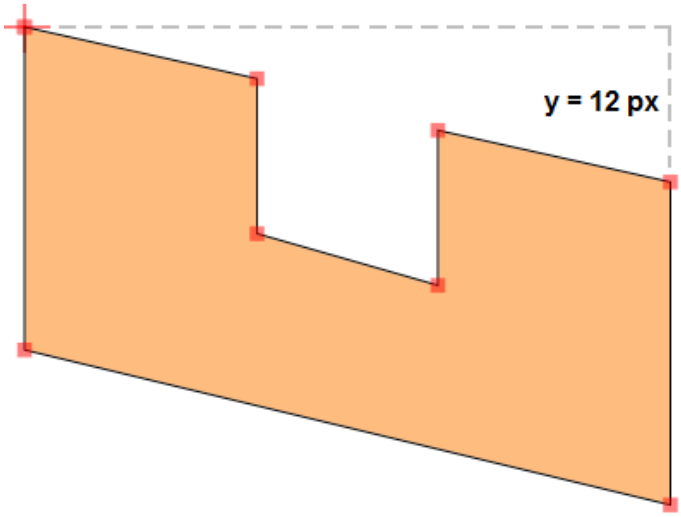
shape.oblique(shape, Pixels, Axis): deforma la **shape** de manera oblicua, en tantos pixeles indicados en el parámetro **Pixels**, respecto al eje asignado **Axis**.

- Ejemplo 1:

Return [fx]:

```
shape.oblique( mi_shape, 12, "y" )
```

Entonces la función deforma la **shape** 12 pixeles positivos respecto al eje "**y**":



Drawing commands

```
m 0 0 l 0 25 l 50 37 l 50 12 l 32 8 l 32 20 l 18 16 l 18 4 l 0 0
```

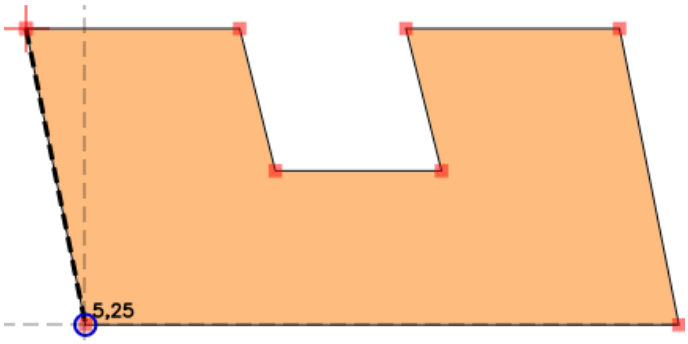
Todas las coordenadas en "x" se conservan, y las de "y" son desplazadas de forma progresiva hasta que aquellas que acompañan a las coordenadas "x" más alejadas del origen, se desplacen la cantidad de pixeles asignados en el parámetro **Pixels** (12 px). **Pixels** también puede ser un valor negativo, lo que deformaría la **shape** hacia arriba.

- Ejemplo 2:

Return [fx]:

```
shape.oblique( mi_shape, 5, "x" )
```

En este caso, la **shape** se deformará 5 pixeles hacia la derecha, dado que **Pixels** es positivo:



Drawing commands

```
m 0 0 l 5 25 l 55 25 l 50 0 l 32 0 l 35 12 l 21 12 l 18 0 l 0 0
```

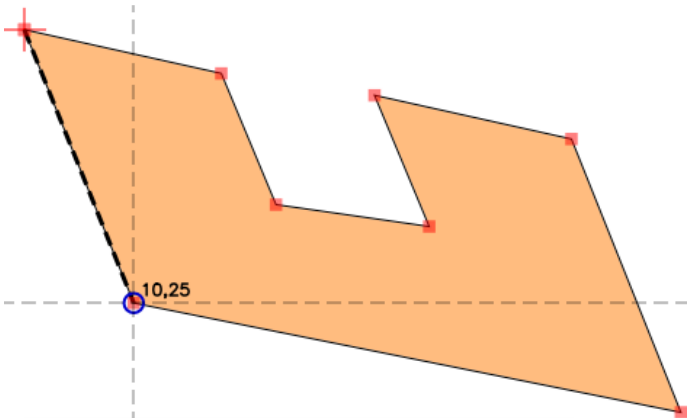
En el anterior ejemplo, las coordenadas que se conservan intactas en las **shape** son las de "y", y las coordenadas de "x" se deforman de forma progresiva.

- Ejemplo 3:

Return [fx]:

```
shape.oblique( mi_shape, 10 )
```

Ahora, al no poner el parámetro **Axis**, entonces la función deforma la **shape** en ambos ejes, en igual cantidad de pixeles (10 px); 10 px hacia la derecha (x = 10px) y 10 px hacia abajo (y = 10 px).



Drawing commands

```
m 0 0 l 10 25 l 60 35 l 50 10 l 32 6 l 37 18 l 23 16 l 18 4 l 0 0
```

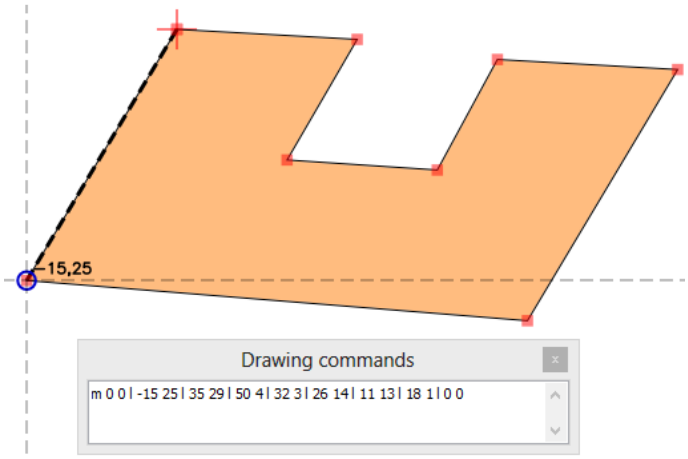
- Ejemplo 4:

Con este método podemos decidir la cantidad de pixeles en que se deformará la **shape** en ambos ejes:

Return [fx]:

```
shape.oblique( mi_shape, { -15, 4 } )
```

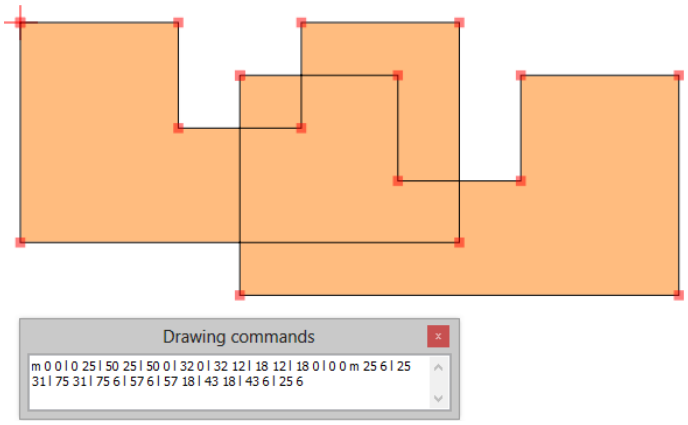
Vemos cómo la **shape** se deformó 15 pixeles a la izquierda (x = -15 px) y 4 pixeles hacia abajo (y = 4 px):



---

shape.reverse(shape): esta función reescribe la **shape** de manera que quede exactamente igual, pero dibujada a la inversa para que pueda ser sustraída de otra.

Para el siguiente ejemplo, dupliqué la misma **shape** y la desplacé varios pixeles respecto a la original, de manera que queden superpuestas como podemos ver en la siguiente imagen:

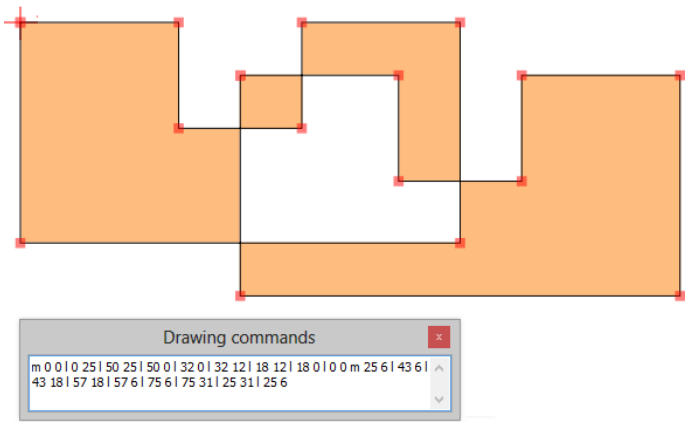


Ahora, aplicaremos la función a esa segunda **shape** para que sea redibujada de manera inversa:

Return [fx]:

```
shape.reverse( "m 25 6 l 25 31 l 75 31 l 75 6 l 57 6 l 57 18 l 43 18 l 43 6 l 25 6 " )
```

Las Shapes siguen siendo las mismas, pero el área que coincide entre ambas es sustraída, ya que una **shape** está dibujada en un sentido (sentido anti horario) y la otra a la inversa (sentido horario):



---

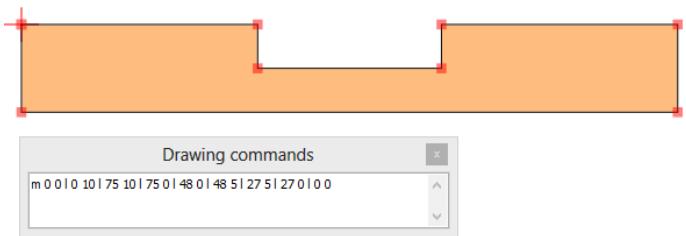
shape.ratio(shape, ratio_x, ratio_y): esta función redimensiona la **shape** en una proporción equivalente a **ratio_x** y **ratio_y**.

- Ejemplo 1:

Return [fx]:

```
shape.ratio( mi_shape, 1.5, 0.4 )
```

ratio_x = 1.5, es decir que la **shape** ahora es 1.5 veces más ancha de lo que era originalmente (150 %):



ratio_y = 0.4, es decir que la altura de la **shape** solo el 40% de la altura original.

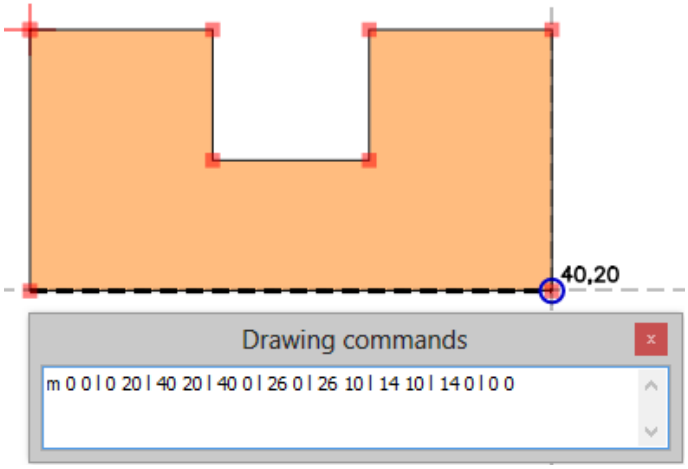
- Ejemplo 2:

En este modo se omite el parámetro **ratio_y**, entonces la función asume que **ratio_x** es la proporción del tamaño final, respecto a ambos ejes:

Return [fx]:

```
shape.ratio( mi_shape, 0.8 )
```

La **shape** que retorna es casi la misma, solo que su tamaño es un 80% (**ratio_x** = 0.8) del tamaño de la **shape** original:



---

shape.size(shape, size_x, size_y): esta función es similar a la función **shape.ratio**, pero con la diferencia que redimensiona la **shape** de tal manera que el ancho de la misma determinado según el parámetro **size_x** en pixeles, y su altura será determinada por el parámetro **size_y**, también en pixeles.

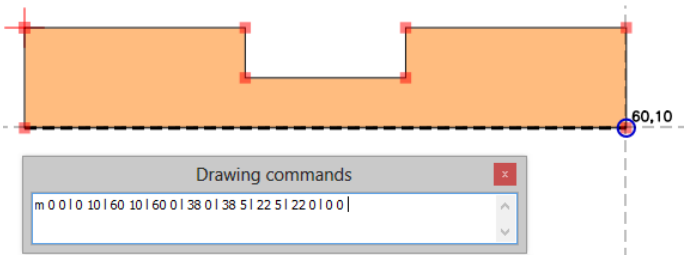
- Ejemplo 1:

De este modo, la **shape** quedará midiendo 60 X 10 px:

Return [fx]:

```
shape.size( mi_shape, 60, 10 )
```

Como se evidencia, tanto en la imagen de la **shape** como en el código de la misma, las dimensiones de la **shape** son las ingresadas en la función (60 X 10 px):



- Ejemplo 2:

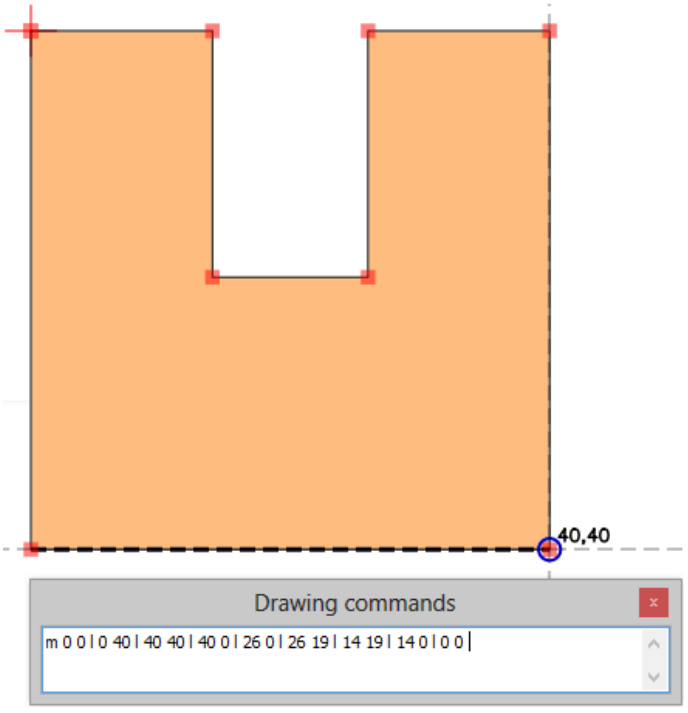
Se omite el parámetro **size_y**, de modo que la función asume que tanto el ancho como el alto de la **shape** serán el mismo, o sea **size_x**:

Return [fx]:

```
shape.size( mi_shape, 40 )
```

Usada la función de este modo, la **shape** queda con las medidas en pixeles que hayamos ingresado en la función, en este caso 40 px.

- Ancho = 40 px
- Alto = 40 px



shape.info(shape): brinda información primaria básica de la **shape**. Dicha información está dada en seis variables:

1. **minx**
2. **maxx**
3. **miny**
4. **maxy**
5. **w_shape**
6. **h_shape**

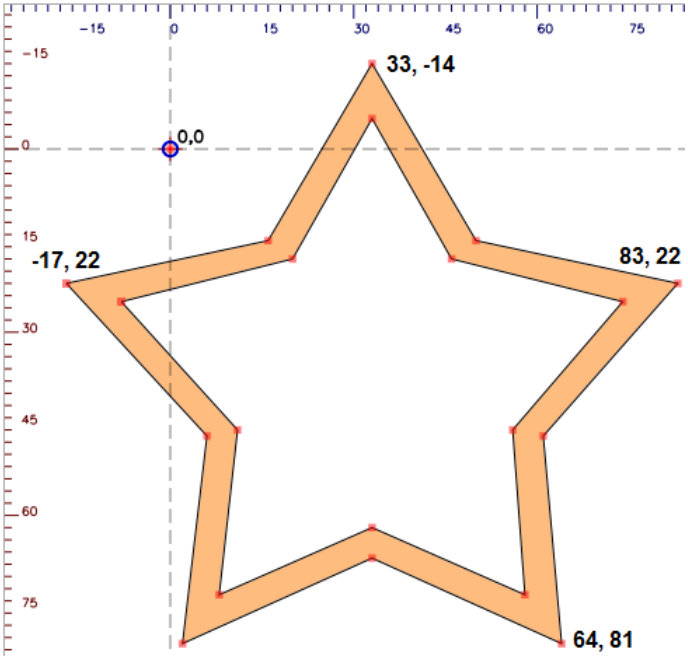
Ejemplo:

Declaramos una variable con el nombre que queramos y la igualamos a la función con una **shape**:

Variables:

```
Shape_info = shap.info( "m -17 22 l 6 47 l 2 81 l 33 67 l 64 81 l 61 47 l 83 22 l 50 15 l 33 -14 l 16 15 l -17 22 m -8 25 l 20 18 l 33 -5 l 46 18 l 74 25 l 56 46 l 58 73 l 33 62 l 8 73 l 11 46 l -8 " )
```

Esta es la shape que corresponde al código anterior:



Al llevar a cabo este procedimiento en la celda de texto "**Variables**", ya podemos usar las anteriores seis variables mencionadas, con los siguientes valores:

- **minx** = -17, que es el mínimo valor respecto a "x"
- **maxx** = 83, máximo valor en "x"
- **miny** = -14, mínimo valor en "y"
- **maxy** = 81, máximo valor en "y"
- **w_shape** = **maxx** – **minx** = 83 – (–17) = 100 px, corresponde al ancho de la shape ingresada.
- **h_shape** = **maxy** – **miny** = 81 – (–14) = 95 px, corresponde al alto de la shape ingresada.

Las anteriores seis variable ya pueden ser usadas como valores numéricos en cualquier otra celda de texto de la ventana de modificación del **Kara Effector**. Ejemplo:

Add Tags: Add Tags Language: Lua

```
"\\fscy" .. h_shape
```

Que a la postre retornará: \fscy85, dado que la altura de la **shape** ingresada era de 85 px.

Es todo por ahora para este **Tomo XV**, pero las funciones de la Librería **shape** aún no llegan a su fin. En el próximo **Tomo** continuaremos profundizando en el mundo de las Shapes y las posibilidades que nos ofrecen. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** 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, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

www.facebook.com/karaeffector