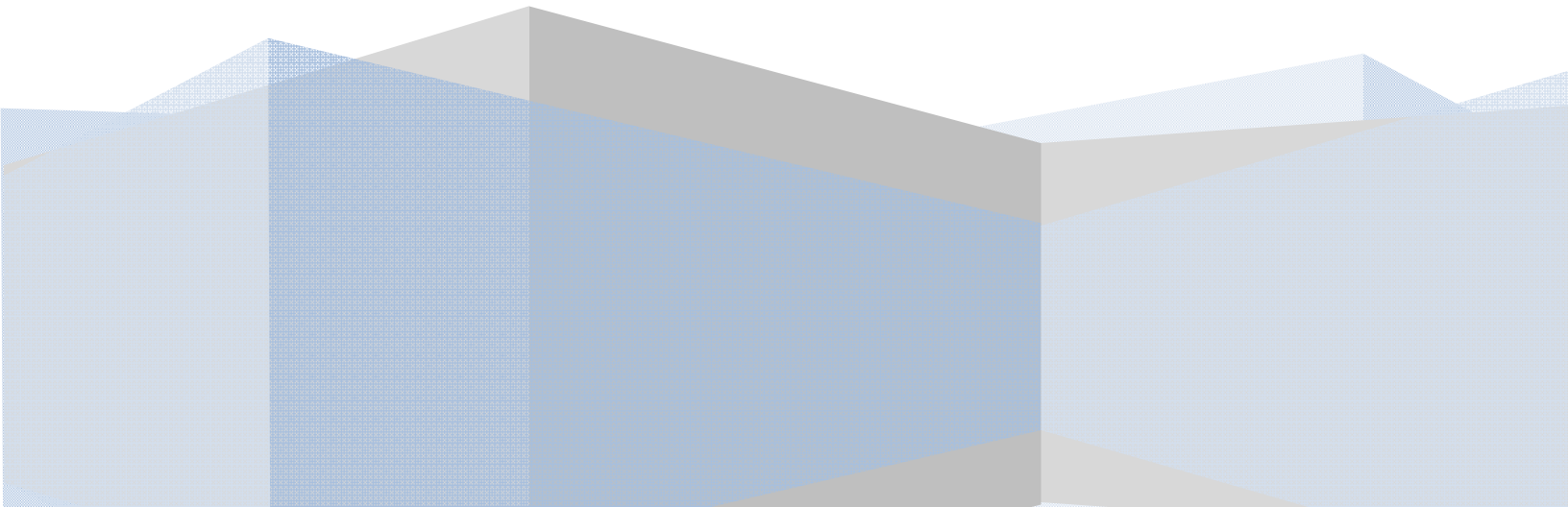


www.iGuideStocks.com

How to use IGS Formula Language



Language support - IGS FL (*iGuideStocks Formula Language*)

iGuideStocks has its own proprietary programming language which will allow you to create your own strategies, indicators, scans, queries and more, which will guide you to make smart investment decisions.

Some of the features supported by IGS FL are:

1. Easy and powerful language
2. Major candle stick pattern recognition library, which can be used while creating strategy
3. Support of if-else loop, while loop, arrays etc.
4. Create your own function library (or use existing one)
5. Support of function call in any strategy
6. String operation, Print, Alert messages.
7. Support of drawing component in language like text, circle, rectangle etc.
8. Plot graph using any plot function
9. Support Mathematic operation like abs, sqrt, pow
10. Support of Buy, Sell function with buy / sell arrow on chart as condition is met
11. Support of in-built major indicator like RSI, MACD, Bollinger bands, Moving averages etc
12. Customized indicator support
13. Comment support to understand your own logic of strategy
14. Inspect variables after execution.

iGuideStocks has several advanced features based on IGS Formula Language such as, back testing, customized indicator, stock screening, back testing comparison, writing functions, applying strategies and finding out buy/sell signal with comments. You can modify the code and apply the changes on the fly and graph will be displayed with changes by clicking on execute.

IGS FL is very easy to use and does not require compilation. We have built in code checks and user just need to write and execute a specific formula or strategy. The Language supports looping, assigning variables and function call.

Programming Elements of IGS FL

The language has following components

- 1) Variables
- 2) Statement and Expression
- 3) In-built Function
- 4) In-built Data Array or Indicators
- 5) Customized indicator or Data Array
- 6) Function Call
- 7) Looping, if-else, while
- 8) Operator (+,-,*,/,and, or, >, <,<=,<>,<=,>=)
- 9) Statement

Writing a Very Simple Program using IGS FL

Here is an example of very first program to print opening price of a stock on output window for a specific graph.

Steps:

- 1) Select a stock from the main panel.
- 2) You will see the graph of the stock in the graph panel
- 3) Click on formula editor-> Following screen will display



- 4) Now to print opening price on output window write following

```
print{open[0]}
```
- 5) Click on “apply formula on stock” icon.
- 6) Click execute and then click on “output window” icon to see the result.

Once you are done with the simple example then let's understand more examples and some complex investment strategies.

IGS FL Programming Elements

Statement

Writing statement in IGS FL is very simple provided you follow the basic rules below:

- 1) Each token must have a space in between
- 2) There must not be any space between { and function name when you call a function
- 3) There must not be any space between an indicator [bracket.
- 4) := is assignment operator and variable are written in left side of assignment operator

- 5) New line character works as an end of statement so all statements must be completed in a single line.
- 6) Variable must start with alphabet
- 7) You can't use any function name or in-built indicator name or user defined function name as a variable. So make sure that the variable is unique
- 8) There are some in-built variables used by language, which you can't use in your language as variable.
- 9) Each function's last statement work as a return value. There is no return statement.
- 10) The delimiter for function call is , (comma) and delimiter for indicator parameter is ;(semi-colon). So if you are calling to get value for 14 day simple moving average for today .The call will be RSI[14;0]. Suppose you want to get value of 2 to power 3 then the function call be pow{2,3}.

Some examples of invalid statements

1) `A := b+5`

Invalid because there must be a space between b+5. So correct statement is `A := b + 5`

2) `A := abs { 10 }`

Invalid because there is space between abs and { so language will not be able to recognize function call. So the correct statement will be `A := abs { 10 }`

3) `A := b + 5 ((6+7)*10)`

Invalid because there is no space between operator opening parentheses. So the correct statement will be `A := b + 5 * ((6 + 7) * 10)`

4) `A := RSI [14,0]`

Invalid because the delimiter for indicator call must be ;(semi colon) not , (comma). So the correct statement will be `A := RSI[14;0]`

5) `A := b +7;`

Invalid because no need for ; at the end of the statement. In IGS FL the end of statement is new line character. `A := b + 7`

6) `A := b +`

7

Invalid because statement is completing in two lines, It must complete in one line. So the correct statement will be `A : b + 7`

Variables and Array.

Variables are the element, which are defined on the left side of any statement.

For example:

`a := 5`

Here is a variable which has a value assigned as 5.

IGS also support array but using array in IGS is slightly different. To use array you need to define your variable like this: **Var{ <expr> }**. First expression is calculated and it works as an variable.

For example

```
Var{ "x" + 1 } := 50
```

The result of this expression is x1 variable will have value 50 after this execution.

Now to use this array either you can directly reference it or use it in a loop where value will be calculated dynamically. Use *echo* function in this case

Example

```
A := 1
While ( a < 10 )
{
    var{ "x" + a } := a + 10
    a := a + 1
}
```

This will create an array of variable x like this: x1, x2, x3, x4, x5, x6, x7, x8 and x9. and the value of all these variable will be 11,12,13,14,15,16,17,18 and 19 respectively.

Now suppose you want to print array x so you need to write following statement

```
A := 1
While ( a < 10 )
{
    print{ echo{ "x" + a } }
    a := a + 1
}
```

This will print value of x1, x2, x3, x4, x5, x6, x7, x8 and x9 on output window.

String Literals

String literals can be defined using `""`. Any string which is inside quotes is called as constant string. All print statements will be converted in upper case.

Example

```
Print{"Hi from IGS FL"} will display HI FROM IGS FL
```

Comments

In IGS FL any statement starting with `@` character is considered as a comment line and IGS FL will not execute that line. This will also be useful users to write their own comments to understand that logic.

Example

@this is a comment

Include statement (#)

The structure of include statement is like this

#<var name>: <expression>

Example exercise:

Create a customized indicator which will give you difference between high value of the day and low value of day and then using that value take a simple moving average of 14 days and display the current value of that simple moving average.

Answer

Here we will create a customized indicator first and then we'll use it in our program.

Step1: Define our customized indicator like this:

```
#MYCUSTIND:HIGH[0] – LOW[0]
```

@ this will give difference between current high and low value as an array called as @MYCUSTIND.

```
Print{ SMAANY[MYCUSTIND,14,0]
```

Here **SMAANY** is a built-in indicator. It takes 3 parameter first is [any inbuilt or customized indicator](#), second parameter is [no of days](#) for which this indicator is applicable and third is the relative day of days before. For current day it should be zero and for earlier days the value should be how many days before. If you want it to be 2 days before then the value will be -2. [SMAANY\[MYCUSTIND,14, -2\]](#)

So in this example you were able to create a simple customized indicator and then were able to use it in our In-built indicator.

Suppose if your indicator is complicated and cannot be written in a single statement then you should first write your indicator in another Formula language file (last statement must return the value) and save that indicator. Now you can call it directly using include statement and can use it in your strategy.

If loop

Syntax:

```
if ( <cond expr> )
```

```
{
```

```
    <statements>
```

```
}
```

Nested if statement

```
if ( <cond expr> )
{
    if ( <cond expr >
    {
        <statements>
    }
}
```

Example 1

Write a program to print if value of a variable **a** is equal to 10.

```
A := 10
If ( a == 10 )
{
    print{ "correct" }
}
```

Example 2

Write a program to print if a is 10 and b is 20.

```
A := 10
B := 20
If ( ( a == 10 ) and ( b == 20 ) )
{
    print{ "correct" }
}
```

Example 3

Write a program to print in case of nested if

```
A := 10
If ( a == 10 )
```

```
{  
  a := a + 5  
  if ( a == 15 )  
  {  
    print{"correct"}  
  }  
}
```

If-else loop

Syntax / Structure

```
if ( <cond expr> )  
{  
  <statements>  
}  
else  
{  
  <statements>  
}
```

Example 1

Print a if a is greater than b else print b

```
if ( a > b )  
{  
  print{ a }  
}  
else  
{  
  print{ b }  
}
```

If-else-if loop

Structure / Syntax

```
if ( <cond expr1> )  
{  
    <statements>  
}  
else if ( <cond expr2> )  
{  
    <statements>  
}  
else  
{  
    <statements>  
}
```

Example 1

```
If ( a > b )  
{  
    print{ a }  
}  
else if ( b > c )  
{  
    print{ b }  
}  
else  
{  
    print{c}  
}
```

while loop

Structure / Syntax

```
while ( <cond expr> )
{
    <statements>
}
```

while loop will continue to execute till the condition is true.

Example 1

Print 1 to 10.

```
A := 1
While ( a < 10 )
{
    print{ a }
    a := a + 1
}
```

In Built Indicators

IGS FL uses a lot of built-in indicators or arrays, which can be used to create a customized indicator or any other strategy. Let's have a look at built-in indicator and understand can you reference them to get any backward value of the indicator. The basic indicators / arrays are:

Basic Indicator Name	Explanation
Date	Date or Time on the bar
Open	Opening price on thet bar
High	High price of the bar
Low	Low price of the bar
Close	Closing price of the bar
Volume	Volume of the security on that day

Let's say you want opening price of a stock 2 days back, then you need to use an open array like this

`Open[-2]` it will give you 2 days old value

Like this you can go backward (NOT forward) to get any data. When you apply your formula on a graph then the formula runs on each bar of the graph. So like `Open[0]` will automatically be current price of that bar on a graph.

For example, open array has value like this

`Open = 5,10,15,10,20`

So if you write statement like `println{open[0] + " and " + open[-1]}` and apply this strategy on a graph. The output window will display

5 and 0

Note: 0 because the `open[-1]` does not exist and for the value that does not exist value it will print zero

10 and 5

15 and 10

10 and 15

20 and 10

Now If you apply the same formula on stock screening then result will be different. In this situation only current value is taken into consideration and formula will run only on current value so output window will display

20 and 10 (current value and one day before value)

Moving Averages

Simple Moving Average (SMA)

This is the average stock price over a certain period of time. Keep in mind that equal weighting is given to each daily price.

Syntax

`SMA[noofdays;daysbefore]`

Example

`SMA[14;0]`

This will give average price of last 14 days including today.

Exponential Moving Average (EMA)

A type of moving average that is similar to a simple moving average, except that more weight is given to the latest data. The exponential moving average is also known as "exponentially weighted moving average".

Syntax

EMA[noofdays;daysbefore]

Example

EMA[14;0]

This will give EMA of last 14 days including today.

Weighted Moving Average (WMA)

A weighted average is any average that has multiplying factors to give different weights to different data points

Syntax

WMA[noofdays;daysbefore]

Example

WMA[14;0]

This will give WMA of last 14 days including today.

Mahesh Moving Average (MMA)

It is a moving average invented by *iGuideStocks* Team and it is based on both volume and price. Read article at www.iguidestocks.com to know more about this moving average.

Syntax

MMA[noofdays;daysbefore]

MMA[14;0]

This will give MMA of last 14 days including today.

Moving Average Convergence/Divergence Indicator(MACD)

MACDLINE:

It shows the difference between a fast and slow exponential moving average (EMA) of closing prices. The most popular MACD is the difference between a security's 12-days EMA and 26-day EMA.

Syntax

MACDLINE[lownoofdays;highnoofdays;signalline;daysbefore]

Example

MACDLINE[12,26,9,0]

This will return difference between 12-day EMA and 26-day EMA where signal is 9 day for current day.

Example

MACDLINE[20,40,10,-5]

This will return difference between 20-day EMA and 40-day EMA where signal is 10 day for 5 days before current day.

MACDSIGNAL :

It is Exponential Moving Averages (EMAs) of MACD. The most popular MACDSIGNAL is 9-day EMA.

Syntax:

MACDSIGNAL[lownoofdays,highnoofdays,signalline,daysbefore]

Example

MACDSIGNAL[12,26,9,0]

This will return 9 day EMA of difference between 12-day EMA and 26-day EMA for current day.

Example

MACDSIGNAL[20,40,10,-5]

This will return 10 day EMA of difference between 20-day EMA and 40-day EMA for 5 days before current day.

Willam %R

This is used to find out overbought and oversold level. The scale ranges from 0 to -100 with readings from 0 to -20 considered overbought, and readings from -80 to -100 considered oversold.

Syntax

`WILPER[noofdays;daysbefore]`

Example

`WILPER[14;0]`

This will give 14 days Willam %R for current day.

Bollinger Bands

Bollinger Bands allows user to compare volatility and relative price levels over a period of time. The indicator consists of three bands designed to encompass the majority of a security's price action.

1. A simple moving average in the middle
2. An upper band (SMA plus 2 standard deviations)
3. A lower band (SMA minus 2 standard deviations)

UBBAND

Upper band of Bollinger band

Syntax

`UBBAND[noofdays;standard deviation;daysbefore]`

Example

`UBBAND[20;2;0]` will return upper band using 20 days and 2 as standard deviation for current day.

LBBAND

Lower band of Bollinger band

Syntax

LBBAND[noofdays;standard deviation;daysbefore]

Example

LBBAND[20;2;0] will return upper band using 20 days and 2 as standard deviation for current day.

HIGHINDAYS

The highest value in no of days.

Syntax :

HIGHINDAYS[noofdays;daysbefore]

Example

HIGHINDAYS [12;0]

This will return highest value in specified last 12 days

LOWINDAYS

The lowest value in no of days.

Syntax:

LOWINDAYS[noofdays;daysbefore]

Example

LOWINDAYS [12;0]

This will return lowest value in last 12 days

CCI (Commodity Channel Index)

This indicator was designed to identify cyclical turns in commodities. The assumption behind the indicator is that commodities (or stocks or bonds) move in cycles, with highs and lows coming at periodic intervals.

Syntax:

CCI[noofdays;daysbefore]

This will return cci value for specified no of days

CMF (Chaikin Money Flow)

The *Chaikin Money Flow* oscillator is calculated from the daily readings of the Accumulation/Distribution Line. The basic premise behind the Accumulation Distribution Line is that the degree of buying or selling pressure can be determined by the location of the Close relative to the High and Low for the corresponding period (Closing Location Value).

Syntax :

[CMF\[noofdays;daysbefore\]](#)

This will return CMF value for specified no of days

Aroon Oscillator

Aroon is an indicator system that can be used to determine whether a stock is trending or not and how strong the trend is.

The Aroon indicator system consists of two lines, 'Aroon(up)' and 'Aroon(down)'. It takes a single parameter which is the number of time periods to use in the calculation. Aroon(up) is the amount of time (on a percentage basis) that has elapsed between the start of the time period and the point at which the highest price during that time period occurred. If the stock closes at a new high for the given period, Aroon(up) will be +100. For each subsequent period that passes without another new high, Aroon(up) moves down by an amount equal to $(1 / \# \text{ of periods}) \times 100$

AROOPUP

Syntax : [AROOPUP\[noofdays;daysbefore\]](#)

AROOPDOWN

Syntax : [AROOPdown\[noofdays;daysbefore\]](#)

AROOPOSI

Syntax : [AROOPOSI\[noofdays;daysbefore\]](#)

ATR (Average True Range)

The Average True Range (ATR) indicator measures a security's volatility. As such, the indicator does not provide an indication of price direction or duration, simply the degree of price movement or volatility.

AVGTRUERANGE

Syntax:

[AVGTRUERANGE\[noofdays;daysbefore\]](#)

WILLAMACDIS

Syntax:

[WILLIMAMACDIS\[noofdays;daysbefore\]](#)

ACCDIS

Syntax:

[ACCDIS\[noofdays;daysbefore\]](#)

Here is complete list of all data elements:

Name	Syntax	Explanation
Open	Open[daysbefore]	
Close	Close[daysbefore]	
High	High[daysbefore]	
Low	Low[daysbefore]	
Volume	Volume[daysbefore]	
Date	Date[daysbefore]	
SMA	SMA[noofday;daysbefore]	Simple Moving Average
WMA	WMA[noofdays;daysbefore]	Weighted Moving Average
EMA	EMA[noofdays;daysbefore]	Exponential Moving Average
MMA	MMA[noofdays;daysbefore]	Mahesh Moving Average
RSI	RSI[noofdays;daysbefore]	
MACDLINE	MACDLINE[lowday;highdays;sigday;daysbefore]	
MACDSIGNAL	MACDSIGNAL[lowday;highdays;sigday;daysbefore]	

MACDDIFF	MACDDIFF[lowday;highdays;sigday; daysbefore]	
LOWINDAYS	LOWINDAYS[noofdays;daysbefore]	Lowest Value in days
HIGHINDAYS	HighIndays[noofdays;daysbefor	Highest Value in days
WILPER	WILPER[noofdays;daysbefore]	Willam %R
CCI	CCI[noofdays;daysbefore]	
CMF	CMF[noofdays;daysbefore]	
UBBAND	UBBAND[noofdays;stand. Dev.;daysdefore]	
LBBAND	LBBAND[noofdays;stand. Dev.;daysbefore]	
AROOPUP	AROOPUP[noofdays;daybefore]	
AROOPDOWN	AROOPDOWN[noofdays;daybefor e]	
AROOPOSI	AROOPOSI[noofdays;daybefore]	
AVGTRUERANGLE	AVGTRUERANGLE	
<i>WILLAMACCDIS</i>	WILLAMACCDIS[noofdays;daybefor e]	
<i>ACCDIS</i>	ACCDIS[noofdays;daybefore]	

Operators

An "operator" is a function which acts on functions to produce other functions. Mathematical operators:

+ , - , * , / , >= , > , < , <= == , := etc.

Following operators are supported by IGS Formula Language

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
And	And operation
Or	Or operation
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
<>	Not equal
True	True operation
False	False operation
(Open bracket
)	Close bracket

Precedence of operation is as below

*, / highest

+, - next

and, or, <, >, <>, ==, <=, >= next

Example

$$1 + 2 * 3 = 7$$

As first operation will be $2 * 3$ and then 1 will be added to it.

You can also use addition operation on string.

In Built Function:

Definition of inbuilt functions

All inbuilt function call has following structure

<Function name>{ parameters }

Note: there should not be a space between function name and { otherwise parser will not recognize it.

BUY:

This function is used to display buy signal (white arrow) on chart with comment.

Syntax `buy{ expression or a sting }`

Example: `buy{ "" }` will show only a white arrow on chart

`Buy{ "Buy Now" }` will show a white arrow and Buy Now written at the end of arrow

`Buy{ 1 + 5 }` will show a white arrow and 6 written at end of arrow

SELL:

This function is used to display sell signal (red arrow) on chart with comment.

Syntax `sell{ expression or a sting }`

Example: `sell{ "" }` will show only a red arrow on chart

`sell{ "s1" }` will show a red arrow and s1 written at end of arrow

ABS:

This function is used to convert any -ve value to +ve.

Syntax: `abs{ expression }`

Takes single parameter or expression which must be a numeric value

Example: `abs{ -10 }` will return 10

`abs{ 100.35 }` will return 100.35

INVERT:

This function will convert *true* to *false* and *false* to *true*.

Syntax: `INVERT{ expression }`

Takes single parameter or expression which must be operator true or false

Example: `invert{ true }` will return false

`invert{ false }` will return true

`invert{ 10 > 5 }` will return false as `10 > 5` condition will return true and invert will convert true to false

PERCENT:

This function will give percentage value of parameter.

Syntax: PERCENT{ expression }

Takes single parameter or expression which must be a numeric value

Example: percent{ 1 } will return 100

percent{ 1 + 2 } will return 300

percent { -10 } will return false as -1000.

ONEOF:

This function will return true if particular string is part of other string

Syntax: ONEOF{ symbolstring, mainstring}

This will check mainstring and if symbolstring is found in it then it will return true else will return false

Example: oneof{ "abc","abc:gm:infy" } will return true as "abc:gm:infy" string has "abc" string

PRINT:

This function will print value on the output window without a newline character

Syntax print{ <expr>}

Example : print{ "IGS" } will print IGS on output window

Print{ 1 + 5 } will print 6 on output window

println:

This function will print value on output window with a newline character

Syntax: println{ <expr>}

Example: println{ "IGS" } will print IGS on output window with newline character

Println{ 1 + 5 } will print 6 on output window with newline character

ALERT:

This function will pop an alert message on the position where alert condition is satisfied on the graph.

Syntax: `alert{ <expr> or string }`

Example:

```
if ( open[0] > close[0] )
{
    alert{ "Open is greater than close" }
}
```

In this example where ever open is greater than close on a specific chart an alert message window will pop up. It is very useful for intraday trading where graph is updating in say every 10-15 seconds. Alert will pop up whenever user's buy / sell strategy / condition is met.

SQRT:

This function will return Square root of the parameter. Parameter must be a +ve numeric value

Syntax: `sqrt{ <expr> }`

Example: `sqrt{ 4 }` will return 2 .

POW:

This function will return raise to the power value of the parameter.

Syntax: `pow{ <expr>,<expr> }`

Example: `pow{2,3 }` will return 8

Drawing functions in IGS:

IGS supports major drawing component, User can plot graph, draw circle, rectangle, line or text at a particular point using these drawing functions.

Plot function

Using IGS FL user can plot graph on main graph panel. You can plot up to 3 graphs using plot1, plot2, plot3 function. The structure of the function is like this

Syntax: `Plot1{value,color} or plot2{value,color} or plot3{value,color}`

This will plot graph at specified value in specified color.

Examples

`Plot1{high[0],red}` will plot a graph in red color using high value .

Circle

Syntax: `circle{value,color,text,size}`

This function will draw a filled circle at specified point of value parameter. You can also define color, text to be displayed and size of circle.

Example 1

`Circle{low[0],red,"Y",5}` → this will draw a circle at low value of that bar using color red. The size of circle will be 5 and it will also show text "Y".

Rectangle

Syntax `rectangle{value,color,text,size}`

This function will draw a filled rectangle at specified point of value parameter. You can also define color, text to be displayed and size of rectangle.

Example 1

`rectangle{low[0],red,"Y",5}` → this will draw a rectangle at low value of that bar using color red. The size of rectangle will be 5 and it will also show text "Y".

Text

Syntax `text{value,color,text,size}`

This function will draw text at specified point which is defined by value parameter. The color of rectangle will be defined by *color* parameter. The *size* parameter will do nothing but is required.

Example 1

`text{low[0],red,"Y",5}` → this will draw text "Y" at low value of that bar using color red .

Candle Stick formula

Barlength → will return length of the bar (high –low)

Candlelength → will return candle's length `abs{open-low}`

Greencandle → will return true if close is greater than open

Redcandle → **will** return true if close is less than open

Doji →

Hanging man

Hammer

InvertedHammer

ShootingStar

BlackSpinningTop

WhiteSpinningTop

BearishEngulfing

BullishEngulfing

ThreeOutsideUpPattern

BullishHarami

ThreeInsideUpPattern

PiercingLine

BearishHarami

Other internal variable

buy: this is internal variable created by function `buy{""}`.

Sell: this is internal variable created by function `sell{""}`.

Alert: this is internal variable created by function `alert{""}`.

Pattern: this variable is used to show which pattern is occurring on stock screening screen. Whenever you need display a pattern on the screen use this variable

Buytype: this variable is used show buy type. The values for this variable can be `strongbuy`, `strongsell`, `buy`, `sell` and `hold`. Different color is assigned for each buytype

`Strongbuy` green

`Strongsell` red

`Sell` pink

`Buy` grey

List of all in-built function in tabular form for quick reference

Name	Syntax	Explanation
Buy	Buy{string text}	Displays buy arrow with text
Sell	Sell{string text}	Displays sell arrow with text

Alert	Alert{string text}	Will pop up a message window.
Echo	Echo{string expr}	This will first calculate the expression and will return the value
Var	Var{string expr}	Expression is calculated and created as a variable, used to create array in IGS FL
Abs	Abs{float para}	It will return positive value of the parameter
Invert	Inver{Boolean cond}	Will invert the the condition. Will convert true to false and false to true.
Percent	Percent{float para}	Will multiple parameter by 100 and return it
Oneof	Oneof{string partstr, string fullstr}	This function will return true if particular string is part of other string
Print	Print{expr}	This function will print value on the output window without a newline character
Println	Println{expr}	This function will print value on the output window with a newline character
Sqrt	Sqrt{float val}	Will return square root of the variable
Pow	Pow{float val1,float val2}	Will return power of value2 for base val1. Example pow{2,3} will return 8
Plot, plot2, plot3	Plot1{expr exp,color color} Plot2{expr exp,color color} Plot3{expr exp,color color}	Using IGS FL user can plot graph on main graph panel. You can plot up to 3 graphs using plot1, plot2, plot3 function.
Circle	Circle{expr ,color,text,size}	This function will draw a filled circle at specified point of value parameter.

Rectangle	Rectangle{ expr,color,text,size }	This function will draw a filled rectangle at specified point of value parameter
Text	Text{ expr,color,text,size }	This function will put a text at specified point of value parameter
Intvalue	Intvalue{float expr}	Will return integer value of expression

Sample Exercise

1. Find out major candle stick pattern and then display an alert whenever a specific pattern occurs

Solution:

Let's take some of major candle stick patterns and start.

1. **Green candle:** When close is greater than open
2. **Red candle:** When open is greater than close
3. **DOJI** : When open and close price are nearly same (here say difference between open and close is less than 0.5 %)
4. **Hammer:** Hammer candlesticks form when a security moves significantly lower after the open, but rallies to close well above the intraday low. The resulting candlestick looks like a square lollipop with a long stick. If this candlestick forms during an advance, then it is called a Hanging Man.
5. **Hanging Man:** Candlesticks form when a security moves significantly lower after the open, but rallies to close well above the intraday low. The resulting candlestick looks like a square lollipop with a long stick. If this candlestick forms during a decline, then it is called a Hammer.

Now let's give this formula new name as candlestick. And start working on it.

The code will look like this

```
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]
OGTC := ( O > C )
```

```

PATTERN := " "
$ if difference between open and close is less than 1/10 th of total candle length it is $dozi

IF ( abs{ O - C } <= ( ( H - L ) * 0.1 ) )
{
    PATTERN := PATTERN + " NearDoji, "
}

$if open is greather than close it is black candle else white candle
IF ( OGTC )
{
    $if open is greather than close it is black candle else white candle

    PATTERN := PATTERN + " BlackCandle, "
}
ELSE
{
    PATTERN := PATTERN + " WHITECANDLE, "
}

IF ( (( H - L ) > 4 * ( O - C ) ) AND ( ( C - L ) / ( H - L ) > 0.75 ) AND ( ( O - L ) / ( H - L ) > 0.75 ) )
{
    Pattern := "HangingMan, "
}
IF ( (( H - L ) > 3 * ( O - C ) ) AND ( ( C - L ) / ( H - L ) > 0.6 ) AND ( ( O - L ) / ( H - L ) > 0.6 ) )
{
    PATTERN := "Hammer, "
}
}
ALERT{ PATTERN }

```

Example 2

Find out buy and sell signal using MACD

Solution:

```

A := MACDLINE[12;26;9;0] > MACDSIGNAL[12;26;9;0] AND MACDLINE[12;26;9;-1] <
MACDSIGNAL[12;26;9;-1]

IF ( A )
{
    BUY{""}

    BUYTYPE := "STRONGBUY"
}

B := MACDLINE[12;26;9;0] < MACDSIGNAL[12;26;9;0] AND MACDLINE[12;26;9;-1] >
MACDSIGNAL[12;26;9;-1]

```

IF (B)

{

SELL{" "}

BUYTYPE := "STRONGSELL"

}

Example 3

Write open, close price on output window

Solution:

```
println{ open[0] + " and " + close[0] }
```