

Dear Users,

In [iguidestocks](#) you can write your own formulas as you want. [iguidestocks](#) provides a full featured API for writing formulas.

In main [iguidestocks](#) document you can learn how to write a formula in this document we will not talk in short about all features which software provide

Writing Buy/Sell Formula

We will start with writing strategy

When we write strategy/formula we can use all operator loop and API and can write formula in such a way that our graph can not only show buy/sell signal but can do more

We will start with write simple [iguidestocks](#) formula

When you write formula in [iguidestocks](#) formula language here are some of the points which you must remember

1. To get buy/sell signal you must use buy {<text>} and sell {<text>} function in your formula. On the bar where the condition is met will show a white color (or the color you have configured by changing buy/sell color) arrow and red color arrow for sell.

To show you this as an example I will create a very simple formula which will generate buy/sell signal for you.

Suppose you want to generate buy signal when closing price crossover 15 days simple moving average.

```
a := cross[close;mov[close;14;s;0];0]
if ( a == 1 )
{
  buy{}
}
b := cross[mov[close;14;s;0]; close;0]
if ( b == 1 )
{
  sell{}
}
```

In this formula we used indicator cross, which basically find out when first data array crosses second data array. It returns 1 in those conditions.

So our formula generates a buy signal, when we get 1, means in condition 1 when close crosses simple moving average, it generates a buy signal and visa a versa.

Applying Buy/Sell Formula on Buy/Sell Screener Screen

So now you know how to get a buy/sell signal in iguidestocks formula language.

2. Now you have written the formula for getting buy/sell signal. if you would like to use this formula in **"Buy/Sell Screener"** and would like to get buy/sell signal for all stocks in a category then you need to add following and modify your formula like this

```
a := cross[close;mov[close;14;s;0];0]
if ( a == 1 )
{
  buy{}
  BUYTYPE := "STRONGBUY"
}
b := cross[mov[close;14;s;0]; close;0]
if ( b == 1 )
{
  sell{}
  BUYTYPE := "STRONGSELL"
}
```

After doing this now you can run this formula on all stocks and can get buy/sell signal which will be shown in different color on that screen.

If you want to show any specific comment then also you can do by using alert function like this

```
a := cross[close;mov[close;14;s;0];0]
if ( a == 1 )
{
  buy{}
  BUYTYPE := "STRONGBUY"
  Alert{"closing now up from moving avg"}
}
```

```

b := cross[mov[close;14;s;0]; close;0]
if ( b == 1 )
{
  sell{}
  BUYTYPE := "STRONGSELL"
  Alert{"closing now down from moving avg"}
}

```

After doing this save this formula and run on graph by using apply strategy icon and get buy/sell signal. And to run for a specific category click on menu → buy/sell screener and select this formula and any category and execute it. You will see red, green rows if any signal generated or all other will be in black.

Now you have learned how to get buy/sell signal. alert will open a popup whenever you click on graph and if there is any alert. Like in this example on buy signal if you click it will open alert popup box.

Applying value dynamically on a formula

Now we will understand how we can change fix value in this case we have used 14 days moving average into a parameterized so that you can pass value as a parameter and also while calling this formula you can pass the value.

To change fix value into parameter we need to change or formula like this

```

@PARAMETER DAYS
@VALUE 14
@COMMENT Moving avg buy/sell
a := cross[close;mov[close;$days$s;s;0];0]
if ( a == 1 )
{
  buy{}
  BUYTYPE := "STRONGBUY"
  Alert{"closing now up from moving avg"}
}
b := cross[mov[close; $days$s;s;0]; close;0]
if ( b == 1 )
{

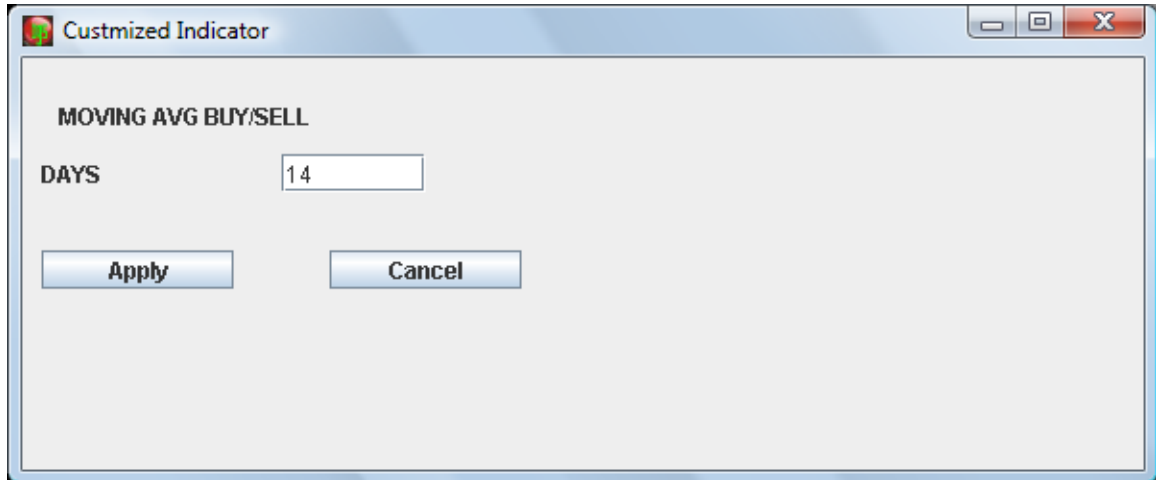
```

```

sell{}
BUYTYPE := "STRONGSELL"
Alert{"closing now down from moving avg"}
}

```

Now our formula has parameter days in it. And when you execute it will open a pop up like this



If you want to provide multiple parameters then you need to make following modification in your formula like this

```

@PARAMETER DAYS avgtype
@VALUE 14 S
@COMMENT Moving avg buy/sell
a := cross[close;mov[close;$days$;$avgtype$;0];0]
if ( a == 1 )
{
    buy{}
    BUYTYPE := "STRONGBUY"
    Alert{"closing now up from moving avg"}
}
b := cross[mov[close; $days$;s;0]; close;0]
if ( b == 1 )
{
    sell{}
    BUYTYPE := "STRONGSELL"
    Alert{"closing now down from moving avg"}
}

```

Now it will ask for 2 parameters which are days and avgtype.

Writing function

Now we will learn how to write function and how to use our formula in other formula.

In iguidestocks in a formula we can create function or can use a simple statements as an indicator or can call another formula

We will start and will take each thing one by one

In our first formula we will use statement as an indicator.

Let's say we would like to draw a graph of simple moving average for price difference

Here we can use it like this

To make a statement as an indicator we can use # and then variable name then colon and expression like this

```
#a: close[0] - open[0]

if ( a[0] > 0 and a[-1] > 0 )
{
    buy{}
}
```

Here we are generating a buy signal when today and yesterday stock closed in +ive.

Here we declared variable a as a data array which has value close[0] – open[0].

We can declare multiple data array like this and can use them.

Now suppose we need to calculate some values using multiple statements then a single statement will not work so to do so we need to declare function which will return data array.

In this example we will declare function.

In iguidestocks we can declare function like this

```
Function
{
    @variable
    <stmts>
}
```

Now in this example we will define a function and then we will use it.

Let's say in this example we will find bullish engulfing pattern and we will defined it as a function in same file.

```
if ( buleng[0] )
{
    Buy{}
}
Function buleng
{
    O1 := OPEN[-1]
    O2 := OPEN[-2]
    H1 := HIGH[-1]
    H2 := HIGH[-2]
    L1 := LOW[-1]
    L2 := LOW[-1]
    C1 := CLOSE[-1]
    C2 := CLOSE[-2]
    O := OPEN[0]
    C := CLOSE[0]
    H := HIGH[0]
    L := LOW[0]
    X1 := ( O1 > C1 )
    X2 := ( C > O )
    X3 := ( C >= O1 )
    X4 := ( C1 >= O )
    X5 := ( ( C - O ) > ( O1 - C1 ) )
    X := ( X1 AND X2 AND X3 AND X4 AND X5 )
}
```

When you defined formula it is calculated as an indicator and you can pass parameter too. In this function definition we will write our function with parameter and we will use it.

```
Plot1{mysma[14;0],red}  
Function mysma  
{  
@PARAMETER DAYS  
    Sma[$days$;0]  
}
```

You can also save your formula same way in library directory and then you can call it in your program.

Like the example which we have written previously we can save it and call it.

There is lots of function like candlength, barlength and many candlestick functions are written and can be called

Advance API Functions in IGS 3.0

Now we have talked about how to use and write formulas in iguidestocks's formula language.

In igs 3.0 a lot of advance functions been added we will talk about them now.

When you write any formula in igs it runs for each bar and return value for that bar but sometimes it is needed to calculate values globally and also you need to initialize value or use it in formula.

Igs 3.0 fl provides two global variables which are

Barno and noofbars

These can be used in formula for initialization or for some other stuff. The first bar is zero so last bar no is barno - 1 .

To add any variable as a global variable you need to use addtglobal function which looks like this

Addtglobal{varibale, value}

Now you can use this variable in function the value will be used from global variable area. If you need to change the value of that variable then also you need to use addtglobal function a normal assignment will make it a local variable available for that bar only.

Some of this example will show you how you can use them.

Let's say you want to calculate average volume and then you would like to use it in each bar . To do so first bar it we will calculate average volume and will add it in global variable.

```
if ( barno == 0 )
{
    i := 0
    totvolume := 0
    while ( i < noofbars )
    {
        vol := getvalue{volume[0],i}
        totvolume := totvolume + vol
        i := i + 1
    }
    avgvolume := totvolume / i
    print{ " avg volume is " + avgvolume}
}
```

Here are some of more functions which are very useful like

Getvalue{<expr>,barpos}

In this function expr is calculated for specified bar position .

Some of function which is also very useful

Expose{<variable>}

This function will expose your variable to calling program. Means when you expect more than one return value from your function you can this function.

This is very useful function when you write any query. Like suppose you would like to display pivot value and support values then you use it like this

```
PIVOTPOINT := ( HIGH[0] + LOW[0] + CLOSE[0] ) / 3
P := PIVOTPOINT
R1 := 2 * P - LOW[0]
S1 := 2 * P - HIGH[0]
R2 := ( P - S1 ) + R1
S2 := P - ( R2 - S1 )
expose{p}
expose{r1}
expose{r2}
expose{s1}
expose{s2}
TRUE
```

Now save this function in library by name mypivot.fl and then write following query.

```
select symbol,p,r1,r2,s1,s2 where mypivot == true
```

This will run mypivot function which basically returning true and exposing p,r1,r2,s1,s2 variables and we are displaying those variables in our query.

Like this some more function are

getnotzerobaraway{nth,dataarray}

This function will return the nth nonzero value from the data array. This function is very useful when you want to find out double top, bottom pattern as using this function over peak indicator you can find out two nearest top and can check if they are nearly equal

getnotzerovalue{nth,dataarray}

This function will return the nth nonzero value from the data array. This function is very useful when you want to find out double top, bottom pattern as using this function over peak indicator you can find out two nearest top and can check if they are nearly equal

sumbars{expr,noofbars,direction}

This will sum expression for noofbars starting from current bar. The direction can be a-ahead or b-backward

In this example we will count average volume of the security

```
if ( barno == 0 )
{
x := sumbars{volume[0],noofbars,a}
avgvolume := volume[0] / noofbars
println{avgvolume}
}
```

sumcondmeet{expr,noofcount,direction}

This will start from current position and calculate expression for count . If any expression is true then it will add it and at end will return sum of all expression which were matched.

If (barno == noofbars - 1)

in this example we will calculate how much red candle stock created in last 5 days. Means how many time stock went in red (close is less than open)

```
{
i := sumcondmeet{close[0] open[0], 5,b}
println{i}
}
```

updatevar{var,expr,position}

This function should be used to update any variable for old calculated variable . Suppose you have written a formula for stock and you would like to change value of previously calculated variable then you should use this function

```
a := close[0]
updatevar{a,0,barno - 1}
```

Initially we assigned value close[0] to a and then when formula is executed on nextbar it update value of variable a for previous bar.

Stock scanning using IGS query Language

Now we know how to use formula language. Now we would talk about scanning stocks .

To scan stocks iguidestocks has developed its own query language which follows industry standard sql language syntax.

To write any query you need to click on stock filter menu option and then you can write query.

The syntax of query language looks like this

```
Select <columnnames> from table where <conditions> sortby  
<columnname>
```

Here column name can be any indicator or basic open,high,low,close and symbol.

For example you need to display only open,high,low,close for a specified category the formula will be

```
Select open,high,low,close from table
```

This will return all stocks for specified category.

Now if you want to add a condition like open should be greater than close then query would be

```
Select symbol,open,high,low,close from table where open > close and  
it will return stocks which has open greater than close
```

Now suppose you would like to sort them on open price then query will change like this

```
Select symbol, open,high,low,close from table where open > close  
sortby open and data will be sorted by open price.
```

Suppose you would like to display some indicator and also would like to add a condition based on indicator then formula will change like this

```
Select symbol,open,high,low,locse,rsi[14;0] where open > close and  
rsi[14;0] > 70
```

This way you can display any indicator or add it in condition.

Now after that you would like to display some calculation in your query then also igs supports it like this

Let's say you would like to display percentage change in stock price. Now you can first define percentage change and then can use it your query like this

```
#change: ( close[0] - close[-1] ) * 100 / close[0]  
Select symbol,change from table orderby change
```

This will display all stocks with change sortedby percentage change.

Now if you would like to add change in last 3 days then you can write query like this

```
#change: ( close[0] - close[-1] ) * 100 / close[0]  
Select symbol,change[0],change[-1],change[-2] from table orderby  
change
```

In iguidestocks you can also call inbuilt formulas which are in library folder . All functions related to candlestick pattern are stored in library . Let's say would like to use

Inbuilt formula in the query (only in condition, you cannot use it in query column).

```
Select symbol where greencandle == true .
```

This will return all stocks which has green candle.

Now we will go in more detail let's say you also would like to display value returned by any function.

Suppose one function called barlength which you would like to display. Then you need to write your query this way

```
#mybarlength:barlength  
Select symbol,mybarlength from table
```

Now sometimes condition occurs where you want to get more than one value from your formula . One of the example is getting pivot, support and resistance values. When this type of condition occurs then iguidestocks has added a specific function called expose in its API which will allow exposing multiple variables other than the return value from formula.

Expose is the function which you can use to do the same.

Here is the example for same

First write function and save it in library folder. We have already written one function called pivot and saved it in library the code of that function looks like this

```
PIVOTPOINT := ( HIGH[0] + LOW[0] + CLOSE[0] ) / 3
P := PIVOTPOINT
R1 := 2 * P - LOW[0]
S1 := 2 * P - HIGH[0]
R2 := ( P - S1 ) + R1
S2 := P - ( R2 - S1 )
expose{p}
expose{r1}
expose{r2}
expose{s1}
expose{s2}
TRUE
```

If you see in this formula we have exposed following variables
p,r1,r2,s1,s2

Now we can use them in query like this

```
select symbol,p,s1,s2,r1,r2 from table where pivot
```

This query is making a call to pivot which is internally returning all exposed variables

In this example we will find out double top pattern using igs fl and igs query.

Here are steps for same

First of all we will write following formula and will save by name as doubletop in library folder

```

@PARAMETER MINCHANGE TOPDIFF
@VALUE 10 3
@COMMENT PROVIDE MINCHANGE TO GET PEAK & MAX DIFF IN
NEAREST TOPS
#P:PEAK[CLOSE[0];$MINCHANGE$]
#T:TROUGH[CLOSE[0];$MINCHANGE$]
B := P
RET := FALSE
IF ( B <> 0 )
{
  X := GETNOTZEROVALUE{2,P}
  D := ( ( X - B ) * 100 ) / ( X + .001 )
  D := ABS{D}
  IF ( D < $TOPDIFF$ )
  {
    POS := GETNOTZEROBARAWAY{2,P}
    LINEXY{BARNO,B,BARNO - POS,X,YELLOW}
    GETVALUE{TEXT{ CLOSE[0],RED,"T1-" + CLOSE[0],8
},BARNO - POS}
    GETVALUE{TEXT{ CLOSE[0],RED,"T2-" + CLOSE[0] ,8 },BARNO
}
}

  TVAL := GETNOTZEROVALUE{1,T}
  TPOS := BARNO - GETNOTZEROBARAWAY{1,T}
  LINEXY{TPOS - 30,TVAL,TPOS + 30,TVAL,YELLOW}
  GETVALUE{TEXT{ CLOSE[0],RED,"N-" + CLOSE[0] ,8 },TPOS }
  RET := TRUE
}

}
ALERTTEXT{"T1-TOP1,T2-TOP2,N-NECKLINE"}
ALERTTEXT{"DOUBLE TOP PATTERN"}
RET

```

In this formula ret variable is the variable returned by function and we created dynamic variable like minchange and topdiff

Now this formula will return value of ret which in sense be true if double top pattern is found for specified value of minchange and topdiff.

Now following query

```
select symbol from table where doubletop[10;3;0] == true
```

 will
going to find out doubletop pattern for you. Here we passed
minchange value as 10 and topdiff value as 3 and 0 for today.

Now we have covered most of the part of using igs fl and query language.

In igs fl we also support