
Kara Effector 3.2:

En el **Tomo VI** veremos un poco más de los lenguajes **LUA** y **Automation Auto-4**, ya que el saber más de ellos nos ayudará a comprender la estructura de cualquier Efecto hecho en el **Kara Effector** o incluso, cualquier otro hecho en **Aegisub**.

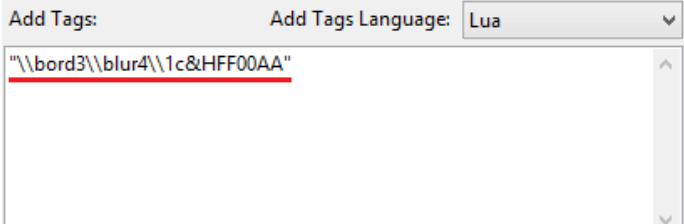
Entender un Efecto es el primer paso para desarrollar uno propio, tanto en el **Aegisub** o en **Kara Effector**. Espero que lo visto en este Tomo sea de fácil comprensión para todos.

OBJETOS

Para hacer un Efecto Karaoke o una simple Plantilla con tags para un archivo de Subtítulos en lenguaje **LUA**, en el **Kara Effector** o en **Automation Auto-4**, solo existen dos **HERRAMIENTAS**, dichas Herramientas reciben el nombre de “strings” y “values”.

OBJETOS: STRING

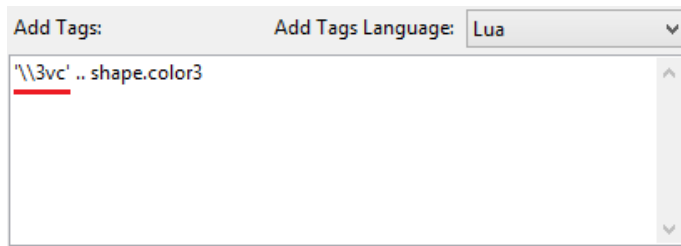
Un **string** (cadena) es un caracter o sucesión de ellos que carecen de algún tipo de valor. En lenguaje **LUA** un **string** es todo aquello que esté *entre comillas*, ya sean dobles o simples. Veamos algunos ejemplos de strings en **LUA**:



The screenshot shows a software interface with two labels at the top: "Add Tags:" and "Add Tags Language:". The "Add Tags Language:" label has a dropdown menu currently set to "Lua". Below these labels is a large text input area. Inside this area, the string `"\\bord3\\blur4\\1c&HFF00AA"` is entered and highlighted with a red underline. To the right of the text input area is a vertical scrollbar with up and down arrow buttons.

En este caso, todo aquello que está entre las comillas dobles es un string, ya que para el lenguaje **LUA** esto sería lo mismo que una palabra cualquiera, sin valor alguno.

En el siguiente ejemplo veremos los dos tipos de **Objetos**:



Lo que está entre las comillas simples es un **string**, el resto es **value**.

En **LUA**, si algo está entre comillas es un **string**, incluso si es un número o si hay operaciones matemáticas, ejemplo:

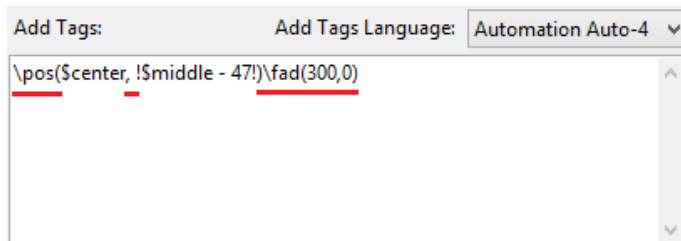
"12 + 3"

Al estar entre comillas no se realiza la operación (suma).

12 + 3

Ya sin las comillas, **LUA** ejecuta la operación y devuelve el resultado (15).

Por otro lado, en **Automation Auto-4**, un **string** también es fácil de reconocer. Un **string** es todo aquello que no está precedido del signo dólar (variables dólar) ni tampoco está dentro de los signos de admiración. Ejemplos:



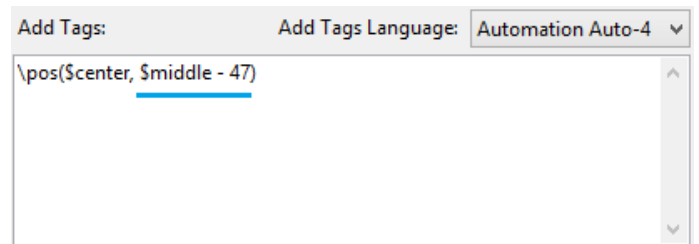
De esta expresión, los únicos **Objetos** que no son un **string** en lenguaje **Automation Auto-4** son:

\$center

!\$middle - 47!

El resto, incluidos los paréntesis y las comas, son un **string** y por ende carecen de valor operacional. Podemos afirmar que los tags con que hacemos los Efectos también son un **string**, aunque para el **Aegisub** sí tengan valor operacional o funcional, ya que ni en **LUA** ni en **Automation Auto-4** se pueden hacer operaciones con ellos.

Un **error** común que se comete a veces en el lenguaje **Automation Auto-4** es intentar hacer una operación sin poner los signos de admiración:



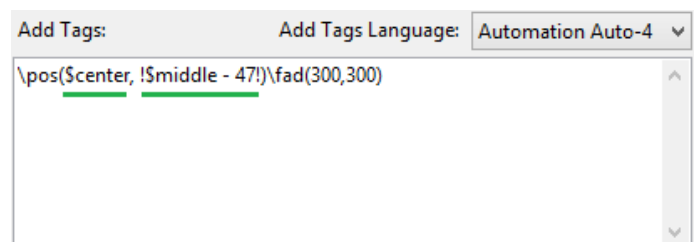
Al no poner los signos de admiración, el **Kara Effector** da por sentado que estamos escribiendo un string y por ello no ejecuta la operación que habíamos pensado:



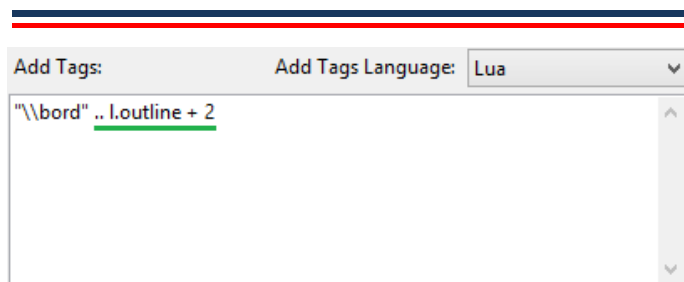
Se ve claramente cómo queda expresada la sustracción (resta), pero no devolvió el resultado de la misma. **Aegisub** toma en cuenta solo al 42 y omite el resto.

OBJETOS: VALUE

El **value** es todo lo que tiene algún tipo de valor, ya sea numérico, operacional o funcional. Un **value** es todo lo que no es un **string**. En **LUA** un **value** es todo aquello que no está dentro de las comillas simple o dobles. De forma similar, en **Automation Auto-4**, un **value** es todo aquello que no está precedido del signo dólar (variable dólar) ni tampoco está dentro de los signos de admiración.



En verde podemos ver dos ejemplos de **values** (valores) en lenguaje **Automation Auto-4**.



En **LUA**, como ya lo sabemos, sino está dentro de las comillas, en este caso dobles, es un **value**. Los dos puntos seguidos (..) tienen valor operacional (concatenación) y la variable **l.outline** tiene un valor numérico (espesor del borde medido en pixeles).

Ahora veamos un cuadro con los diferentes tipos de **value** y algunos ejemplos:

| TIPO DE VALUE | EJEMPLOS |
|-----------------|--|
| Numérico | Pi, e, 1276, syl.center, line.middle |
| Operación | +, -, /, *, ^, .., %, : |
| Función | math.random, string.format |
| Tabla | A = {1, 2, 'F'} B = {"&HFF&", "&H00&"} C = {[1] = 43; [2] = 86} D = { } |
| Operador Lógico | for, if, while, repeat, or, and |
| Variable | Color1 = "&HFF0000&" Radius = 6*pi Linevc = "" |

Una vez claro los conceptos de **string** y **value** ya podemos seguir viendo las Librerías que aun nos faltan. La que veremos a continuación es referente a los **strings** y con las funciones que hay en ellas notarán que los **string** son más que simples objetos que carecen de valor.

Librería "string" [LUA]

string.byte(s)

Retorna el **Valor Numérico** del caracter **s** según su equivalente decimal en la [Tabla ASCII](#)
Veamos un ejemplo de la Tabla:

| | | | |
|----|---|-----|---|
| 64 | @ | 96 | ` |
| 65 | A | 97 | a |
| 66 | B | 98 | b |
| 67 | C | 99 | c |
| 68 | D | 100 | d |

Vemos que el valor decimal de “@” es 64, así como el de “b” es 98.

| | | | | | | | | | | | | | | | | | | | | | |
|------------------------------|--|---|---|---|---|---|---|---|---|--|---|---|---|---|---|---|---|---|---|--|--|
| | <p>Ejemplo 1: string.byte("A") = 65</p> <p>Ejemplo 2: Str = "C" string.byte(Str) = 67</p> <p>Los valores decimales de la Tabla ASCII van desde el 0 hasta el 255. El modo abreviado de usar esta función es así:</p> <p>Ejemplo 3: s = "B" s:byte() = 66</p> <p>Ejemplo 4: s = "d" s:byte() = 100</p> <p>Esta función tiene más modo de uso, pero al menos por el momento no serán de gran utilidad y de ahí que las haya omitido.</p> | | | | | | | | | | | | | | | | | | | | |
| string.char(...) | <p>Retorna el caracter asignado por la Tabla ASCII de un número entre 0 y 255. Ejemplo:</p> <p>string.char(65) = "A" string.char(66) = "B" string.char(65, 66, 100) = "ABd"</p> | | | | | | | | | | | | | | | | | | | | |
| string.find(s, ptr) | <p>Retorna las posiciones del inicio y final del string s dentro de un string mayor ptr.</p> <p>Ejemplo 1: L = "Demo lua string" s = "lua"</p> <p>string.find(s, L) = 6, 8 6 es el inicio y 8 es el final:</p> <table border="1"><tr><td>D</td><td>e</td><td>m</td><td>o</td><td></td><td>l</td><td>u</td><td>a</td><td></td><td>s</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td></td><td></td></tr></table> <p>La manera abreviada es: s:find(L) = 6, 8 Si el string s no pertenece al string ptr, entonces retorna nil</p> | D | e | m | o | | l | u | a | | s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | |
| D | e | m | o | | l | u | a | | s | | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | | | | | | | | |
| string.format(s, ...) | <p>Inserta valores al string s. El modo general de asignar los valores el por medio del string "%s".</p> <p>Ejemplo 1: string.format("\\be%s", l.shadow) Es decir que inserta el valor de l.shadow al string "\\be%s". Hay dos formas abreviadas de usar esta función en el Kara Effector:</p> <ul style="list-style-type: none">• format(s, ...)• F(s, ...) <p>Ejemplo 2: F("\\pos(%s,%s)", svl.center, 20*pi)</p> | | | | | | | | | | | | | | | | | | | | |

| | |
|--|---|
| | <p>Por cada vez que aparezca “%s” en el string, se debe asignar el valor que se insertará en esa posición. Este modo es el más usado en el Kara Effector y es la función string que más se usa en él. Los demás modos no son tan relevantes para hacer Efectos Karaoques.</p> |
| string.gsub(s, ptr, replace, n) | <p>Reemplaza al string s que pertenece al string ptr, por un valor u otro string llamado replace, n cantidad de veces.</p> <p>El modo abreviado de uso es: ptr:gsub(s, replace, n) n es un entero positivo, es opcional en la función.</p> <p>Ejemplo 1: ptr = “el sol en la mañana” ptr:gsub(“a”, “X”) = “el sol en lX mXñXnX” La función reemplazó al carácter “a” por “X” todas las veces.</p> <p>Ejemplo 2: ptr = “el sol en la mañana” ptr:gsub(“a”, “X”, 2) = “el sol en lX mXñana” El remplazo se hizo solo dos veces, dado que n = 2 Esta función también es mucho más extensa de lo que aparenta, pero de eso nos ocuparemos más adelante.</p> |
| string.len(s) | <p>Retorna la cantidad de caracteres que tiene el string s.</p> <p>El modo abreviado es: s:len()</p> <p>Ejemplo 1: s = “&HFFFFFF&” s:len() = 9</p> <p>Ejemplo 2: s = “El sol” s:len() = 6</p> <p>No olvidemos que el espacio entre palabra y palabra también es un caracter.</p> |
| string.lower(s) | <p>Retorna al string s con todas las letras mayúsculas en él convertidas en minúsculas.</p> <p>El modo abreviado es: s:lower()</p> <p>Ejemplo 1: s = “La Noche” s:lower() = “la noche”</p> |

| | |
|----------------------------|--|
| | <p>Ejemplo 2: s = “EL SOL” s:lower() = “el sol”</p> |
| string.rep(s, n) | <p>Repite al string s una n cantidad de veces.</p> <p>El modo abreviado es: s:rep(n)</p> <p>Ejemplo 1: s = “Demo” s:rep(3) = “DemoDemoDemo”</p> <p>Ejemplo 2: s = “lua ” s:rep(5) = “lua lua lua lua lua”</p> |
| string.reverse(s) | <p>Invierte al string s.</p> <p>El modo abreviado es: s:reverse()</p> <p>Ejemplo 1: s = “Demo” s:reverse() = “omeD”</p> <p>Ejemplo 2: s = “La Luna y el Sol” s:reverse() = “loS le y anuL aL”</p> |
| string.sub(s, i, j) | <p>Recorta al string s desde la posición i hasta la posición j.</p> <p>Tanto i como j son números enteros y pueden ser positivos o negativos. Al ser positivos la posición se empieza a contar de izquierda a derecha, si son negativos, se cuenta al revés.</p> <p>El modo abreviado es: s:sub(i, j)</p> <p>Ejemplos: s = “El Sol” s:sub(1, -1) = “El sol” s:sub(1, 1) = “E” s:sub(-3, -1) = “Sol” s:sub(2, -1) = “l Sol”</p> <p>El parámetro j es opcional s:sub(-2) = “o” s:sub(4) = “S”</p> |
| string.upper(s) | <p>Retorna al string s con todas las letras minúsculas en él convertidas en mayúsculas.</p> <p>El modo abreviado es: s:upper()</p> <p>Ejemplo 1: s = “La Noche” s:upper() = “LA NOCHE”</p> <p>Ejemplo 2: s = “¡¿Qué?, no puedes!” s:upper() = “¡¿QUÉ?, NO PUEDES!”</p> |

De esta Librería también omití un par de funciones que quizás, al necesitarlas más adelante, las explique a cada una de ellas, aunque al ser una biblioteca de **LUA** se hace un poco más simple hallar información en la web sobre ellas.

Terminada esta librería, es momento para ver las formas abreviadas de las funciones comúnmente usadas en el **Kara Effector**:

| ABREVIATURA | FUNCIÓN |
|-------------|---------------|
| pi | math.pi |
| sin | math.sin |
| cos | math.cos |
| tan | math.tan |
| asin | math.asin |
| acos | math.acos |
| atan | math.atan |
| sinh | math.sinh |
| cosh | math.cosh |
| tanh | math.tanh |
| log | math.log10 |
| ln | math.log |
| abs | math.abs |
| floor | math.floor |
| ceil | math.ceil |
| deg | math.deg |
| rad | math.rad |
| r | math.random |
| R | math.R |
| Rf | math.Rfake |
| rand | math.random |
| format | string.format |
| F | string.format |

Todas las anteriores abreviaturas de las funciones aplican tanto en lenguaje **LUA** como en **Automation Auto-4**.

A continuación veremos un poco de dos de las **teorías de color** que maneja el formato .ass que nos servirán para entender las siguientes Librerías.

TEORÍA DEL COLOR RGB (Red, Green, Blue):

Color RGB

Red: 11

Green: 87

Blue: 233

ASS: &HE9570B&

HTML: #0B57E9

De la anterior imagen vemos cómo el tono de azul que está en la parte superior de la misma, está formado por tres valores de rojo, verde y azul:

| Tono | Valor Decimal | Valor Hexadecimal |
|-------|---------------|-------------------|
| Rojo | 11 | 0B |
| Verde | 87 | 57 |
| Azul | 233 | E9 |

En formato .ass el tono de azul del ejemplo anterior sería:

&HE9570B&

De forma general, todo color en formato .ass tiene la siguiente estructura:

&H [Azul] [Verde] [Rojo] &

La estructura de un color en formato .ass es similar a la del formato **HTML**, pero los tonos invertidos, como se ve en la imagen anterior:


















































#0B57E9


















































El **Kara Effector** tiene una función que convierte un color de formato **HTML** a .ass:

color.ass("#0B57E9") = &HE9570B&

Veamos un listado de colores conocidos en formato **HTML** que con la anterior función ya sabemos cómo pasarlos a formato .ass. En la Tabla aparece el color, el nombre en inglés y el código del mismo en formato **HTML**:

| | | |
|--|----------------|----------|
| | AliceBlue | # F0F8FF |
| | AntiqueWhite | # FAEBD7 |
| | Aqua | # 00FFFF |
| | Aquamarine | # 7FFFD4 |
| | Azure | # F0FFFF |
| | Beige | # F5F5DC |
| | Bisque | # FFE4C4 |
| | Black | # 000000 |
| | BlanchedAlmond | # FFEBCD |
| | Blue | # 0000FF |
| | BlueViolet | # 8A2BE2 |
| | Brown | # A52A2A |
| | BurlyWood | # DEB887 |
| | CadetBlue | # 5F9EA0 |
| | Chartreuse | # 7FFF00 |
| | Chocolate | # D2691E |
| | Coral | # FF7F50 |
| | CornflowerBlue | # 6495ED |

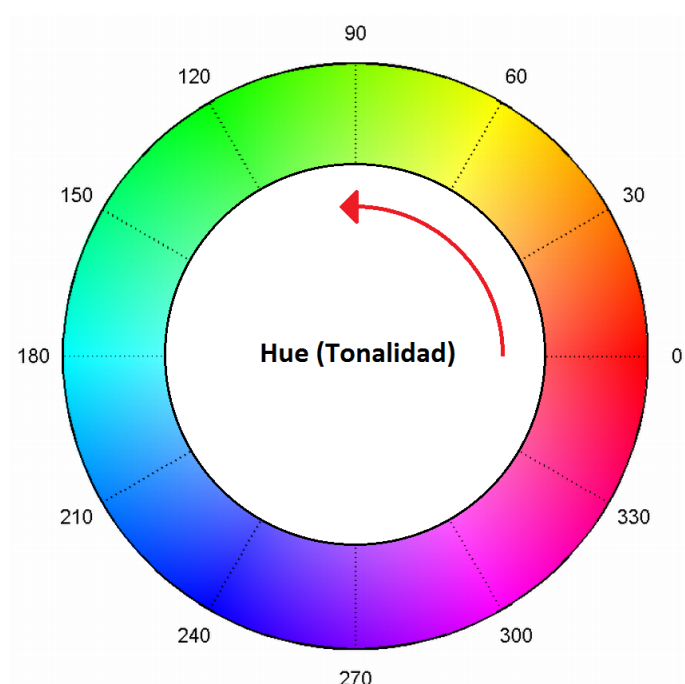
| | | |
|--|------------------------|---------|
|  | Corn silk | #FFF8DC |
|  | Crimson | #DC143C |
|  | Cyan | #00FFFF |
|  | Dark blue | #00008B |
|  | Dark cyan | #008B8B |
|  | Dark goldenrod | #B8860B |
|  | Dark gray | #A9A9A9 |
|  | Dark green | #006400 |
|  | Dark khaki | #BDB76B |
|  | Dark magenta | #8B008B |
|  | Dark olive green | #556B2F |
|  | Dark orange | #FF8C00 |
|  | Dark orchid | #9932CC |
|  | Dark red | #8B0000 |
|  | Dark salmon | #E9967A |
|  | Dark sea green | #8FBC8F |
|  | Dark slate blue | #483D8B |
|  | Dark slate gray | #2F4F4F |
|  | Dark turquoise | #00CED1 |
|  | Dark violet | #9400D3 |
|  | Deep pink | #FF1493 |
|  | Deep sky blue | #00BFFF |
|  | Dim gray | #696969 |
|  | Dodger blue | #1E90FF |
|  | Firebrick | #B22222 |
|  | Floral white | #FFFAF0 |
|  | Forest green | #228B22 |
|  | Fuchsia | #FF00FF |
|  | Gainsboro | #DCDCDC |
|  | Ghost white | #F8F8FF |
|  | Gold | #FFD700 |
|  | Goldenrod | #DAA520 |
|  | Gray | #808080 |
|  | Green | #008000 |
|  | Green yellow | #ADFF2F |
|  | Honeydew | #F0FFF0 |
|  | Hot pink | #FF69B4 |
|  | Indian red | #CD5C5C |
|  | Indigo | #4B0082 |
|  | Ivory | #FFFFF0 |
|  | Khaki | #F0E68C |
|  | Lavender | #E6E6FA |
|  | Lavender blush | #FFF0F5 |
|  | Lawn green | #7CFC00 |
|  | Lemon chiffon | #FFFACD |
|  | Light blue | #ADD8E6 |
|  | Light coral | #F08080 |
|  | Light cyan | #E0FFFF |
|  | Light goldenrod yellow | #FAFAD2 |

| | | |
|---|---------------------|---------|
|  | Light gray | #D3D3D3 |
|  | Light green | #90EE90 |
|  | Light pink | #FFB6C1 |
|  | Light salmon | #FFA07A |
|  | Light sea green | #20B2AA |
|  | Light sky blue | #87CEFA |
|  | Light slate gray | #778899 |
|  | Light steel blue | #B0C4DE |
|  | Light yellow | #FFFFE0 |
|  | Lime | #00FF00 |
|  | Lime green | #32CD32 |
|  | Linen | #FAF0E6 |
|  | Magenta | #FF00FF |
|  | Maroon | #800000 |
|  | Medium aquamarine | #66CDAA |
|  | Medium blue | #0000CD |
|  | Medium orchid | #BA55D3 |
|  | Medium purple | #9370DB |
|  | Medium sea green | #3CB371 |
|  | Medium slate blue | #7B68EE |
|  | Medium spring green | #00FA9A |
|  | Medium turquoise | #48D1CC |
|  | Medium violet red | #C71585 |
|  | Midnight blue | #191970 |
|  | Mint cream | #F5FFFA |
|  | Misty rose | #FFE4E1 |
|  | Moccasin | #FFE4B5 |
|  | Navajo white | #FFDEAD |
|  | Navy | #000080 |
|  | Old lace | #FDF5E6 |
|  | Olive | #808000 |
|  | Olive drab | #6B8E23 |
|  | Orange | #FFA500 |
|  | Orange red | #FF4500 |
|  | Orchid | #DA70D6 |
|  | Pale goldenrod | #EEE8AA |
|  | Pale green | #98FB98 |
|  | Pale turquoise | #AFEEEE |
|  | Pale violet red | #DB7093 |
|  | Papaya whip | #FFEFD5 |
|  | Peach puff | #FFDAB9 |
|  | Peru | #CD853F |
|  | Pink | #FFC0CB |
|  | Plum | #DDA0DD |
|  | Powder blue | #B0E0E6 |
|  | Purple | #800080 |
|  | Red | #FF0000 |
|  | Rosy brown | #BC8F8F |
|  | Royal blue | #4169E1 |

| | |
|-------------|---------|
| SaddleBrown | #8B4513 |
| Salmon | #FA8072 |
| SandyBrown | #F4A460 |
| SeaGreen | #2E8B57 |
| SeaShell | #FFF5EE |
| Sienna | #A0522D |
| Silver | #C0C0C0 |
| SkyBlue | #87CEEB |
| SlateBlue | #6A5ACD |
| SlateGray | #708090 |
| Snow | #FFFAFA |
| SpringGreen | #00FF7F |
| SteelBlue | #4682B4 |
| Tan | #D2B48C |
| Teal | #008080 |
| Thistle | #D8BFD8 |
| Tomato | #FF6347 |
| Transparent | #FFFFFF |
| Turquoise | #40E0D0 |
| Violet | #EE82EE |
| Wheat | #F5DEB3 |
| White | #FFFFFF |
| WhiteSmoke | #F5F5F5 |
| Yellow | #FFFF00 |
| YellowGreen | #9ACD32 |

TEORÍA DEL COLOR HSV (Hue, Saturation, Value):

Hue: es la tonalidad de un color, cuyo valor es un real desde 0 hasta 360, como vemos en la imagen:



En la anterior imagen vemos cómo **Hue** = 0 equivale al rojo, **Hue** = 60 equivale al amarillo, **Hue** = 240 equivale al azul.

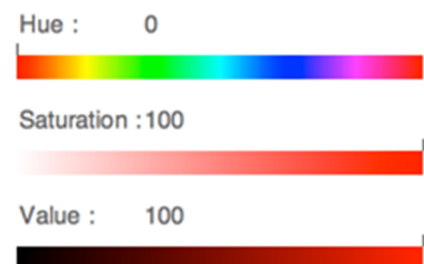
Veamos un ejemplo con el color Rojo. Con la teoría del color **RGB** sería (los valores de R, G y B son reales entre 0 como mínimo y 255 como máximo):



En formato HTML: **#FF0000**

En formato .ass: **&H0000FF&**

Pero en la teoría HSV sería:



- Hue = 0
- Saturation = 100
- Value = 100

Lo que da pie a las siguientes dos definiciones que hacen parte a la teoría HSV.

Saturation: es la cantidad de blanco que tendrá la tonalidad **Hue**. Su valor es un número real entre 0 y 100, donde 0 completamente blanco y 100 equivale a que la tonalidad no tendría nada de blanco.

Value: es la cantidad de negro que tendrá la tonalidad **Hue**. Su valor es un número real entre 0 y 100, donde 0 completamente negro y 100 equivale a que la tonalidad no tendría nada de negro.

Si **Saturation** es 0, no importa el valor de **Hue**, el color siempre será **Blanco**. De manera similar pasaría con **Value**, ya que si es 0, no importaría el valor de **Hue**, el color que resultaría siempre sería **Negro**.

Para poner en práctica la teoría del color **HSV**, **Aegisub** ya trae por default un par de funciones que hacen posible convertir los tres valores (H, S y V) en un color en formato .ass, y la primera de ellas es:

HSV_to_RGB(H, S, V): convierte los valores de **H**, **S** y **V** en valores de 0 a 255 de Rojo, Verde y Azul. Es decir que esta función retorna tres resultados al mismo tiempo.

Para esta función de Aegisub, **Hue** sigue siendo un real entre 0 y 360, pero **Saturation** y **Value** son un número real entre 0 y 1.

Ejemplo 1:

R, G, B = **HSV_to_RGB(0, 1, 1)**

- R = 255
- G = 0
- B = 0

Ejemplo 2:

R, G, B = **HSV_to_RGB(264, 0.75, 0.2)**

- R = 28.05
- G = 12.75
- B = 51

Y la siguiente función del **Aegisub** es la que convierte estos tres valores en un color en formato .ass para que pueda ser usado en los tags de colores:

ass_color(R, G, B): transforma los valores de R, G y B en un color en formato .ass: **&H[Azul][Verde][Rojo]&**

Ejemplos:

ass_color(255, 0, 0) = &H0000FF&

ass_color(47, 82, 242) = &HF2522F&

ass_color(158, 80, 153) = &H99509E&

lo que quiere decir que debemos combinar estas dos funciones para convertir los valores de H, S y V en un color en formato .ass:

ass_color(HSV_to_RGB(H, S, V))

veamos algunos ejemplos de esta combinación:

| Estilo | Efecto | Texto |
|---------|----------|---|
| Default | template | !_G.ass_color(_G.HSV_to_RGB(45, 1, 0.5))! |
| Default | karaoke | |
| Default | fx | &H005F7F& |
| Default | fx | &H005F7F& |

Recordemos que desde el **Aegisub**, para usar cualquiera de las funciones que por default que vienen en él, éstas deben iniciar por “_G.” como se ve en la imagen anterior. En el **Kara Effector** da igual si se coloca este prefijo o no, las funciones son reconocidas de las dos formas.

Un ejemplo práctico en el **Kara Effector** sería:

Add Tags: Add Tags Language: Lua

```
"\1c" .. ass_color(HSV_to_RGB(R(360), 1, 0.5))
```

Add Tags: Add Tags Language: Automation Auto-4

```
\1class_color(HSV_to_RGB(R(360), 1, 0.5))!
```

Uno más usando la función **string.format**:

Add Tags: Add Tags Language: Lua

```
format("\1c%s\1bord%s", ass_color(HSV_to_RGB(45, 1, 1)), r(2, 5))
```

Es decir que con esta combinación de funciones podemos obtener un color con valores constantes, como en el ejemplo de la imagen anterior, o con valores aleatorios (random), como en las otras dos anteriores imágenes.

En este punto, el nivel de complejidad ha ido en aumento y cada vez tenemos un mayor conocimiento y contamos con más herramientas para desarrollar nuestros propios Efectos. La mayoría de las próximas funciones de las siguientes librerías tienen aplicación directamente para hacer nuevos Efectos y vendrán acompañadas con varios ejemplos y ejercicios prácticos. No olviden 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.