Librería “table” [KE]

Es la ampliación de la Librería “**table**” con que cuenta el **Kara Effector** y algunas de sus funciones nos sirven para hacer Efectos y las restantes para crear nuevas funciones.

table.inside(t, e)	Retorna **true** (verdadero) en el caso que el elemento “**e**” esté en la tabla “**t**”. En caso contrario retorna **false** (falso). Ejemplo: P = {“a”, “b”, “c”} **table.inside**(P, “c”) = **true** **table.inside**(P, “e”) = **false**
table.index(t, e)	Retorna el **índice** del elemento “**e**” en el caso de que dicho elemento pertenezca a la tabla “**t**”. En caso contrario retorna de nuevo al elemento. Ejemplo: R = {13, 42, 63, 34, 25} **table.index**(R, 42) = 2 Retorna 2, porque R[2] = 42 **table.index**(R, “AA”) = “AA” Retorna “AA”, porque: **table.inside**(R, “AA”) = **false** O sea que “AA” no está en R
table.compare (t1, t2)	Retorna **true** (verdadero) en el caso de que la tabla **t1** sea igual que la tabla **t2**, de otro modo retorna **false** (falso). Ejemplo: A = {1, 2, 3} B = {1, 2, 3, 4} **table.compare**(A, B) = **false** C = {7, “a”} D = {7, “a”} **table.compare**(C, D) = **true**

	t puede ser una tabla o un número entero positivo mayor o igual que 2. Para el caso en que **t** sea una tabla, retorna a la misma tabla con sus elementos en desorden. Los elementos se desordenan de forma aleatoria (random). Ej: H = {“a”, “b”, “c”, 3, 9} G = **table.disorder**(H) Un posible resultado sería:
table.disorder(t)	G = {“b” , 9, “a”, 3, “c”} Si **t** es un entero positivo, entonces la función crea una tabla de números consecutivos desde 1 hasta**t**, para luego desordenar esos números. Ej: G = **table.disorder**(6) Un posible resultado sería: G = {3, 5, 6, 2, 1, 4} El número total de resultados es**#t!**, o sea el factorial del tamaño de la tabla. Para este último ejemplo sería: 6! = 720 posibilidades

He decidido hacer un pequeño paréntesis acá para aclarar algunos conceptos que nos ayudarán a entender mucho de lo que se viene. El lenguaje **Automation Auto-4** es una modificación del lenguaje **LUA** para que reconozca las variables **Dólar** (**$**) y las operaciones hechas dentro de los signos de admiración (**!!**). En pocas palabras, el lenguaje **Automation Auto-4** está basado en el lenguaje **LUA**.

Otro de los conceptos que quería mencionar es uno que en ocasiones nos encontramos en los parámetros que requiere una función, y son tres puntos seguidos (…). Los tres puntos seguidos pueden ser una variable, una tabla, un elemento o un listado de elementos y/o una combinación de éstos. Estos tres puntos se colocan para indicar que podemos poner cuantas cosas queramos.

table.concat1(t, …)

Retorna la tabla **t** con todos sus

Elementos concatenados a (…).

Ejemplo 1: (tabla y un elemento)

A = {“a”, “b”, “c”}

B = **table.concat1**(A, 9)

B = {“9a”, “9b”, “9c”}

Ejemplo 2: (tabla y más de un elemento)

F = {1, 2, 3}

G = **table.concat1**(F, a, b)

G = {a1, b1, a2, b2, c1, c2}

Ejemplo 3: (tabla con tabla)

M = {4, 6, 8}

N = {f, g}

O = **table.concat1**(M, N)

O = {f4, g4, f6, g6, f8, g8}

En estos ejemplos vemos cómo los tres puntos pueden ser un solo elemento o varios, también pueden ser una tabla

table.concat2(t, …)

Es similar a **table.concat1** y la diferencia se notará en los ejemplos.

Ejemplo 1: (tabla y un elemento)

A = {“a”, “b”, “c”}

B = **table.concat1**(A, 9)

B = {“9a”, “9b”, “9c”}

Ejemplo 2: (tabla y más de un elemento)

F = {1, 2, 3}

G = **table.concat1**(F, a, b)

G = {a1b1, a2b2, c1c2}

Ejemplo 3: (tabla con tabla)

M = {4, 6, 8}

N = {f, g, h}

O = **table.concat1**(M, N)

O = {f4g4h4, f6g6h6, f8g8h8}

table.replay(n, …)

Retorna una tabla con **n** veces repetidas a (…).

Ejemplo 1:

A = **table.replay**(4, "a")

A = {"a", "a", "a", "a"}

Ejemplo 2:

B = **table.replay**(3, f, g, h)

B = {f, g, h, f, g, h, f, g, h}

Ejemplo 3:

C = {7, 8, 9}

D = **table.replay**(2, C)

D = {7, 8, 9, 7, 8, 9}

Se nota cómo se repiten **n** veces el o los elementos ingresados

table.count(t, e)

Retorna el número de veces en el que el elemento **e** aparece en la tabla **t**. en el caso en que el elemento **e** no esté en **t**, retorna 0. Ejemplo:

A = {a, b, a, 7, a, 8, 9, a}

table.count(A, a) = 4

table.count(A, c) = 0

table.count(A, b) = 1

table.pos(t, e)

Retorna una tabla con el o los**índices** del elemento **e** en la tabla**t**. En el caso de que el elemento **e**no esté en **t**, retorna una tabla vacía. Ejemplo:

A = {a, b, a, 7, a, 8, 9, a}

B = **table.count**(A, a)

B = {1, 3, 5, 8}

C = **table.count**(A, c)

C = { }

D = **table.count**(A, b)

D = {2}

table.retire(t, …)

Retorna la tabla **t**, pero retira a (…) de la tabla en el caso en que están en ella.

Ejemplo 1:

A = {a, b, a, 7, a, 8, 9, a}

B = **table.retire**(A, a)

B = {b, 7, 8, 9}

Ejemplo 2:

C = **table.retire**(A, a, 7, 8)

C = {b, 9}

Ejemplo 3:

D = {7, 8, 9}

E = **table.retire**(A, D)

E = {a, b, a, a, a}

Inserta los elementos de la tabla **t2**en la tabla **t1** a partir del **índice i**, o en el caso de no estar el parámetro **i**, se insertan al final de la tabla **t1**.

Ejemplo 1:

A = {6, 7, 8}

B = {a, b, c, d}

C = **table.inserttable**(A, B, 3)

table.inserttable (t1, t2, i)

C = {6, 7, a, b, c, d, 9}

Los elementos de la tabla B se insertaron en la tabla A, a partir del **índice** 3 (la tercera posición).

Ejemplo 2:

D = {5, 4, 1} E = {f, g}

F = **table.inserttable**(D, E)

F = {5, 4, 1, f, g}

table.reverse(t)

Retorna a la tabla t, pero con el**índice** invertido.

Ejemplo 1:

A = {3, 4, 5, 6, 7}

B = **table.reverse**(A)

B = {7, 6, 5, 4, 3}

Ejemplo 2:

C = {f, 5, 3, a, m, 2, 9}

D = **table.reverse**(C)

D = {9, 2, m, a, 3, 5, f}

table.cyclic(t)

Genera un "**ciclo**" con todos los elementos de la tabla t.

Ejemplo 1:

A = {2, 4, 6, 8}

B = **table.cyclic**(A)

B = {2, 4, 6, 8, 6, 4, 2}

Ejemplo 2:

C = {a, 4, 2, d, 5, b}

D = **table.cyclic**(C)

D = {a, 4, 2, d, 5, b, 5, d, 2, 4, a}

table.op(t, mode)

Genera operaciones con los elementos de la tabla **t**. Dichas operaciones dependen del modo "**mode**". Esta función es exclusiva para las tablas con elementos numéricos.

Ejemplo 1: **mode** = "sum"

Suma los elementos de la tabla

A = {9, 1, 16, 4}

table.op(A, "sum")

= 9 + 1 + 16 + 4

= 30

Ejemplo 2: **mode** = "concat"

Une a los elementos de la tabla

table.op(A, "concat") = 14916

Ejemplo 3: **mode** = "average"

Obtiene un promedio de la tabla

table.op(A, "average")

= (9 + 1 + 16 + 4) / #A

= 30 / 4 = 7.5

Ejemplo 4: **mode** = “min”

Da al menor de los elementos

table.op(A, “min”) = 1

Ejemplo 5: **mode** = “max”

Da al mayor de los elementos

table.op(A, “max”) = 16

Ejemplo 6: **mode** = “add”

Adiciona un tercer parámetro a
cada uno de los elementos

B = **table.op**(A, “add” -2)

B = {9 – 2, 1 – 2, 16 – 2, 4 – 2}

B = {7, -1, 14, 2}

---

table.make(Objet, Size, limit_i, limit_f, …): esta función crea una **tabla** de un tamaño determinado “**Size**” y que contiene elementos equidistantes entre sí, del tipo “**Objet**”.

El parámetro **Objet** indica el tipo de elementos que tendrá la **tabla** retornada, y tiene cuatro opciones:

- “**number**”
- “**color**”
- “**alpha**”
- **Tags string**

Cada uno de ellos crea elementos distintos en nuestra **tabla** y más adelante veremos ejemplos de cada.

El parámetro **Size** indica el tamaño de la **tabla**, es decir, la cantidad de elementos que contendrá. **Size** debe ser un número entero mayor a cero.

Los parámetros **limit_i** y **limit_f** indican los límites inferior y superior que tendrán los elementos de la **tabla** creada. El primer elemento de la tabla sería **limit_i** y el último vendría a ser **limit_f**, y en medio de ellos todos los elementos equidistantes dependiendo del parámetro Size.

Los tres puntos seguidos (**…**) hacen referencia a uno o más tags que queramos añadirle a cada uno de los elementos que harán parte de la **tabla** retornada. Este parámetro es opcional.

- **Ejemplo 1. Objet** = "**number**":

mi_tabla = **table.make("number", 8, 20, 55)**

mi_tabla = {

[1] = 20, **limit_i**

[2] = 25,

[3] = 30,

[4] = 35,

[5] = 40,

[6] = 45,

[7] = 50,

[8] = 55 **limit_f**

}

Del anterior ejemplo vemos que la anterior **tabla** contiene 8 elementos (**#mi_tabla** = 8), y cada uno de ellos es un número equidistante entre 20 (**limit_i**) y 55 (**limit_f**).

- **Ejemplo 2. Añadir un tag**:

mi_tabla = **table.make("number", 8, 20, 55, "\\fr")**

mi_tabla = {

[1] = **fr**20, **limit_i**

[2] = **fr**25,

[3] = **fr**30,

[4] = **fr**35,

[5] = **fr**40,

[6] = **fr**45,

[7] = **fr**50,

[8] = **fr**55 **limit_f**

}

Ahora en este ejemplo, como en el quinto parametro colocamos "**fr**", entonces la función añade este tag al inicio de cada elemento de la **tabla**.

- **Ejemplo 3. Añadir más de un tag**:

Para añadir dos o más tags a los elementos hay dos diferentes métodos. **Método 1**:

mi_tabla = **table.make("number", 8, 20, 55, "\\fr","\\b")**

mi_tabla = {

[1] = **fr**20 **b**20, **limit_i**

[2] = **fr**25 **b**25,

[3] = **fr**30 **b**30,

[4] = **fr**35 **b**35,

[5] = **fr**40 **b**40,

[6] = **fr**45 **b**45,

[7] = **fr**50 **b**50,

[8] = **fr**55 **b**55 **limit_f**

}

Cada elemento de la **tabla** se multiplica para corresponder a cada uno de los tags ingresados en el quinto parámetro de la función ("**fr**" y "**b**").

- **Ejemplo 4. Añadir más de un tag**:

Método 2:

Tags = {"**frx**", "**fry**", "**frz**", "**fscy**"}

mi_tabla = **table.make("number", 8, 20, 55, Tags)**

mi_tabla = {

[1] = **frx**20 **fry**20 **frz**20 **fscy**20,

[2] = **frx**25 **fry**25 **frz**25 **fscy**25,

[3] = **frx**30 **fry**30 **frz**30 **fscy**30,

[4] = **frx**35 **fry**35 **frz**35 **fscy**35,

[5] = **frx**40 **fry**40 **frz**40 **fscy**40,

[6] = **frx**45 **fry**45 **frz**45 **fscy**45,

[7] = **frx**50 **fry**50 **frz**50 **fscy**50,

[8] = **frx**55 **fry**55 **frz**55 **fscy**55

}

En resumen, cuando **Objet** = "**number**", los parámetros **limit_i** y **limit_f** deben ser números también, de modo que la función cree a cada uno de sus elementos de forma equidistante entre esos dos valores.

Para los siguientes ejemplos, usaremos la siguiente opción del parámetro **Objet**: "**color**"

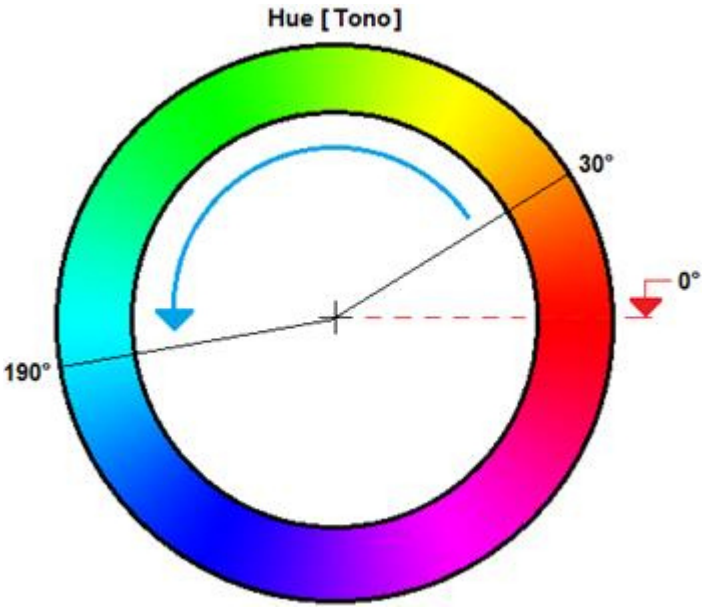
- **Ejemplo 5. Objet** = "**color**":

Límites Numéricos:

mi_tabla = **table.make(“color”, 15, 30, 190)**

- **Objet** = “**color**”
- **Size** = 15 (tamaño de la **tabla**)
- **limit_i** = 30 (límite inferior)
- **limit_f** = 190 (límite superior)

Los valores 30 y 190 hacen referencia a un ángulo entre 0° y 360° del Círculo Cromático de la **Teoría del Color HSV** (**Hue**, **Saturation**, **Value**):



Entonces la función creará una **tabla** de 15 colores entre los 30° y los 190°, equidistantes entre sí:

mi_tabla = {

[1] = “&H007FFF&”,

[2] = “&H00B0FF&”,

[3] = “&H00E0FF&”,

[4] = “&H00FFEC&”,

[5] = “&H00FFBC&”,

[6] = “&H00FF8B&”,

[7] = “&H00FF5B&”,

[8] = “&H00FF2A&”,

[9] = “&H06FF00&”,

[10] = “&H36FF00&”,

[11] = “&H67FF00&”,

[12] = “&H97FF00&”,

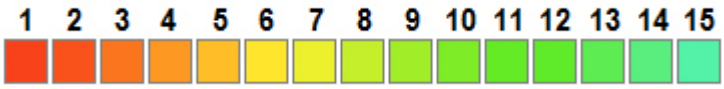
[13] = “&HC8FF00&”,

[14] = “&HF8FF00&”,

[15] = “&HFFD400&”

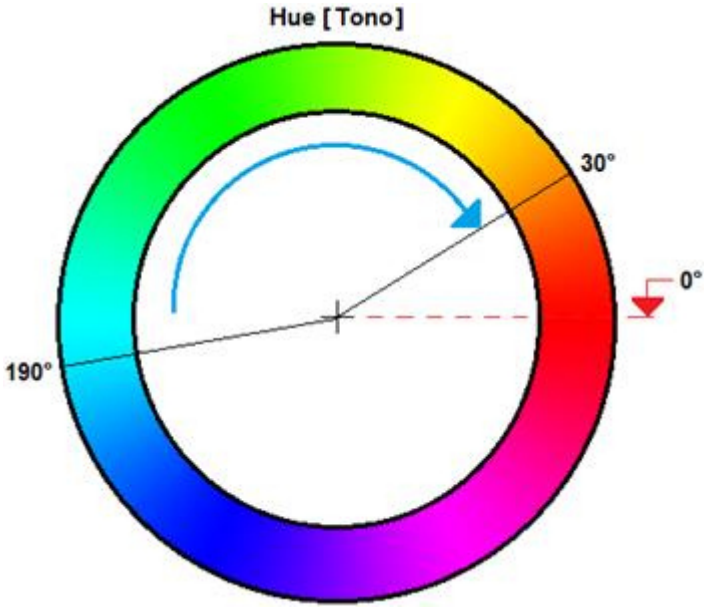
}

Se entiende un poco más si se ven los tonos de los 15 colores de la **tabla** generada. Noten que equivalen a los colores entre 30° y 190°:



No necesariamente el valor de **limit_i** debe ser menor que el de **limit_f**. Ejemplo:

mi_tabla = **table.make(“color”, 20, 190, 30)**

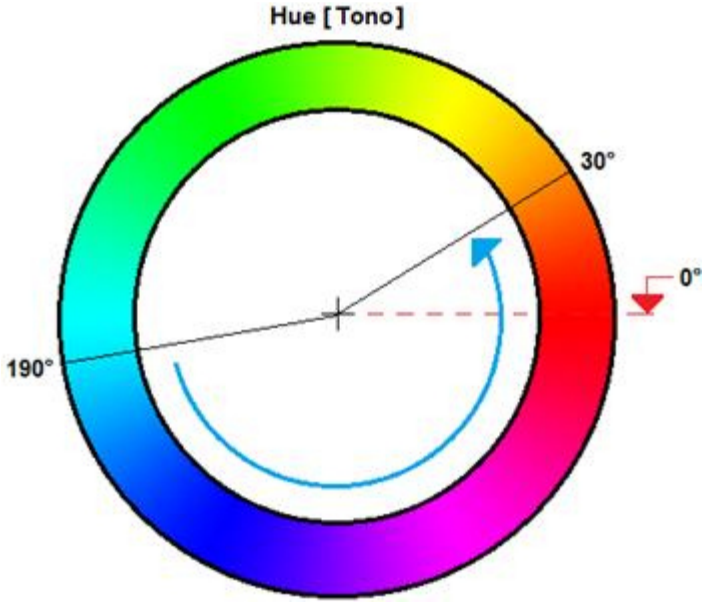


Entonces, lo que hace la función es crear una **tabla** de 20 colores desde los 190°, hacia atrás, hasta los 30°. La función hace el recorrido de las tonalidades en el Círculo Cromático en sentido horario.

Si alguno de los límites de la función excede a los 360°, dicho valor dará la vuelta en el Círculo, literalmente. Por cada giro se restan 360°.

Ejemplo:

mi_tabla = **table.make(“color”, 10, 190, 390)**



La función tomará a 10 colores desde los 190° hasta los 30° en sentido anti horario:

30° = 390° – 360°

Para agregar tags a los colores de la tabla, hacemos como en los primeros ejemplos:

mi_tabla = **table.make(“color”, 7, 60, 150, “\\1c”)**

mi_tabla = {

[1] = **\1c**&H00FFFF&,

[2] = **\1c**&H00FFBF&,

[3] = **\1c**&H00FF7F&,

[4] = **\1c**&H00FF3F&,

[5] = **\1c**&H00FF00&,

[6] = **\1c**&H3FFF00&,

[7] = **\1c**&H7FFF00&

}

- **Ejemplo 6.** Cuando **limit_i** y **limit_f** son dos **strings** de colores:

tbl = **table.make(“color”,6, “&HFFFFFF&”, “&H000000&”)**

tbl = {

[1] = “&HFFFFFF&”,

[2] = “&HCCCCCC&”,

[3] = “&H999999&”,

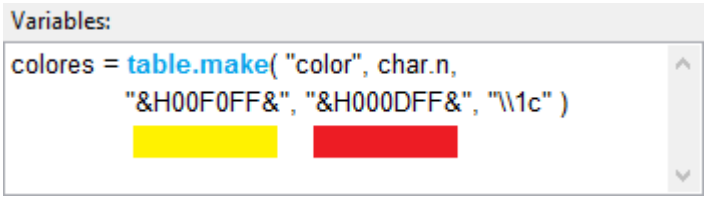
[4] = “&H666666&”,

[5] = “&H333333&”,

[6] = “&H000000&”

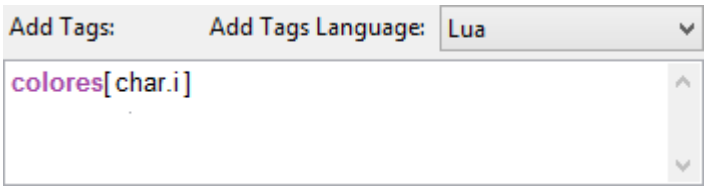
}

Hagamos un ejemplo práctico en el **Kara Effector**, para el cuál usaremos un **Template Type: Char** y la plantilla **ABC Template leadin**:



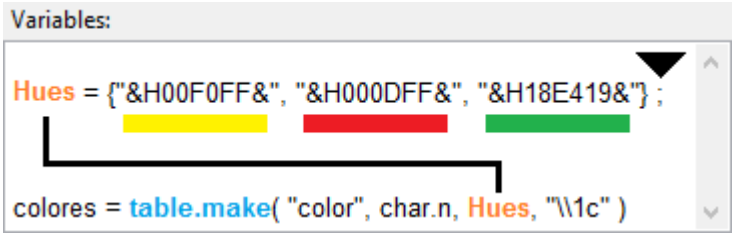
Como podemos ver, el tamaño de la **tabla** es **char.n**, que es el número total de caracteres (letras, números, signos) de la línea karaoke.

Para asignar los colores a cada carácter, hacemos:



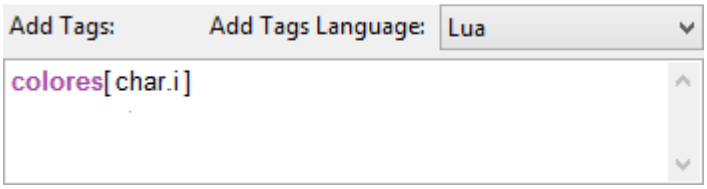
De esta manera, asignamos al primer carácter, el primer color; al segundo le asignamos el segundo color y así con todos los caracteres de la línea karaoke:

- **Ejemplo 7.** Cuando **limit_i** y **limit_f** se “fusionan” es un solo parámetro para ingresar más de dos colores. Este nuevo parámetro será ingresado en forma de**tabla**:

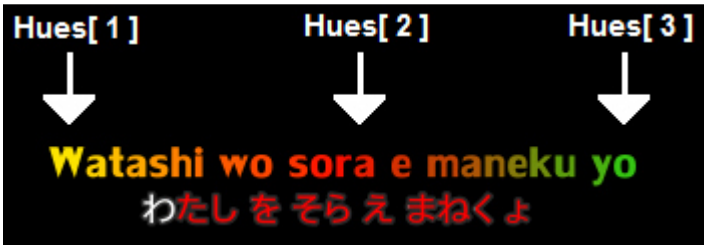


En "**Variables**" declaramos una **tabla** (**Hues**, en la imagen anterior) con las tonalidades de referencia (amarillo, rojo y verde, en este caso) para que la función **table.make** genere los colores equidistantes entre ellas. Nótese que resalto el punto y coma (;) con el cual se deben separa las variables.

Obtenemos los colores de la **tabla** "**colores**" de la misma forma que en el ejemplo anterior:



Y vemos los resultados:



Con el método usado en el anterior ejemplo, se pueden hacer **Gradientes** (degradaciones) de tres o más colores. Todo depende de los resultados deseados y del efecto que queremos hacer. Les recomiendo que practiquen con otros tres colores distintos y luego usando más colores.

- **Ejemplo 8. Objet** = "**alpha**" y límites numéricos.

Los límites numéricos ya no están entre 0 y 360, como en el caso de **Objet** = "**color**"; en este caso los valores de los límites van desde **0** a **255**:

mi_tabla = **table.make("alpha", 10, 45, 86)**

mi_tabla = {

[1] = "&H2D&", 45 en Hexadecimal

[2] = "&H31&",

[3] = "&H36&",

[4] = "&H3A&",

[5] = "&H3F&",

[6] = "&H43&",

[7] = "&H48&",

[8] = “&H4C&”,

[9] = “&H51&”,

[10] = “&H56&” 86 en Hexadecimal

}

Para asignarle los tags, hacemos lo mismo que en los ejemplos de colores. Ejemplo:

mi_tabla = **table.make(“alpha”,8, 50, 200, “\\1a”, “\\3a”)**

mi_tabla = {

[1] = “\\1a&H32& \\3a&H32&”,

[2] = “\\1a&H47& \\3a&H47&”,

[3] = “\\1a&H5C& \\3a&H5C&”,

[4] = “\\1a&H72& \\3a&H72&”,

[5] = “\\1a&H87& \\3a&H87&”,

[6] = “\\1a&H9D& \\3a&H9D&”,

[7] = “\\1a&HB2& \\3a&HB2&”,

[8] = “\\1a&HC8& \\3a&HC8&”

}

- **Ejemplo 9**. Usando los límites como **strings alpha**:

mi_tabla = **table.make(“alpha”, 6, “&H4E&”, “&HAD&”)**

mi_tabla = {

[1] = “&H4E&”, **“&H4E&”**

[2] = “&H61&”,

[3] = “&H74&”,

[4] = “&H87&”,

[5] = “&H9A&”,

[6] = “&HAD&” **“&HAD&”**

}

- **Ejemplo 10**. “fusionar” los parámetros **limit_i** y **limit_f** para poder ingresar una tabla con tres o más **strings alpha**:

Alphas = {“&HFF”, “&HA2&”, “&H16&”, “&HDE&”};

mi_tabla = **table.make(“alpha”, 12, Alphas)**

- **Ejemplo 11. Objet = Tags strings:**

mi_tabla = **table.make(“\\blur”, 5)**

mi_tabla = {

[1] = “\\blur5”,

[2] = “\\blur5”,

[3] = “\\blur5”,

[4] = “\\blur5”,

[5] = “\\blur5”

}

Es decir que el valor de Size (5 para el anterior ejemplo), no solo indica el tamaño de la**tabla**, sino que también se concatena (se une) con el tag ingresado (“**blur**”).

- **Ejemplo 12:**

mi_tabla = **table.make(“\\fscx”, 6, 80, 120)**

mi_tabla = {

[1] = “\\fscx80″, <i class=”fa fa-long-arrow-left”></i> 80

[2] = “\\fscx88”,

[3] = “\\fscx96”,

[4] = “\\fscx104”,

[5] = “\\fscx112”,

[6] = “\\fscx120″<i class=”fa fa-long-arrow-left”></i> 120

}

Los valores numéricos equidistantes entre **limit_i** y **limit_f** (80 y 120) se concatenan al tag “**fscx**”.

table.rmake(Objet, Size, limit_i, limit_f, …): esta función es similar a**table.make**, pero con la diferencia que los elementos de la **tabla** ya no están ni organizados ni equidistantes entre sí, sino que crea los elementos de forma totalmente aleatoria, teniendo en cuenta los limites inferior y superior de los parámetros **limit_i** y **limit_f**.

table.gradient(color1, color2, algorithm): crea una tabla de colores (o de alphas), correspondientes a un Gradiente entre los parámetros **color1** y **color2**. El tamaño de la **tabla** generada dependerá del **Template Type**, de tal manera que a cada objeto karaoke le corresponda un único elemento. Si por ejemplo el **Template Type** es**Translation Word**, entonces el tamaño de la **tabla** que se generará será **word.n**, de modo que por cada palabra de la línea karaoke, haya un elemento en la **tabla** que le corresponde.

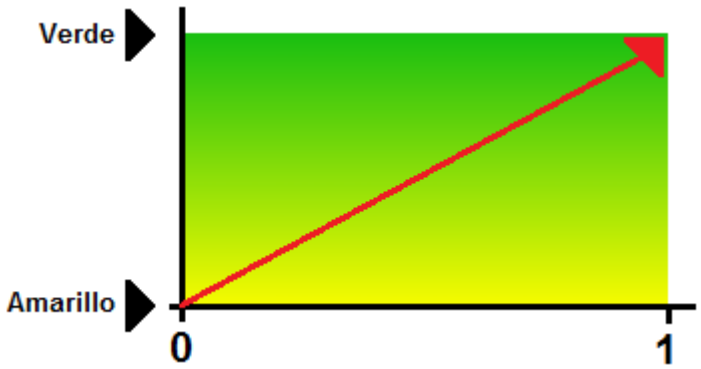
El parámetro **algorithm** es opcional y determina el modo de transición desde **color1** hasta**color2**.

- **Ejemplo 1. Template Type: Word**

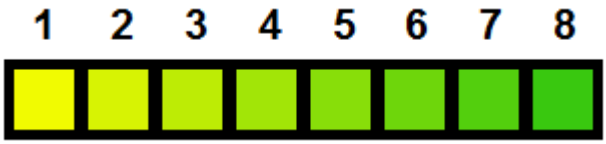
word.n = 8 (suponiendo que una línea tiene 8 palabras. Este número es solo para saber cómo se determina el tamaño de la **tabla**)

mi_tabla =**table.gradient("&H00FFFF&", "&H12BE12&")**

Como hemos omitido al tercer parámetro de la función (**algorithm**), entonces la transición entre el amarillo y el verde se hará de forma lineal:



Los 8 elementos de la tabla generada serán:



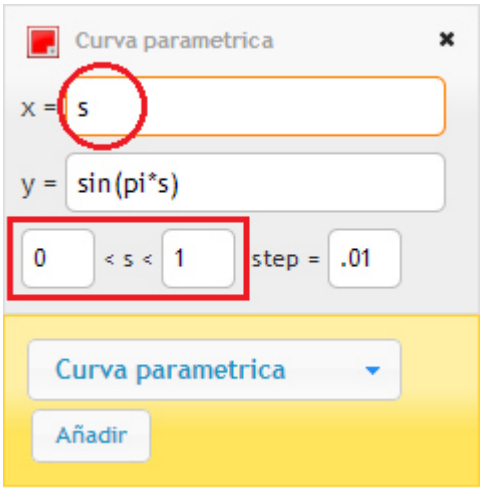
De este modo, la degradación se hace normalmente, ya que omitimos un algoritmo que modifique la transición entre los dos colores ingresados.

- **Ejemplo 2. Template Type: Syl**

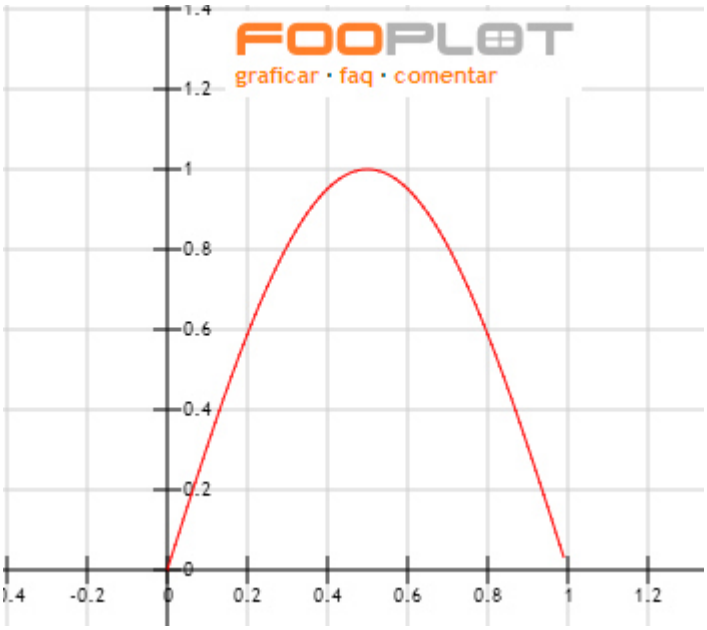
syl.n = 12

Uno de los métodos simples para crear un **algoritmo** para esta función, es usar un graficador de funciones online; el que generalmente usamos es "**fooplot**" en la opción de **Curva Paramétrica**. Los pasos son:

- En [**x** =] ponemos la letra (s)
- El dominio se pone desde 0 a 1



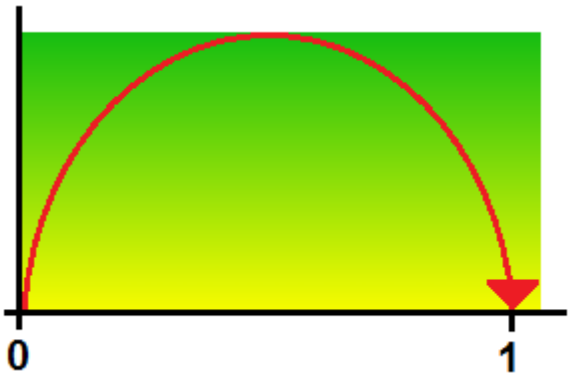
- En [**y** =] declaramos el algoritmo en función de la letra (s), como en la imagen anterior, y en la gráfica veremos esto:



Hecho esto, copiamos el algoritmo hecho en [y] y lo pegamos en el tercer parámetro de la función, entre comillas simples o dobles, y añadiendo el signo de porcentaje (%) antes de cada letra (s) que haya en el algoritmo:

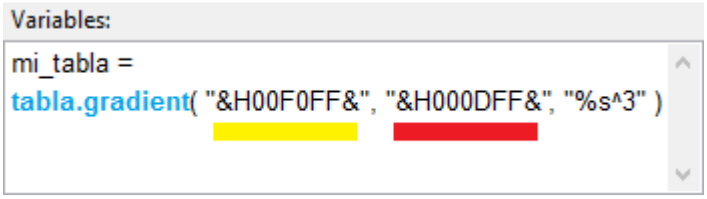
sin(pi*s) “sin(pi*%s)”

mi_tabla = **table.gradient(“&H00FFFF&”, “&H12BE12&”, “sin(pi*%s)”)**



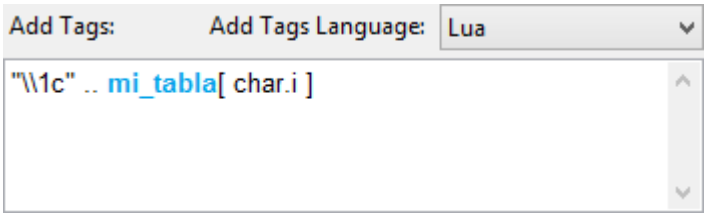
Entonces se hará un Gradiente desde el amarillo hasta el verde y luego regresará al amarillo:

- **Ejemplo 3:**



Vemos la gráfica de (s^3) desde 0 a 1:

Con un **Template Type: Char**, llamamos a los colores que la **tabla** creó de la siguiente manera:

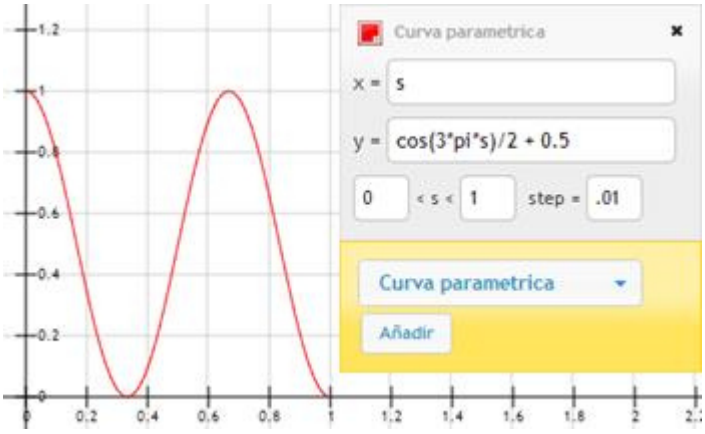


El Gradiente creado por el algoritmo "**%s^3**" se verá en el vídeo más o menos como en la siguiente gráfica:

Notamos que el rojo empieza casi al final de la línea y ya no es simétrico como en el**Gradiente lineal**.

- **Ejemplo 4:**

Algoritmo = "**cos(3*pi*%s)/2 + 0.5**"



Del algoritmo dependerá la transición entre el color 1 y el color2, así que las posibilidades son infinitas, hay tantas transiciones como algoritmos puedan crear.

Para hacer Gradientes de alphas con esta función, solo se deben colocar los **strings alphas** en ambos parámetros de la misma, ejemplo:

mi_tabla = **table.gradient(“&HFF&”, “&HD8&”)**

---

table.gradient2(…): esta función crea una **tabla** con los colores (o alphas) de un**Gradiente Lineal** entre todos los elementos ingresados (dos o más) en la función.

Al igual que la anterior función, el tamaño de la **tabla** es dependiente del **Template Type**.

- **Ejemplo 1:**

mi_tabla = **table.gradient2(“&H00FFFF&”, “&H12BE12&”)**

Entonces la función crea un **Gradiente Lineal** entre los dos colores ingresados.

- **Ejemplo 2:**

alphas = { “&H00&”, “&HAA”, “&H5D&”, “&HFF&” };

mi_tabla = **table.gradient2(alphas)**

En resumen, los tres puntos seguidos (…) en la función, hacen referencia a los colores o alphas a ingresar, o a una **tabla** de colores o de alphas, de la cual necesitemos crear una**tabla** del Gradiente Lineal generados por ellos.

Si le ingresamos 3, 4, 7, 10 o la cantidad de colores que deseemos, la función hará una**tabla**, en donde el tamaño dependerá del **Template Type**, con el **Gradiente Lineal** de todos los colores ingresados. Aplica de la misma manera para los alphas.

---

table.gradient3(Size_table, …): esta función hace exactamente los mismo que la función anterior, pero el tamaño de la **tabla** generada ya no dependerá del tipo de plantilla (**Template Type**), sino que dependerá del parámetro **Size_table**.

- **Ejemplo 1:**

colores = {“&H00FF00&”, “&HFFAA00&”, “&H00DDFF&”};

mi_tabla = **table.gradient3(24, colores)**

24 = Tamaño de la **tabla**

Omití algunas funciones de la ampliación de la Librería “**table**” por un par de motivos: para ver algunos conceptos previos y para poder darle más espacio y mayor número de ejemplos para total comprensión.

Estas han sido unas entradas cargadas con mucha teoría, teoría que no necesariamente deben memorizar ni dominar toda al 100%, pero es mejor tener a la mano el medio para consultar alguna duda, que necesitar un concepto y no saber en dónde buscar.

Lo que con certeza les puedo asegurar es que el tener claro cuáles son y para qué sirve cada una de las variables y funciones de las Librerías vistas en esta serie de documentación, les dará las herramientas necesarias para crear sus propias funciones, en donde las posibilidades son infinitas y los Efectos que se pueden lograr con cada una de ellas no tienen comparación.

En el **Kara Effector** la teoría es tan importante como la práctica, ambas deben ir de la mano, ya que sin una de ellas la otra no sería suficiente. De a poco iremos haciendo un equilibrio entre ellas y por eso en los próximas entradas irá aumentando el número de ejemplos para ponerlos en práctica y aumentar nuestras destrezas.

Espero que en este tramo del camino ya hayan podido ver algunos de los Efectos que por default vienen en el **Kara Effector** y hayan podido entender un poco mejor en qué consiste cada uno de ellos, y eso gracias a los conceptos, variables y funciones vistas.