YOLOL Tricks

From Starbase wiki

YOLOL is a rather constrained language. The line and chip length limits and slow execution speed incentivize clever shortcuts. The small math library and limited operation space require building larger operations from smaller building blocks. To those ends, this page serves to catalog a variety of things you can accomplish with YOLOL that might not be obvious from the main [YOLOL] documentation.

Contents

General tips

Anything that accept numbers also accept something that represents a number. For example if you wanted to use a button to turn something on instead of doing

```
if :Button then :Device=1 else :Device=0 end
```

You can achieve the same thing by simply doing

```
:Device=:Button
```

":Button" is actually just a value, most buttons are either 0 (not clicked) or 1 (clicked) so setting :Device to be :Button is really no different from setting it to 0 and 1, its just that our number is now "dynamic" in the sense that it is directly tied to the value of the :Button. Understanding this can be used for many things, we can incorporate this into our goto# by simply doing something like "goto:Button+1" which will goto1 if :Button is 0 and goto2 if :Button is 1. So that means we can both use something that has the value of a number instead of a number as well as carry out math to determine what the number actually should be. Assume we want :Device to get the value 100 if :Button is pressed, and have the value 0 if it is not pressed. We can do that like this:

```
:Device=:Button*100
```

It is also useful to know that if a line contains a runtime error then this line will be skipped entirely. This means you can skip lines by dividing by 0. Just directly dividing by 0 is of course not helpful but if we use a "dynamic value" in the sense that we are dividing by a variable, such as :Button, that means we sometimes will have a line containing no errors so it runs and sometimes it contains an error and will be skipped.

```
goto2/:Button
```

If our :Button is 0 this is dividing by zero so the entire line is skipped, if :Button is 1 we are instead dividing 2 with 1 which returns 2.

Need more decimals?

YOLOL supports 3 decimal places, 1.234, which may cause you some issues. If your final number however isn't the one that needs a higher amount of accuracy than this but rather the requirement for more than 3 decimals is somewhere in the equation giving you your number then you can solve this issue by multiplying your numbers used during the equation and then dividing the final number. For example if you need to multiply x with 1.2345 and .234 simply is not accurate enough you could instead multiply x with 12.345 and then divide the result by the same amount, in this case 10.

Don't have professional chips?

Trigonometric functions are locked behind professional chips, which you can't build at the moment (there is a few chips left from alpha that circulate the market). Here is solution:

polynomial approximations for sin/cos functions ***(!! for sin/cos make sure -180<:x<180 !!)***[1]

sin(x):

```
i=:x/180 :sinx=4*i*(1-ABS i)
```

cos(x):

```
i=:x/180 i+=.5-2*(i>.5) :cosx1=4*i*(1-ABS i)
```
(using the cos(x)=sin(x+90) identity)

```
i=:x/180 :cosx2=i^2*(-6+4*ABS i)+1
```
(standalone derivation; slightly shorter but different error bars so ymmv when using it with sin)

asin(y):

```
:asiny=40*:y^3+50*:y
```

acos(y):

```
:acosy=-40*:y^3-50*:y+90
```

atan(y):

```
i=:y/SQRT(1+:y^2) :atany=40*i^3+50*i
```
(using the asin identity)

String Manipulation

Nickname	Code	Explanation
If Empty	**string-(string+otherstring)**	If **otherstring** is empty then this evaluates to an empty string, else it evaluates to **string**
Select	**a="foo1" b="bar2" c="meh3" x=2 s=a+b+c-x-a-b-c s=="bar"**	One string from a set can be selected by concatenating them with indices, then removing an index, then removing all the strings which will fail for the un-indexed one. If one index is a substring of another then the order of the strings will matter.
Contains	**s="~meh~foo~bar~" t="foo" c=s!=s-("~"+t+"~")**	Evaluates to 1 if **t** is present in test set **s**, and 0 if not Alternatively, if the values in the string to test (**s**) are arranged in alphabetically descending order, **s>s-("~"+t+"~")** can also be used. For non-distinct values, (for example searching for **"tin"** in the string **"non-distinct values"**) the delimiter (**"~"**) can be omitted. Depending on the values that can occur in **s** and **t**, some or all of the delimiters (**"~"**) can be omitted, or changed into other values, such as numbers.

Number tricks

Nickname	Code	Explanation
Not	**0^b 1-b b==0**	Used to negate a **0** or **1** value. **0^b**: Does not need parenthesis when used in a multiplication. Works any non-negative input value, not only **0** and **1**. Requires an Advanced or Professional chip **1-b**: Works **only** with **0** and **1**. Works on all chips. **b==0**: Works with all input values, not only **0** and **1**. Works on all chips.
Select	**a+(b-a)*s a*0^s+b*s**	Select one of **a** or **b** based on a **0** or **1**, **s** select signal. Equivalent to **if s then result=b else result=a end**
Hex	**0xHEXDIGITS**	write a number in base 16, digits from 0-9 and a,b,c,d,e,f; works only in YOLOL
Scientific	**AeB**	A*10^B; works in YOLOL and when written by the U-tool in device fields

Skipping IF Statements

If statements can in many cases be avoided which can save us a considerable amount of characters, as we will no longer waste characters on "if", "then", and "end". Multiplication (*) works as a replacement for AND while addition (+) works as a replacement for OR in many scenarios.

```
if :ButtonState==1 and :DoorState==1 then :LampOn=1 end
```

While this if statement could be made shorter by skipping out on unnecessary spaces and removing the unnecessary "==1" part of the comparison we could also skip out on using an if statement entirely.

```
:LampOn=:ButtonState*:DoorState
```

This will assign LampOn the value of ButtonState multiplied with DoorState. if both are 1 that returns a 1, if either one (or both) of them is zero then it returns zero, making this a functional replacement for an if statement containing AND. We could do the same thing for OR by instead using addition.

Using math to avoid the necessity of IF statements works in many cases and tend to be easy to do when the values involved are 0 and 1, such as for anything which is either on or off. If you need to shorten down your code for one reason or the other this is a good venue to explore. Lets finish this section with a slightly beefier example of this in use.

```
if :One and :Two then :LampOn=1 goto2 end if :One<1 and :Three then :LampOn=1 end
```

Here we are checking if :One and :Two are true (equal to 1) and if they are we change :LampOn to 1 and move to line 2. IF they are not we check if One is False (0) and Three is true (1) and in that case we still turn on the lamp. However if we skip out on using if statements to approach this we could shorten it to this:

```
:LampOn=(:One*:Two)+((1-:One)*:Three)
```

Keep this in mind when writing your yolol and you will find yourself saving a lot of characters and fitting more things on the same line and even more things on the same chip!

When you need to disable whole script with one button and squeeze-in some variables initialisation:

```
if :Button != 1 then goto1 end a = 123 b = 456 c = 789
```

This can be shortened to:

```
a=123 b=456 c=789 goto1+:Button
```

When **Button** disabled **goto** will jump to line 1, otherwise it will jump to line 2.

Code Golf

If you want to develop readable code and only shorten it for deployment, consider using yodk (https://github.com/dbaumgarten/yodk) or Yololc (https://github.com/martindevans/Yolol) which can minify your code for you.

Shortening identifiers to one character saves bytes, use **f=b** rather than **foo=bar**.

Reassigning fields to identifiers saves bytes if you use them more than a few times. **f=:f f f f f f f f f** is shorter than **:f :f :f :f :f :f :f :f :f** .

Most whitespace is optional. You only actually need a space when code would be ambiguous without it. Spaces can almost always be omitted before **:**, after **if** or **then** or **end**, between a number or symbol and a letter, and in various other places.

1. Extremely unprofessional sinusoidal approximations (https://www.reddit.com/r/starbase/comments/puz32d/extremely_unprofessional_sinusoidal_approximations/)