Kara Effector 3.2:

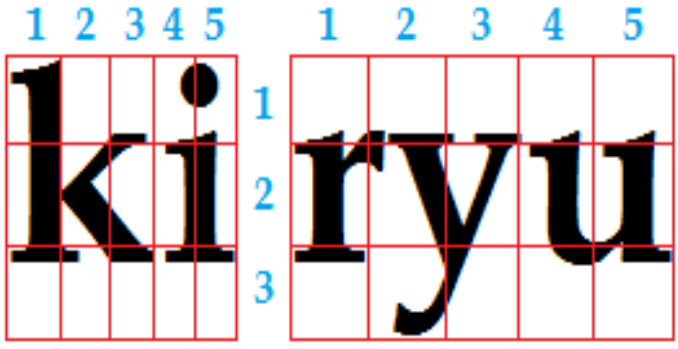
Como lo había mencionado al final de la anterior entrega, este **Tomo X** es la continuación de la librería "**tag**", que como las demás librerías del **Kara Effector**, es importante que sepamos en qué consiste cada una de sus funciones para poder sacarle el máximo provecho posible.

Sin más, retomaremos la librería "**tag**" en la función en donde nos habíamos quedado; la función **tag.clip**

Librería tag [KE]:

Veremos la forma de hacer un **clip multiple cuadrado**, es decir que las dimensiones del clip sean las mismas. Ya sabemos cómo hacer un clip reticulado o de cuadricula con la función **tag.clip** y es de la siguiente forma, ejemplo:

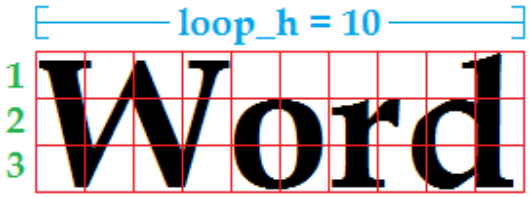
Pero este método no garantiza que los clip´s tengan las mismas dimensiones, es decir que sean cuadrados:

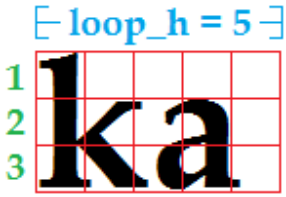


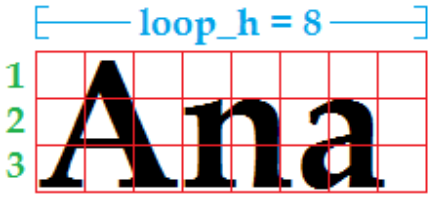
En la imagen anterior vemos cómo para algunas sílabas las porporciones de los clip's no son para nada cuadradas y eso es porque la cantidad de clip´s verticales es constante, lo que no es ideal dados los distintos anchos de la Líneas, Palabras, Sílabas, Caracteres y demás.

Entonces, para que los clip's sean cuadrados, les mostraré dos formas distintas de hacerlo. Y la primera forma de hacerlo es usando la varible **loop_h**, así:

loop_h: (variable) es un número entero calculado por el **Kara Effector**, teniendo en cuenta la cantidad de clip's verticales (del ejemplo anterior: 3), que hace que el ancho y el alto de los clip's tengan las mismas dimensiones y sean cuadrados, ejemplo:



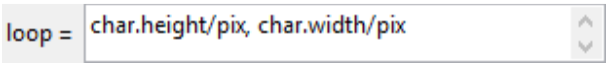




Entonces el **loop_h** varía en todos los casos con el fin de que los clip's den la función **tag.clip** sean cuadrados. La segunda forma de hacerlo es:

- Declaramos una variable con la medida en pixeles de las dimensiones de los clp's, ejemplo:

- Dependiendo del **Template Type**, por ejemplo un **Template Type: Char**, debemos escribir en la celda de texto **loop**, así:



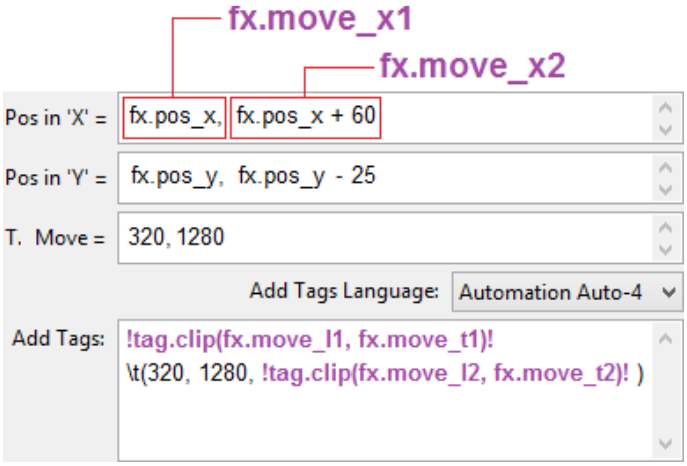
Si por ejemplo, un caracter mide 70 pixeles de ancho por 40 de alto, tendríamos:

- 40/pix = 40/10 = 4
- 70/pix = 70/10 = 7
- 4 x 7 = 28

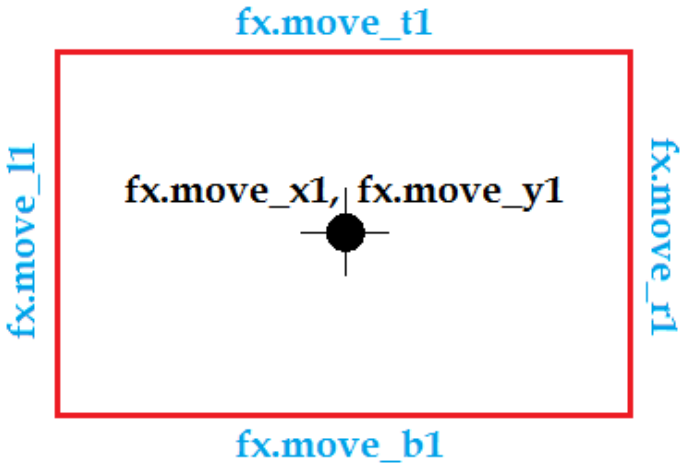
O sea que obtendíamos 28 clip's de 10 x 10 pixeles de la función **tag.clip**, que a su vez sería la cantidad del **loop**, es decir que **maxj** = 28

A menor tamaño de los clip´s, mayor será la cantidad del loop, y a moyor cantidad de clip's, mayor será la cantidad de recursos consumidos por la memoria RAM y la PC se hará mucho más lenta.

A continuación veremos cómo mover los clip's generados por la función **tag.clip** en el **Kara Effector**, usando algunas cosas que ya hemos aprendido hasta el momento.



fx.move_l1 es la coordenada de la posición izquierda de **fx.move_x1**, y **fx.move_t1** es la superior de **fx.move_y1**:



Las dimenciones del rectángulo anterior dependerán del **Template Type**, por ejemplo, si es tipo Translation Word, entonces las dimensiones serán las de cada una de cada Palabra de cada Línea.

Desde y hacia dónde se mueve un clip es desición de cada uno, según un efecto así lo requiera, lo más importante es saber cómo hacerlo y el poder contar con una función como **tag.clip** que nos facilita un poco esa labor.

---

tag.iclip(left, top, width, height, mode): similar a **tag.clip**, pero con la leve diferencia que no hace clip's sino iclip's.

Para los que aún no están familiarizados con los iclip's y en qué consisten, recordemos qué son y para qué se usan.

- **clip:** es un rectángulo con posición y dimensiones específicas que hace visible únicamente a todo lo que esté dentro de dicho rectángulo. Ejemplo:

  O sea que todo lo que está dentro del clip es lo que veremos y todo lo que esté por fuera de él quedará totalmente invisible.
- **iclip:** es un rectángulo con posición y dimensiones específicas que hace visible únicamente a todo lo que esté por fuera de dicho rectángulo. Ejemplo:

  O sea que todo lo que está por fuera del iclip es lo que veremos y todo lo que esté por dentro de él quedará totalmente invisible.

La elección entre el clip y el iclip dependerá del efecto.

---

tag.clip2(left, top, width, height): esta función es también similar a las función **tag.clip**, pero con la gran diferencia que siempre genera el mismo clip sin impotar el loop, es decir que no genera clip's verticales, horizontales ni reticulares, sino que siempre genera el mismo clip con las mismas dimensiones, es por eso que no necesita el parámetro **mode**.

---

tag.iclip2(left, top, width, height): es similar a tag.clip2, pero con la diferencia que no genera clip's sino iclip's y es la última funcion de la libería "**tag**" que emplea clip's rectangulares por medio de coordenadas.

A continuación veremos dos funciones más basadas en **clip's**, pero esta vez no serán rectangulares sino basadas en **shapes**, es decir que están enfocadas en la figuras que dibujamos en el **AssDraw3**.

tag.movevc(shape, x, y, Dx, Dy, t_i, t_f): similar a la función tag.clip con la diferencia que el clip que genera es una **shape**. Todos los parámetros de esta función, excepto el primero (**shape**), pueden tener valores por default. Veamos qué son y en qué consisten:

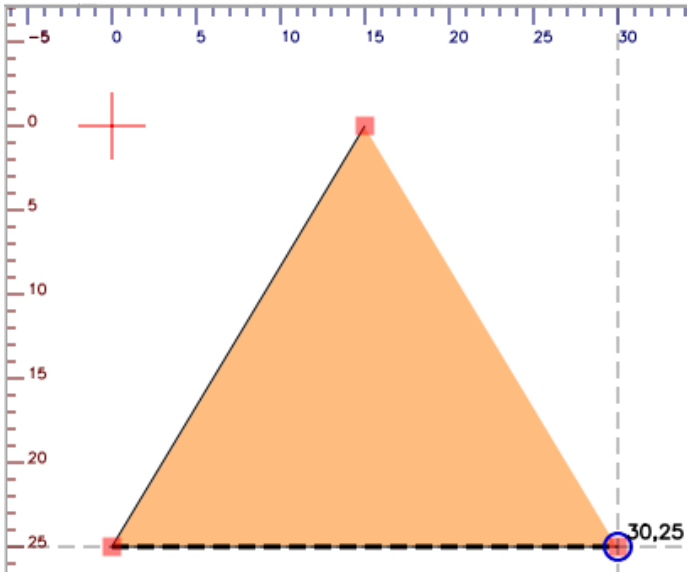
- **x:** coordenada con respecto al eje "x" que hace referencia a la posición en donde estará el centro de la **shape**, su valor por default es **fx.move_x1**
- **y:** coordenada con respecto al eje "y" que hace referencia a la posición en donde estará el centro de la **shape**, su valor por default es **fx.move_y1**
- **Dx:** cantidad de desplazamiento en pixeles con respecto al centro de la **shape**, con referencia al eje "x". Su valor por deafault es **fx.move_x2 – fx.move_x1**

  De no existir un **fx.move_x2**, recordemos que su valor por default es **fx.move_x1**, por lo que el valor por default del parámetro **Dx**, en este caso, sería: **fx.move_x1 – fx.move_x1 = 0**
- **Dy:** cantidad de desplazamiento en pixeles con respecto al centro de la **shape**, con referencia al eje "y". Su valor por deafault es **fx.move_y2 – fx.move_y1**

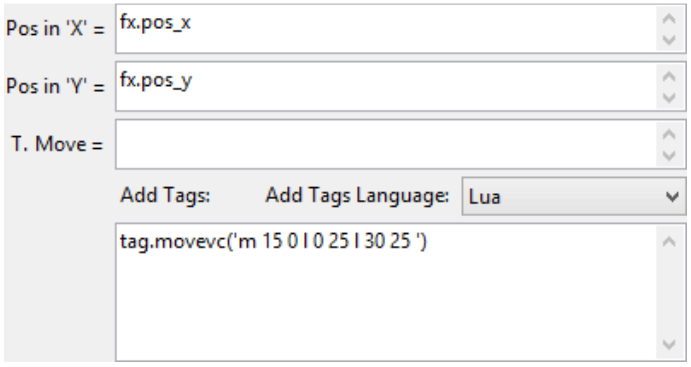
  De no existir un **fx.move_y2**, recordemos que su valor por default es **fx.move_y1**, por lo que el valor por default del parámetro **Dy**, en este caso, sería: **fx.move_y1 – fx.move_y1 = 0**

- **t_i:** es el tiempo en que iniciará el movimiento del clip, en caso que decidamos que se mueva. Su valor por default es **fx.movet_i**
  De no existir un **fx.movet_i**, recordemos que su valor por default es 0, por lo que en este caso el valor por default de **t_i** sería 0.
- **t_f:** es el tiempo en que finalizará el movimiento del clip, en caso que decidamos que se mueva. Su valor por default es **fx.movet_f**
  De no existir un **fx.movet_f**, recordemos que su valor por default es **fx.dur**, por lo que en este caso el valor por default de **t_f** sería **fx.dur**

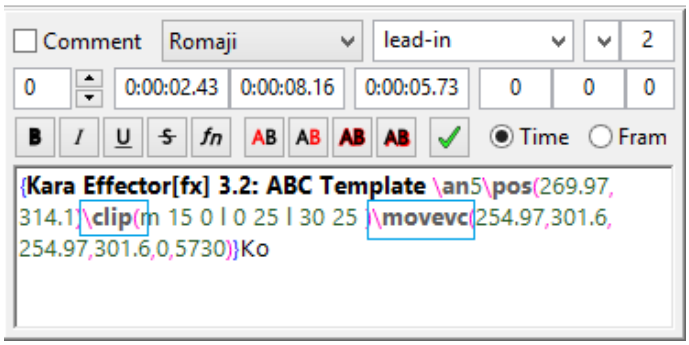
Para el siguiente ejemplo usaré un simple triángulo hecho en el **AssDraw3**, como lo podemos ver en esta imagen:



El triángulo tiene 30 pixeles de ancho por 25 de alto. El siguiente paso es copiar el codigo de esa **shape** y pegarlo dentro de la función **tag.movevc** y dejaremos el resto de los parámetros por deafult. He usado un **Template Type: Syl**, pero pueden usar caulquiera de los de la lista:



La función **tag.movevc** siempre retorna dos tags, un **\clip** que contiene dentro de sí a la **shape** y un **\movevc** que le da la posición al **clip** y lo mueve si ese fuere el caso:

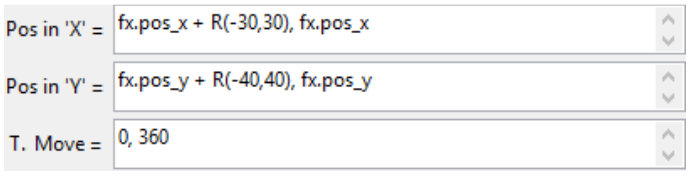


Como el tag de posición es un **\pos** entonces el **\movevc** solo posiciona al clip. Ahora veremos cómo queda el clip del triángulo en cada una de las Sílabas de la Línea:

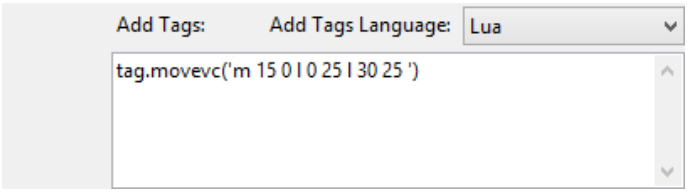
Como el triángulo es mucho más pequeño que el tamaño de cada una de las sílabas, entonces éstas no alcanzan a ser totalmente visibles:

En cuanto el clip y el objeto karaoke al que afecta (ya sea una sílaba, una línea, carácter, palabra, shape o demás) se muevan al mismo tiempo, desde el mismo punto de inicio y hacia el mismo punto final; lo recomendable es dejar el resto de los parámetros de la función **tag.movevc**, por default. Ejemplo:

Hacemos un **\move** con posiciones iniciales random:



Usaremos la función **tag.movevc** con la misma shape y el resto de los parámetros por default:



De este ejemplo tendríamos que:

- **x** = fx.pos_x + R(-30,30)
- **y** = fx.pos_y + R(-40,40)
- **Dx** = fx.pos_x - fx.pos_x + R(-30,30) = R(-30,30)
- **Dy** = fx.pos_y - fx.pos_y + R(-40,40) = R(-40,40)
- **t_i** = 0
- **t_f** = 360

Y el resultado sería:

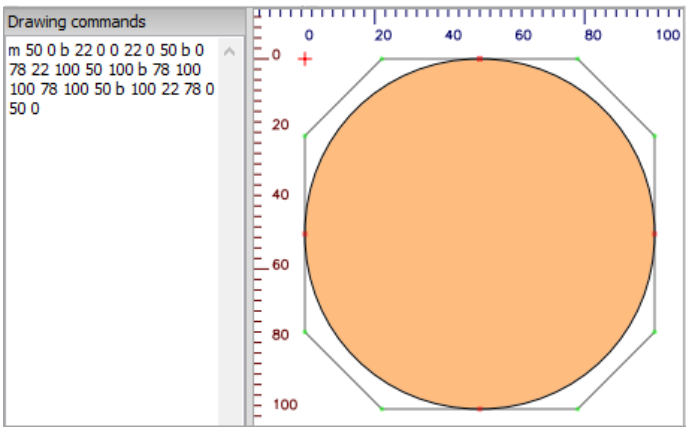
Que luego de 360 ms las sílabas y los clip´s quedaran en su posición narutal:

El tag **\movevc** solo es aplicable con el filtro **VSFilterMod**, de otro modo no se verá el efecto. El **\movevc** solo funciona de la mano de los tags **\pos** y **\move**, pero no lo hace con los tags **\moves3**, **\moves4** y **\mover**. En pocas palabras, el tag **\movevc** solo puede mover a un clip en línea recta, sin importar la dirección.

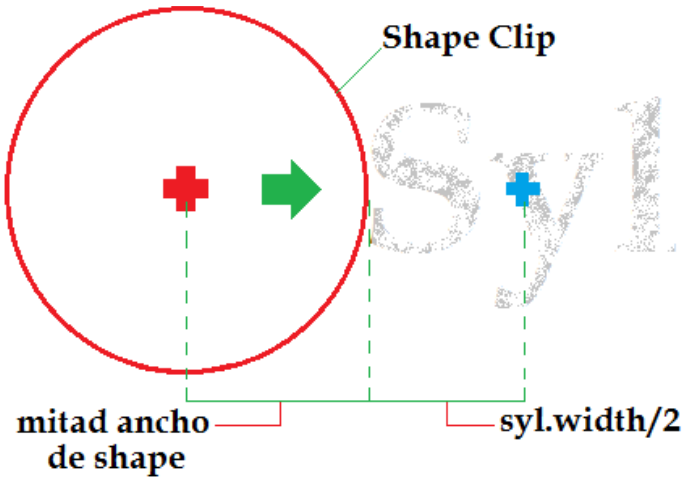
Ahora veremos un ejemplo de objeto estático y clip móvil, es decir, en donde tengamos que usar el resto de los parámetros de la función **tag.movevc**:

En un **Template Type: Syl**, le dejamos las posiciones por default que equivalen a **syl.center** y **syl.middle**:

La **shape** que usaremos para el clip es un círculo de 100 px de ancho, dicha **shape** hace parte de las librerías del **Kara Effector** y se llama: **shape.circle**



Y la posición inicial del clip será justo al lado izquierdo de cada sílaba, como muestra la siguiente imagen:

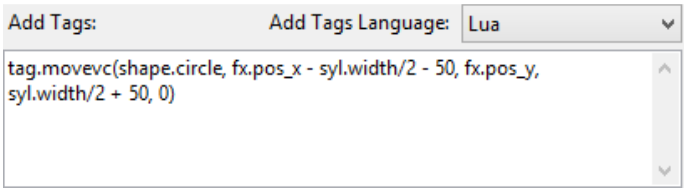


Dicha posición inicial de clip sería:

- **fx.pos_x – syl.width/2 – 50, fx.pos_y**

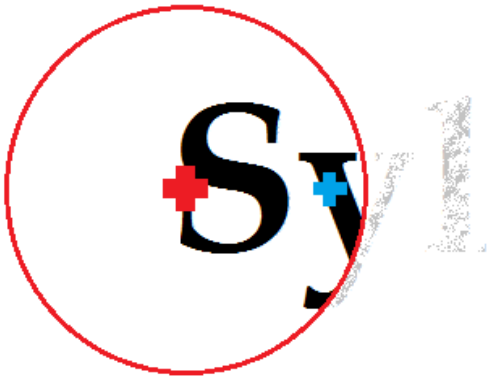
Y la posición final del clip será:

- **fx.pos_x, fx.pos_y**



- **shape = shape.circle**
- **x = fx.pos_x – syl.width/2 – 50**
- **y = fx.pos_y**
- **Dx = syl.width/2 + 50**
- **Dy = 0**

Entonces, el círculo que hace de clip irá avanzando desde donde estaba inicialmente hasta que su centro coincida con el de la sílaba:



Y cuando se cumpla el tiempo total de la línea fx, la shape y la sílaba tendrán el mismo centro:

La función **tag.movevc** nos será de gran ayuda para hacer efectos con clip's de alto nivel, ya que de otra forma sería muy complicado lograr el control en cuanto a posición y movimiento cuando de clip's se trata.

---

tag.movevci(shape, x, y, Dx, Dy, t_i, t_f): similar a **tag.movevc**, pero con la diferencia que no retorna un clip, sino un iclip.

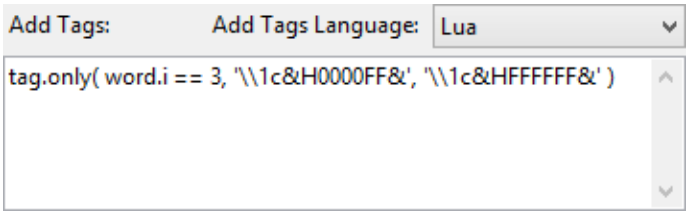
---

tag.only(condition, exit_t, exit_f): esta función retorna alguno de los dos parámetros (**exit_t** o **exit_f**) según el valor de verdad del parámetro **condition.** Si el valor de verdad de **condition** es verdadero (**true**), retorna al parámetro **exit_t**, si es falso (**false**) retorna **exit_f**.

Veamos los tipos de condiciones que podemos usar en la función **tag.only**:

- == igual a
- **~=** diferente a
- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que

Ejemplo:



Esto quiere decir que, si la Palabra en la línea es la número tres, su color primario será **Rojo** ('\\1c&H0000FF&'), pero si dicha condición es falsa, su color primario será **Blanco** ('\\1c&HFFFFFF&'):

En la función **tag.only** solo el tercer parámetro (**exit_f**) puede tener un valor por default, y este valor depende del tipo de objeto que sea el segundo parámetro (**exit_t**). Si **exit_t** es un **string**, entonces el valor por default de **exit_f** es vacío (""), y si **exit_t** es un **número**, el valor por default de **exit_f** es cero (0). Ejemplos:

- **tag.only(syl.i <= 5, "\\blur3")**
  Como **exit_t** es un **string** ("\\blur3"), entonces el valor por default de **exit_f** será vacío. Esto quiere decir que, si la sílaba es una de las cinco primeras de la línea, la función retornará "\\blur3", de lo contrario no retornará nada.
- **l.end_time + tag.only(char.i == char.n, 1200)**
  Como **exit_t** es un **número** (1200) entonces el valor por default de **exit_f** será cero (0). Esto quiere decir que, si el caracter es el último de la línea (char.n), se sumarán 1200 ms al tiempo final de la línea, de lo contrario se le sumará cero.

Los seis tipos de condiciones vistos anteriormente tienen un único valor de verdad, ya sea verdadero (**true**) o falso (**false**), pero no puede tener ambos valores al mismo tiempo. Estas seis condiciones son conocidas como **condiciones simples** y al combinar dos o más de ellas se crean las **condiciones compuestas**.

Condiciones simples:

- == igual a
- **~=** diferente a
- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que

Y para obtener las **condiciones compuestas** es necesario usar **conectores**, la **conjunción** (**and**) o la **disyunción** (**or**). Cada una de las dos mencionadas también tiene un único valor de verdad que dependerá del valor de verdad de cada una de las condiciones que la conforman.

Tabla de verdad de la **conjunción**:

Caso	p	q	p and q
1	V	V	V
2	V	F	F
3	F	V	F
4	F	F	F

Tabla de verdad de la **disyunción**:

Caso	p	q	p or q
1	V	V	V
2	V	F	V
3	F	V	V
4	F	F	F

Ejemplos:

- (syl.i >= 7) **and** (syl.dur < 300)
- (R(2) == 1) **or** (char.width > 80)
- (syl.text **~=** "ke") **and** (syl.i < 5)
- (word.n < 10) **or** (line.width > 600)

Las posibles combinaciones son muchas, así que el tener claro qué es lo que queremos en nuestro efectos nos ayudará a decidirnos por alguna de ellas.

A continuación están las posibles **combinaciones** entre solo dos **condiciones** por medio de alguno de los dos **conectores** (**and** u **or**):

Caso	Cond. 1	Conector	Cond.2
1	==	and // or	==
2	==	and // or	~=
3	==	and // or	<
4	==	and // or	>
5	==	and // or	<=
6	==	and // or	>=
7	~=	and // or	~=
8	~=	and // or	<
9	~=	and // or	>
10	~=	and // or	<=
11	~=	and // or	>=
12	<	and // or	<
13	<	and // or	>
14	<	and // or	<=
15	<	and // or	>=
16	>	and // or	>
17	>	and // or	<=
18	>	and // or	>=
19	<=	and // or	<=
20	<=	and // or	>=
21	>=	and // or	>=

21 posibles combinaciones, pero no se preocupen, no es necesario memorizarlas todas, ya que el solo hecho de saber el valor de verdad de los **conectores** (**and** u **or**) es más que suficiente y el resto es solo cuestión de aplicar un poco de lógica al asunto.

Con el final de la función **tag.only** se da por terminado el **Tomo X**, pero no la librería "tag", ya que aún nos resta por ver unas cuantas funciones más de ella.

En el **Tomo XI** del **Kara Effector** continuaremos con el resto de la Librería "**tag**". 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. También pueden visitarnos en nuestra página de **Facebook**: www.facebook.com/karaeffector