```
.----===-------------------------------------===----------===--------.
|    /__/|          ___        ___           /  /\         /  /\     |
|   |  |:|         /__/\      /  /\         /  /::\       /  /:/     |
|   |  |:|         \  \:\    /  /:/        /  /:/\:\     /  /:/      |
| __|  |:|          \  \:\  /__/::\       /  /:/~/:/    /  /:/  ___  |
|/__/\_|:|_____ ___  \__\:\ \__\/\:\__   /__/:/ /:/___ /__/:/  /  /\ |
|\  \:\/:::::/ /__/\ |  |:|    \  \:\/\  \  \:\/:::::/ \  \:\ /  /:/ |
| \  \::/~~~~  \  \:\|  |:|     \__\::/   \  \::/~~~~   \  \:\  /:/  |
|  \  \:\       \  \:\__|:|     /__/:/     \  \:\        \  \:\/:/   |
|   \  \:\       \__\::::/      \__\/       \  \:\        \  \::/    |
|    \__\/           ~~~~                    \__\/         \__\/     |
|                                                                    |
`--------------------------------------------------------------------'
.----------------,--------------------------------.------------------.
|                (    The Art of Scripting Vol.1   )      by Grifisx |
`----------------`--------------------------------'------------------'
```
Version: etherea-0.1a/20062001

Cycles, Conditions and Aliases

Start:

What is the meaning of a script? Answer is easy: to simplify your life while chatting, to fit your IRC client in the best ambience ever, with the best comfort, best eye candy and so on.

Let's get it started now, much code as much explanation.
Your KVIrc client is right there, download the latest version; if you are Linux users just compile the cvs o grab the latest snapshot package. If you are Windows users just download the latest snapshot (not the stable one).
The latest snapshot release can be found here:
ftp://ftp.kvirc.net/pub/kvirc/snapshots/

First of all you need to activate the bar which will allow the creation of any script.

From the KVIrc client:
Settings --> Toolbars --> Show scripting
You'll see the bar with all the buttons needed to make KVIrc even more snug, moreover to customize it as much as you like.

This tutorial is all about **alias** and its use.
Aliases are commands that can be useful to add users own customized commands plus the ones you already find in the client.
To create an alias first of all you need to open the Alias Editor using the green pushbutton [/a] that appears in the scripting bar (or use the shortcut CTRL+ALT+1 or use the command “Modify Alias” from the Scripting menu), then create a new command (or alias): right click on the command list on your left and select “Add Alias”, type the name you want for your new alias (ie: mynewalias) in the dialog box and hit OK.
You can easily change the name of the alias previously defined using the “Rename” button on the top right window; please be careful in renaming pre-existent aliases, this operation should be done only if you are experienced users.

Now put the code in the already created alias, let's try with this example code that's going to greet all the users in a channel, though having a limit of six unique names (truly, that seemes unuseful but it's needed for the explanation and comprehension.. moreover nothing's unuseful).

And here's the code:

```
# code n1
%idx=6;
foreach(%i,$chan.users){
if(%idx==0) {
say Geez! You're so many =D Greets to you ALL!;
break;
}
say HELLOW %i;
%idx--;
}
# end code
```

Pushing aside the fact I won't explain the syntaxes used in every KVIrc commands (you've got to deal with *Help -> Browser help..* to have a look to the handbook), here's an deepen:

# code n1
This is a simple comment: comments in *kvs* (stands for KVIrc Scripting) see themselves preceded in line by the "#" (sharp) symbol.

%idx=6*;*
This is the initialization of variable *%idx;* kvs variables are preceded by the "%" (percentage) symbol. And it is very different using *%Idx* and *%idx*, as variables starting with a capital letter are played as Global Variables in KVIrc (please refer to AOS_Vol00(Introduction)), meaning that they won't be destroyed by themselves but need the user set their value to null (ie: *%Idx=""*); also they can be seen by other scripts running for the client. The low letter variables instead have their life exists only for the life of the command or the function they are in, then will get destroyed.

To better understand let's try from any window, by the imput line:
/%glomp =3
and then
/echo %glomp

As you can see the result is... nothing! There is no result as the life of *%glomp* had existed only for the time of the execution of */&glomp =3* and then went destroyed.

Otherwise, trying this out:
/%Glomp =3
/echo %Glomp

We can observe that the result is 3, showing that the variable with the capital letter stood alive even after the execution of the first given command.

In our code we do analyze the variable used to greet 6 people in a chan, avoiding to greet every user in a crowded channel.

```
foreach(%i,$chan.users){
code
}
```

Here is the use of the *foreach;* command, its syntax can be seen in the KVIrc guide while you can see *$chan.users* is a KVIrc function returning in an array (to be considred as a list) every nickname of every user at the moment in the channel (other functions can be found in the *Help -> Browser help..*).
See it in action in any channel and try it out:
/echo $chan.users

Ma result is:
[09:31:00] Grifisx_away,morfeux,AM1C0H4CK3R,Cif,franzi_,MysteryBETA,nonno_rik,ohi[gentoo],ohiahiohi,redsend,Scorp[ZzZ],zerymo.

That %i, found in the *foreach*, means an element of the array (the actual one) has been defined by the sintax.
So that the code means "for every element(*foreach*) in the array *$chan.users* do execute this code { code to be executed }". And let's see the code we want it to do.

```
if(%idx==0) {
say Geez! You're so many =D Greets to you ALL!;
break;
}
```

It's a small check to avoid greeting all the users in the channels
Qui facciamo un piccolo controllo che ci eviterà di salutare tutti sul canale, e limiterà il saluto a solo 6 persone.

If (variable *%idx* is equal to zero) execute this code {code}.

say is a KVIrc command used to write some text a defined window.
break is used to break any cycle, in this case it will break the *foreach* cycle.
So the statement "**Geez! You're so many =D Greets to you ALL!**" it's defined with *say* and the cycle breaks.

Let's have a look when the above condition *(if(%idx==0))* is not true:

```
say Hello %i;
%idx--;
```

That means: *say Hello %i* (we stated *%i* as the actual element of the array in which we are doing the cycle, so the nickname) *and decrement %idx*.

On the first turn *%idx* is equal to 6, the condition *(%idx=0)* is not true, it does *say Hello nickname* and decrements *%idx* becoming 5 and executes another loop; this all until the given condition is true, ie *%idx* is equal to zero, and will break.

Here we go for a quick overview of some more KVIrc commands, just

before editing more the code:
On the input line of any channel try:
CTRL+B : it will appear a boxed B (named BOLD) and everything written on the very next will be displayed as bold text;
CTRL+U : it will appear a boxed U (named UNDERLINED) and everything written on the very nex will be displayed as underlined text;
CTRL+K : it will appear a boxed K and a little table of colors associated with numbers, type the number of the chosen color (or single click it with your mouse) and everything written on the very next will be displayed as colored text. (You can make tons of different color sequences, just chose for every word or character a different colors. Note that all the defined colors can be specified and edited/changed via the *Preferences Menu -> Themes -> Default colors*).
These are only a few commands we do need for the moment.

Scripting uses the *$b()* for bold text, *$u()* for underlined text and *$k(<background_color_number>, [foreground_color_number])* for colored text.
Give it a try on the input line:

/echo $b()Hello

Then we're ready to drop the code using some modification:

```
# code n1
%idx=6;
foreach(%i,$chan.users){
if(%idx==0) {
say Geez! You're so many =D Greets to you ALL!;
break;
}
say Hello $k($rand(12))%i;
%idx--;
}
# end code
```

There's a new command as you can see: *$k($rand(12))*. As known *$k()* is used for the colors and *$rand(<value_lag>)* is used to obtain a random number between zero and the given number in the brackets (this can be also seen in the KVIrc Help).

Try this:
/echo $rand(12)

The result is a random number between 0 and 12.
You could object it is not useful to chose two random colors on the follow-on, as casualty allows this randomness, so here's the need of a control done to modify the actual number color if it is the same as the other one.
We need to remember the old value, meaning we do need to create a variable able to store it, and have a match check of both generated numbers; in case the values are identical the go-to-change value number will be the second in time, to which will be added a new value (for example 1) in order to change (increment) the value itself and return a different color.

Transforming into code:

Do create the variable *%oldColor* with initialized value = 0;
do create variable *%newColor* with initialized value *$rand(12)*;
do create the check, if *%oldColor == %newColor* then *%newColor++*;
store in *%oldColor* the actual value of *%newColor.*

The code:

```
%idx=6;
%oldColor =0;
foreach(%i,$chan.users){
%newColor=$rand(12)
if(%idx==0) {
say Geez! You're so many =D Greets to you ALL!;
break;
}
if (%newColor==%oldColor){ %newColor++;};
say Hello $k(%newColor)%i;
%oldColor=%newColor;
%idx--;
}
```

There is a significant thing to look at: the initialization of variable *%oldColor* stands out of the cycle, otherwise it would began to initialize itself every time.
If you like to have your life more hard than a thorned vine, you would likely prefer the nicknames written in different capitalizing, taking turn of an up and a lower case letter.
Before examine the next portion of code take a moment to open the KVIrc Help to the *Functions* side, switch to *s* letter and see how many functions are there, able to modify strings; are all identified with prefix *$str.xxxxxx*; we need a up/lower letter case so we do need these two:
$str.upcase(<string_to_convert>)
$str.lowcase(<string_to_convert>)

On the input line, type:
/echo $str.upcase(“*geeeeeeeez*”)

Hot work is to convert not all the singles nicknames, but letter by letter, we need a function able to divide the string:
$str.section (< string_to_be_divided >, <separator_element>, <position_where_to_divide>, <position_where_to_stop_dividing>).

Try it out with *echo*:
/echo $str.section("Name Surname**Nick**Phone","**", 2, 2);**

The result is “Nick”.
Moreover there's the need of the *$str.len(<string>)* function that returns the lenght of the string.

So the code:

```
%idxN=6;
%oldColor =0;
foreach(%i,$chan.users){
%szNick=%i
%idx=1
%itmp=1
while(%idx!=($str.len(%szNick)+1))
{
      if(%itmp==1)
      {
            %sztmp =$str.upcase($str.section(%szNick,"",%idx,%idx));
            %itmp=0
      }
      else
      {
            %sztmp =$str.lowcase($str.section(%szNick,"",%idx,%idx));
            %itmp=1
      }
      %szNickColorato=%szNickColored%sztmp
      %idx++;
}
%newColor=$rand(12)
if(%idxN==0) {
echo Geez! You're so many =D Greets to you ALL!;
break;
}
if (%newColor==%oldColor){ %newColor++;};
say Hello $k(%newColor)%szNickColorato;
%szNickColored=""
%oldColor=%newColor;
%idxN--;
}
```

Fortunately it's not that complicated.
%idx had been changed into *%indxN* as my habit to call every index *%idx*, following this I need to separate indexes in order not to mistake them.
All variables are initialized, then the cycle can run.

Let's check the essential steps of the transformation and the engine:

while(%idx!=($str.len(%szNick)+1))
until the *%idx* variable is different from the nick+1 lenght (the end has not been reached);

%sztmp =$str.upcase($str.section(%szNick,"",%idx,%idx))
%sztmp is equal to the letter trasformed in Upper case; here I do select *nick(%szNick)* using the char "" (ie: no char, so it is divided letter by letter) starting from actual index until it reaches the actual index itself.

Just use my nickname: Grifisx
/echo $str.section("Grifisx","",1,1)
the result is "G".

And for example:
/echo $str.lowcase($str.section("Grifisx","",1,1))
will have the same result as "g", as it is the same as:
/echo $str.lowcase("G").

As you can see in the code **%itmp=0** is set, so on the next loop it won't be entering in the *if* but in the *else*; *%idx* instead will increment on every cycle:

$str.upcase($str.section("Grifisx","",1,1))
$str.lowcase($str.section("Grifisx","",2,2))

```
$str.upcase($str.section("Grifisx","",3,3))
$str.lowcase($str.section("Grifisx","",4,4))
```

As you can see there's:
%szNickColored=%szNickColorato%sztmp
because **%sztmp** contains the actual transformed letter, in this case of Grifisx let's put the “g”, so it is:
stood that **%szNickColored** is empty in the beginning :

%szNickColorato=%szNickColored%sztmp --> ””=””g
we have then **%szNickColored** equal to “g”, so on the next loop we'll have:
%szNickColored=%szNickColored%sztmp --> g=gR (because %sztmp will hold the second letter of my nickname in uppercase).

So **%szNickColored** is equal to “gR”, and on the next loop:
%szNickColored=%szNickColored%sztmp --> gR=gRi

.. and so on, until:
while(%idx!=($str.len(%szNick)+1))
until the lenght of my nickname is not exceeded by 1 (meaning the nickname is ended).

That's not enough!
We want our greet turned out into a beautiful rainbow drop!
And put your hands on the code in order to turn a string into rainbow.

```
%idxN=6;
%oldColor =0;
foreach(%i,$chan.users){
%szNick=%i
%idx=1
%itmp=1
while(%idx!=($str.len(%szNick)+1))
{
      if(%itmp==1)
      {
            %sztmp =$str.upcase($str.section(%szNick,"",%idx,%idx));
            %itmp=0
      }
      else
      {
            %sztmp =$str.lowcase($str.section(%szNick,"",%idx,%idx));
            %itmp=1
      }
      %szNickColored=%szNickColored%sztmp
      %idx++;
}
%newColor=$rand(12)
if(%idxN==0) {
%szFrase= "Geez! You're so many =D Greets to you ALL!";
%idxW=1
while(%idxW!=($str.len(%szFrase)+1))
{
      %sztmp2 =  $str.section(%szStatement,"",%idxW,%idxW)
      %szStatementColor=%szStatementColor$k($rand(15))%sztmp2
      %idxW++;
}
say %szStatementColor;
break;
}
if (%newColor==%oldColor){ %newColor++;};
say Hello $k(%newColor)%szNickColored;
%szNickColored=""
%oldColor=%newColor;
%idxN--;
}
```

There's not that much to explain as we already saw many of those constructions.
Just a note: this small script can be annoying in some channels you may join, we have built it up with the only purpose to understand and improve the KVIrc scripting system that will lead us further to the creation of more complex scipts and more.

Again, before have my greet, I'd like to end up with the aliases, teaching you on how to switch from an “argument” to an “alias”.

If you want to write:
/colorize “*hello pals*”

first of all create the *colorize* alias stating that its job is only to give colors to the words it's followed by (in this case “*hello pals*”), then insert this (in the alias itself):

```
echo $k($rand(12)) $0-
```

Just it!
$0- indicates ALL the statment (or words), starting from the beginning, that will be read.
$0 (without -) indicates the first element to keep in; *$1* indicates the second element, *$2* the third and so on.. (*$1-* indicates all the statement or the words starting from the second element; *$2-* indicates all the statement or the words starting from the third element..).

One last example on the alias:

```
echo $k($0) $1-
```

and from the input line type:
/colorize 5 I write in colors!

We used the firs value (*$0*) to give the color to the statement, then the *$1-* to take the rest of the statement starting from the first element (*$0* is the number of the color).

Quick note: to create an alias on-the-fly, without having to open the editor, just do:
/alias(aliasname){alias code;}

For example:
/alias(colorize){echo $k($0) $1-;}

It will create the “*colorize*” alias and will instruct it with the

code *echo $k($0) $1-;*
To destroy it, directly from the input line, just type:
/alias(colorize){}

Without any code within the brackets.

Consider aliases as functions in the way they are used, to make themselves give out values using the **return** command; again in the input line:

```
/alias(sum3){ return $($0 + $1 + $2); }; [RETURN]
/%Somma = $sum3(3,4,5) [RETURN]
/echo %Somma [RETURN]
```

The *sum3* alias has been created (it contains the code *$($0 + $1 + $2)* noticing that *$(arithmetical_expression)* is used to compute the value of the arithmetical expression within the brakets); this alias has been treated as a function (note *$sum3* instead of *sum3*) giving it out three numbers *(3,4,5)* and has been used to gain the value of the sum of the three numbers, storing it in the variable *%Somma.*
The last command returns on display the content of variable %Somma.

/ECHO STOP.

- - - - - - -- - - - - - - - -- - - - - - - - - - -- - - - - - -- - - - -
"You see things; and you say `*Why?*' But I dream things that never were; and I say `*Why not?*"
(George Bernad Shaw)
- - - - - - - - -- - - - - - - - -- -- - - - - - - - - - - - - - - - - - -
Grifisx