

Guía de programación

ProBuilder



SUMARIO



Presentación de ProBuilder



Capítulo I: Principios fundamentales

Utilizar ProBuilder.....	2
Tutorial rápido para la creación de un indicador.....	2
Atajos de teclado en la ventana de programación.....	5
Aspectos específicos de la programación en ProBuilder.....	6
Las constantes financieras ProBuilder.....	7
Las constantes de precio y volumen adaptadas a la unidad de tiempo del gráfico.....	7
Constantes diarias del precio.....	8
Constantes de tiempo.....	8
Constantes de precio.....	12
Constante indefinida.....	12
Uso de indicadores ya existentes.....	12
Optimización de variables.....	13



Capítulo II: Funciones e instrucciones ProBuilder

Estructuras de control.....	15
Instrucción condicional IF.....	15
Una condición, un resultado (IF THEN ENDIF).....	15
Una condición, dos resultados (IF THEN ELSE ENDIF).....	15
Condiciones imbricadas.....	15
Condiciones Múltiples (IF THEN ELSE ELSIF ENDIF).....	16
Bucle iterativo FOR.....	17
Avance creciente (FOR, TO, DO, NEXT).....	17
Avance decreciente (FOR, DOWNT, DO, NEXT).....	18
Bucle condicional WHILE.....	19
BREAK.....	20
Con WHILE.....	20
Con FOR.....	20
CONTINUE.....	21
Con WHILE.....	21
Con FOR.....	21
ONCE.....	22
Funciones matemáticas.....	23
Funciones comunes unarias y binarias.....	23
Operadores matemáticos comunes.....	23
Funciones de comparaciones gráficas.....	23
Funciones sumatorias.....	24
Funciones estadísticas.....	24
Operadores lógicos.....	24
Instrucciones ProBuilder.....	24
RETURN.....	25
REM o //.....	25
CustomClose.....	25
CALL.....	26
AS.....	26
COLOURED.....	26



Capítulo III: Aplicaciones prácticas

➤ Creación de un indicador binario o ternario: lógica y método.....	28
➤ Crear indicadores STOP: siga sus posiciones en tiempo real.....	29
■ STOP profit estático.....	30
■ STOP loss estático.....	30
■ STOP de inactividad.....	31
■ Trailing STOP.....	32



Capítulo IV: Ejercicios

➤ Configuraciones de velas japonesas.....	33
➤ Indicadores.....	34



Glosario



Presentación de ProBuilder

ProBuilder es el lenguaje de programación de ProRealTime. Le permite crear indicadores personalizados de análisis técnico, estrategias y sistemas de trading (ProBackTest), o escáneres de mercados (ProScreener). ProBackTest y ProScreener tienen manuales de ayuda propios a causa de sus propias especificidades de programación.

ProBuilder es un lenguaje de tipo BASIC, muy simple y a la vez exhaustivo en cuanto a las posibilidades de programación que ofrece.

Con él podrá construir sus propios programas que utilizarán los precios de cualquier valor que forme parte de la cobertura de mercados de ProRealTime, a partir de los siguientes elementos de base:

- el precio de apertura de cada vela: **Open**
- el precio de cierre de cada vela: **Close**
- el máximo de cada vela: **High**
- el mínimo de cada vela: **Low**
- la cantidad de títulos negociados: **Volume**

Las velas, candlesticks o barras, son representaciones gráficas estandarizadas de cotizaciones recibidas en tiempo real. Por supuesto, ProRealTime le permite personalizar el estilo de representación gráfica más allá de las opciones descritas, incluyendo otras vistas como Renko, Kagi, Heikin-Ashi.

ProBuilder evalúa los datos de cada vela de precios por orden cronológico de la más antigua a la más reciente, ejecutando la fórmula programada en el lenguaje para calcular el valor de los indicadores en cada vela.

Los indicadores creados mediante ProBuilder pueden mostrarse superpuestos al gráfico de precios o en un gráfico individual adyacente, en función del tipo de escala empleado.

En este manual se familiarizará gradualmente con los comandos de programación de ProBuilder con una visión teórica clara acompañada de ejemplos ilustrativos.

Al final del manual encontrará un índice que le ofrece un listado global de los comandos de ProBuilder, indicadores ya codificados y otras funciones complementarias a lo que haya aprendido durante la lectura.

Aquellos lectores más acostumbrados a la programación pueden pasar directamente a la lectura del capítulo II, o consultar el índice para hallar rápidamente la explicación relativa a la función concreta que les pueda interesar.

Para quien no esté familiarizado con la programación, sugerimos visualizar el vídeo titulado '[Programe indicadores simples o dinámicos](#)' y leer íntegramente este manual, que hemos diseñado para permitirle programar lo más rápidamente posible. ¡ Buena lectura !

Capítulo I: Principios fundamentales

Utilizar ProBuilder

Tutorial rápido para la creación de un indicador

El acceso a la zona de programación de un indicador se realiza a través del botón 'Indicador/Backtest' situado hacia la esquina superior derecha de cualquier ventana de gráfico de su plataforma ProRealTime.



Desde ahí podrá acceder a la ventana de gestión de indicadores, donde podrá:

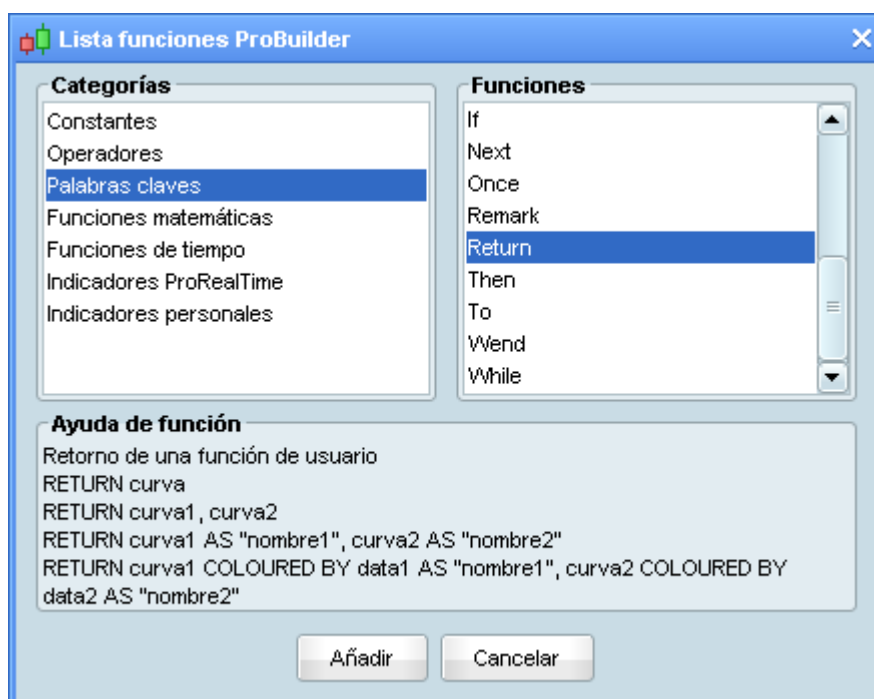
- visualizar un indicador predefinido de cada vela
- crear un indicador personalizado, que podrá aplicar a cualquier valor.

En este último caso deberá pulsar el botón 'Crear indicador' para acceder a la ventana de programación. Esta ventana le ofrece las siguientes posibilidades:

- programar directamente un indicador en la zona de texto reservada al código
- utilizar el asistente 'Insertar función' para abrir una nueva ventana con una biblioteca de funciones clasificadas en siete categorías, para utilizarlas directamente en su codificación.



Veamos el ejemplo del primer elemento típico de los indicadores ProBuilder: la función '**RETURN**'. Está disponible en la sección '[Instrucciones ProBuilder](#)', tal y como se muestra en la imagen a continuación.



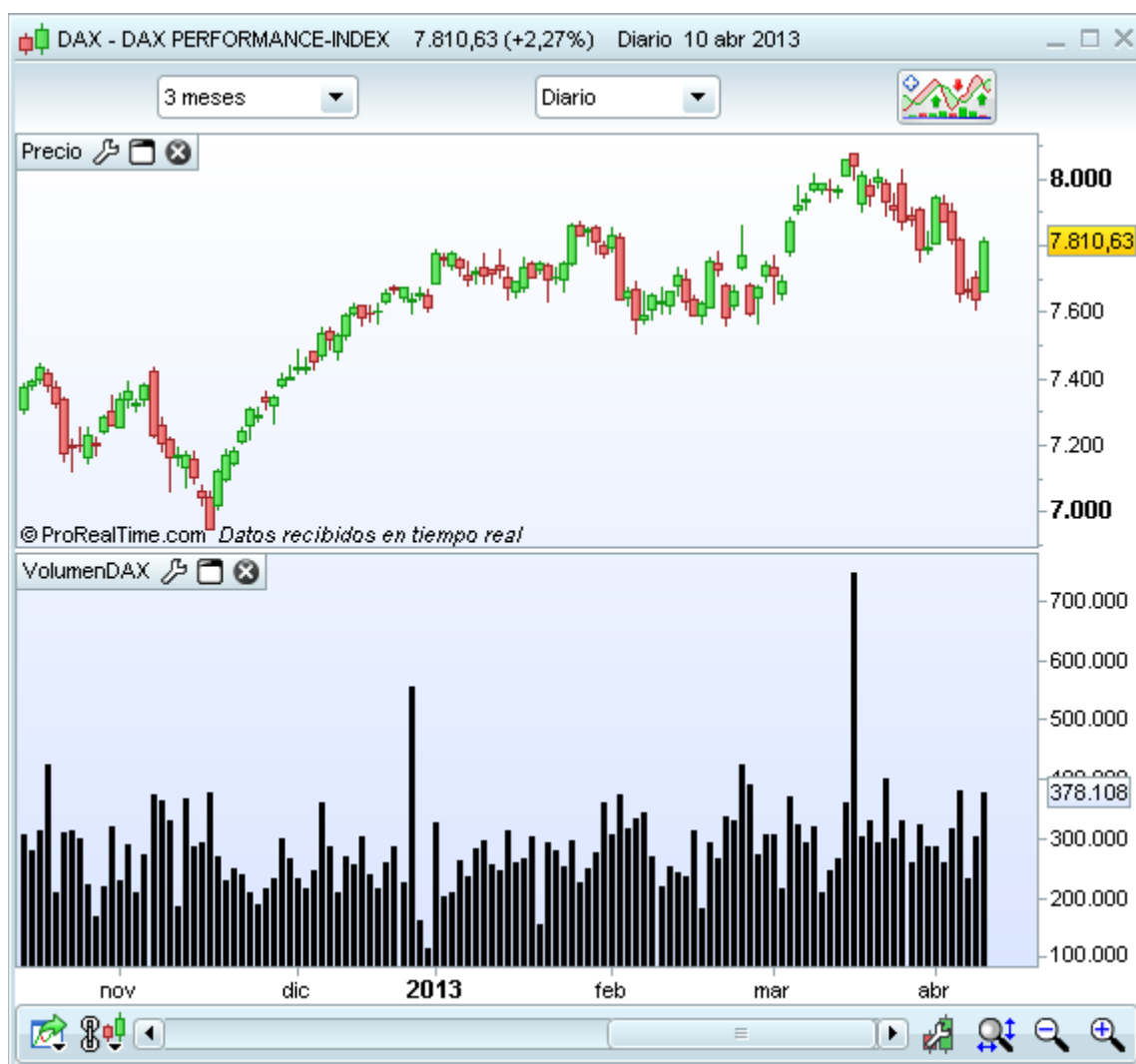
Para añadir el comando en la zona de programación, bastará con seleccionar '**RETURN**' y pulsar 'Añadir'.

★ **RETURN** muestra el resultado del cálculo de su indicador

Supongamos que queremos programar un indicador que muestre el Volumen. Si ya hemos insertado el comando **RETURN**, bastará con ir nuevamente a 'Insertar función', pulsar en 'Constantes' (en la sección 'Categorías', y finalmente, pulsar en '**Volume**' a la derecha, en la sección 'Funciones disponibles' y 'Añadir'.



Antes de pulsar el botón 'Validar programa', es necesario dar un nombre al indicador en el campo previsto a este efecto en la zona superior de la ventana. En este ejemplo, lo hemos llamado 'VolumenDAX'. Tras esto, pulse 'Validar programa' y el indicador aparecerá en la ventana de gráfico.



■ **Atajos de teclado en la ventana de programación**

La ventana de creación de sistemas de trading tiene varias funciones a las que se puede acceder mediante atajos de teclado en la versión 10 de ProRealTime:

- Seleccionarlo todo (Ctrl + A): Selecciona la integralidad del texto en la ventana de programación
- Copiar (Ctrl + C): Copia el texto seleccionado
- Pegar (Ctrl + X): Pega el texto copiado
- Deshacer (Ctrl + Z): Deshace la última acción realizada en la ventana de programación
- Rehacer (Ctrl + Y): Rehace la última acción realizada en la ventana de programación
- Buscar / Remplazar (Ctrl + F): Busca un texto preciso en la ventana de programación / reemplaza un texto contenido en la ventana de programación (esta función es sensible a la diferencia entre minúsculas y mayúsculas)
- Comentar / Descomentar (Ctrl + R): Comenta el código seleccionado / Descomenta el código seleccionado (el código comentado será introducido por '/' o 'REM' y se coloreará en gris. Esto comentario no será tomado en cuenta en la ejecución del código).

En Mac es posible utilizar estos mismos atajos de teclado utilizando la tecla 'Manzana' en lugar de 'Ctrl'. Casi todas estas funciones son accesibles pulsando en el botón derecho de su ratón en la zona de programación de la ventana de creación del sistema de trading.

📌 Aspectos específicos de la programación en ProBuilder

Peculiaridades

A través del lenguaje ProBuilder puede manipular gran cantidad de comandos muy conocidos, amén de otras herramientas más complejas y específicas del análisis técnico. Podrá así programar todo tipo de indicadores, del más sencillo al más sofisticado.

Estos son los principios fundamentales del lenguaje ProBuilder:

- **no es necesario declarar las variables**
- **no es necesario teclear las variables**
- **no se discriminan las mayúsculas y minúsculas** (aunque sí hay una particularidad que se describe más abajo)
- se emplea el mismo símbolo para representar los conceptos de **afectación e igualdad matemática**

Veamos cada punto en detalle:

- Declarar una variable X equivale a indicar su existencia de modo explícito. En ProBuilder puede utilizar directamente X sin tener que definir previamente su existencia.

Veamos un ejemplo para ilustrar la diferencia entre la programación clásica y ProBuilder:

Programación clásica: Sea la variable X. Se atribuye a X el valor 5

Programación ProBuilder: Se atribuye a X el valor 5 (así, X es implícitamente existente y vale 5)

En ProBuilder bastará con escribir: X=5

- Teclear una variable equivale a definir la naturaleza de la variable: puede ser un entero natural (p.ej.: 3; 8; 21; 643; ...), un entero relativo (p.ej.: 3; 632; -37; ...), un decimal (p.ej.: 1,76453534535...), un booleano (VRAI, FAUX),... ?

- En ProBuilder, puede escribir sus comandos tanto en mayúsculas como en minúsculas. Por ejemplo, el conjunto de los comandos IF / THEN / ELSE / ENDIF podría escribirse iF / tHeN / ELse / eNdIF.

Excepción: Cuando utilice una variable, será necesario respetar las mayúsculas y minúsculas en el nombre definido. Por ejemplo, si escribe: 'vARiABLE' y desea emplear dicha variable en un código, será necesario teclear 'vARiABLE' para referirse a dicha variable.

- Atribuir un valor a una variable equivale a darle un valor. Para comprender mejor este principio, podemos considerar una variable como una caja vacía en la que metemos algo. El esquema siguiente le ilustra este principio con el valor Volumen atribuido a la variable X:

X ← Volume

Observe que esta frase se leería de derecha a izquierda: el volumen se atribuye a X.

Para programar esto en lenguaje ProBuilder, bastará con sustituir la flecha por un signo '=':

X = Volume

Este símbolo se utiliza de dos maneras distintas:

- para atribuir una variable (como acabamos de ver en el ejemplo)
- como operador matemático binario (1+ 1 = 2 equivale a 2 = 1 + 1).

Las constantes financieras ProBuilder

Antes de comenzar a codificar sus indicadores personales, vamos a revisar los elementos básicos con los que construirá su código (precios de apertura y cierre, volumen, etc.)

Estas constantes son los 'fundamentos' del análisis técnico.

Mediante la combinación de distintas constantes, será posible resaltar ciertos aspectos de la información proporcionada por los mercados financieros. Encontramos 5 categorías distintas:

Las constantes de precio y volumen adaptadas a la unidad de tiempo del gráfico

Son las constantes 'clásicas', las más frecuentes. Por defecto, indican los valores de la vela actual (independientemente de la unidad de tiempo del gráfico), y se presentan de esta manera::

- **Open** : nivel de apertura de la vela actual
- **High** : nivel máximo de la vela actual
- **Low** : nivel mínimo de la vela actual
- **Close** : nivel de cierre de la vela actual
- **Volume** : cantidad de valores o de contratos negociados en la vela actual

Ejemplo: Rango de la vela actual

```
a = High
b = Low
MiRango = a - b
RETURN MiRango
```

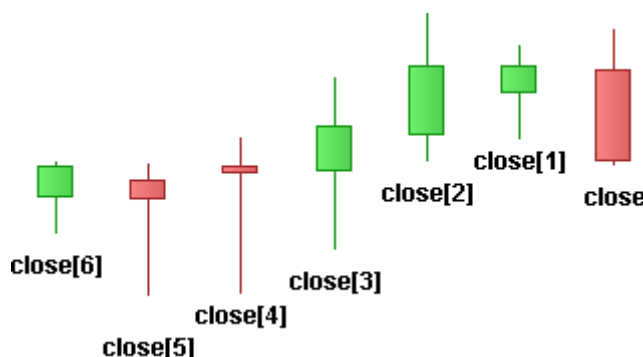
Para llamar a los valores de las velas precedentes, basta con añadir entre corchetes [] la cifra referente a la vela considerada (número de velas a partir de la actual).

Tomemos p.ej. la constante precio al cierre. La llamada del valor se hará de la siguiente manera:

Valor del cierre de la vela actual: Close

Valor del cierre de la vela que precede a la actual: Close[1]

Valor del cierre de la enésima vela que precede a la actual: Close [n]



Esta regla sirve para cualquier constante. Por ejemplo, el precio de apertura de la segunda vela que precede a la vela actual, se utilizará la expresión: Open[2].

El valor que aparezca dependerá del período mostrado en el gráfico.

■ Constantes diarias del precio

Contrariamente a las constantes adaptadas a la unidad de tiempo del gráfico, las constantes diarias del precio se refieren a los valores del día, independientemente del período mostrado en el gráfico.

Otra diferencia respecto a las constantes adaptadas a la unidad de tiempo es que las constantes diarias utilizan paréntesis () para obtener el valor sobre las velas anteriores.

- **DOpen(n)** : precio de apertura del enésimo día anterior al de la vela actual
- **DHigh(n)** : precio máximo del enésimo día anterior al de la vela actual
- **DLow(n)** : precio mínimo del enésimo día anterior al de la vela actual
- **DClose(n)** : precio de cierre del enésimo día anterior al de la vela actual

Observación: si 'n' es igual a 0, se recupera el valor del día actual. Veámoslo con un ejemplo, mostrando el valor del día a continuación.

Ejemplo: Rango diario

```
a = DHigh(0)
b = DLow(0)
MiRango = a - b
RETURN MiRango
```



Para las constantes adaptadas a la unidad de tiempo se utiliza el corchete: Close[3]

Para las constantes diarias se emplean los paréntesis: Dclose(3)

■ Constantes de tiempo

A veces sucede que la constante tiempo no recibe mucha atención en el análisis técnico. Sin embargo, los traders conocen bien la importancia de los momentos claves del día, o de ciertas fechas del año. Para poder limitar el análisis de un indicador a momentos específicos, existen estas constantes:

- **Date** : Fecha en formato YYYYMMJJ, que indica la fecha de cierre de cada vela

Las constantes de tiempo se consideran enteros en ProBuilder. La constante Date será así presentada como un único número de 8 dígitos.

Veamos este Código

```
RETURN Date
```

Supongamos que hoy estamos a 4 de Julio del 2008. El indicador conectará todas las fechas de cierre de las velas, enviando el resultado 20080704.

Para leer una fecha, bastará con seguir esta regla:

20080704 = 2008 años 07 meses y 04 días.

Obviamente, en una fecha con formato YYYYMMJJ, MM no puede ser superior a 12 y JJ, a la cifra 31.

- **Time** : HoraMinutoSegundo en formato HHMMSS. Indica la hora de cierre de cada vela

En el caso del Código

`RETURN Time`

Obtendremos una curva que conecta las horas de cierre de cada vela:



Para leer una hora, bastará con seguir esta regla:

160000 = 16 horas 00 minutos y 00 segundos.

Obviamente, al escribir una hora en formato HHMMSS, HH no podrá ser superior al valor 23, MM no superará el valor 59 y SS tampoco superará el valor 59.

Es posible combinar **Time** y **Date** en un mismo indicador para restringir el resultado a un momento concreto. En el ejemplo mostrado a continuación, vamos a limitar nuestro indicador al primero de Octubre del 2008, a las 9h00 y 1s.

```
a = (Date = 20081001)
b = (Time = 090001)
RETURN (a AND b)
```

Las demás constantes tiempo funcionan de modo análogo:

- **Minute** : Minuto de cierre de cada vela (entre 0 y 59)
- **Hour** : Hora de cierre de cada vela (entre 0 y 23)
- **Day** : Día del mes de cierre de cada vela (entre 1 y 28 o 29 o 30 o 31)
- **Month** : Mes de cierre de cada vela (entre 1 y 12)
- **Year** : Año de cierre de cada vela
- **DayOfWeek** : Día de la semana en el cierre de cada vela (sólo toma en cuenta los días laborales, siendo 1=lunes, 2=martes, 3=miércoles, 4=jueves, 5=viernes)

Ejemplo:

```
a = (Hour > 170000)
b = (Day = 30)
RETURN (a AND b)
```

- **CurrentHour** : Hora actual (del mercado)
- **CurrentMinute** : Minuto actual (del mercado)
- **CurrentMonth** : Mes actual (del mercado)
- **CurrentSecond** : Segundo actual (del mercado)
- **CurrentTime** : HoraMinutoSegundo actual (del mercado)
- **CurrentYear** : Año actual (del mercado)
- **CurrentDayOfWeek** : Día de la semana actual (huso horario del mercado)

La diferencia entre las constantes 'Current' recién mencionadas y las que no tienen 'Current' y presentadas previamente, reside precisamente en la noción 'Actual'.

La imagen mostrada a continuación resalta esta diferencia aplicada a las constantes **CurrentTime** y **Time**. Para simplificar, las constantes con 'Current' no toman en consideración el eje del tiempo, y toman en cuenta únicamente el valor que se muestra en el cuadro blanco.



Time indica la hora de cierre de cada vela

CurrentTime indica la hora actual en el mercado

Si desea ajustar sus indicadores respecto a un contador (número de días transcurridos, cantidad de velas, etc...), las constantes **Days**, **BarIndex** y **IntradayBarIndex** cumplen con este cometido.

- **Days** : Contador de días desde 1900

Esta constante es útil para conocer la cantidad de días transcurridos, concretamente al trabajar en vistas cuantitativas como por ejemplo (x)ticks o (x)volúmenes.

Veamos un ejemplo en el que se muestra el paso de un día a otro en cotizaciones, con una de las vistas mencionadas.

[RETURN Days](#)

(Atención: no confundir con las constantes "Day" y "Days")

■ **BarIndex** : Contador de velas desde el inicio del histórico visible

El contador comienza en la vela situada más a la izquierda del histórico cargado, y cuenta todas las velas hasta llegar a la situada más a la derecha –incluyendo la vela actual que se está formando. La primera vela visible (situada en el extremo izquierdo del gráfico) se considera la vela 0 (contrariamente al funcionamiento de las demás constantes). **BarIndex** se usa en la mayor parte de los casos con la instrucción **IF** que se describe más adelante.

■ **IntradayBarIndex** : Contador de velas intradiarias

El contador muestra la cantidad de velas generadas desde el inicio de la jornada bursátil y se reinicializa a 0 a cada inicio de jornada. Como en el caso precedente, la primera vela visible se considera la vela número 0 (contrariamente al funcionamiento de las demás constantes).

Comparemos las dos constantes creando dos indicadores separados:

`RETURN BarIndex`

y

`RETURN IntradayBarIndex`



Observamos cómo en el caso del indicador IntradayBarIndex, el contador pasa a valer 0 en el inicio de cada jornada bursátil.

■ Constantes de precio

Estas constantes permiten obtener una información más exhaustiva respecto a **Open**, **High**, **Low** y **Close**, ya que combinan estas informaciones de manera que se puedan resaltar ciertos aspectos de la psicología del mercado, resumidos en la vela actual.

- **Range** : diferencia entre High y Low.
- **TypicalPrice** : media entre High, Low y Close
- **WeightedClose** : media ponderada de High (peso 1) Low (peso 1) y Close (peso 2)
- **MedianPrice** : media entre High y Low
- **TotalPrice** : media entre Open, High, Low y Close

El **Range** resalta la volatilidad de la vela actual, que refleja el nerviosismo de los inversores.

La **WeightedClose** insiste en la importancia del precio al cierre (cotización de referencia), que se mantiene fija durante todo el día y separa dos velas consecutivas (siendo así aún más importante en el caso de las velas diarias o semanales).

Las constantes **TypicalPrice** y **TotalPrice** reflejan mejor la psicología del mercado dentro de la vela actual, ya que consideran respectivamente 3 y 4 niveles de precio alcanzados durante la jornada bursátil.

Bajo una perspectiva de robustez, **MedianPrice** permite explotar el plus de calidad explicativa que ofrecen las medianas respecto a las medias móviles, prestándose así mejor a modelizaciones teóricas que tratan de comprender mejor la psicología del mercado.

Range en % :

```
MyRange = Range
Calculo = (MyRange / MyRange[1] - 1) * 100
RETURN Calculo
```

■ Constante indefinida

La palabra clave **Undefined** permite instruir al indicador para que no muestre un resultado en ciertas variables. Por defecto, toda variable no definida está a cero.

- **Undefined** : dato indefinido (equivalente a una casilla vacía o a NULL)

Podrá ver un ejemplo en este mismo manual

🔗 Uso de indicadores ya existentes

Hemos observado hasta ahora las posibilidades ofrecidas por ProBuilder en lo tocante a las constantes y a su comportamiento al acceder a velas pasadas. Este comportamiento se aplica también al funcionamiento de indicadores preexistentes: a continuación veremos que los que Ud. programe seguirán el mismo principio.

Los indicadores ProBuilder se componen de 3 elementos, cuya sintaxis es como sigue:

NombreDeFunción [calculado en n velas] (sobre tal precio o cual indicador)

Cuando pulsamos el botón 'Insertar función' para buscar una función ProBuilder se crean automáticamente valores por defecto para el **período** y para el argumento **precio o indicador**

Average[20](Close)

Es posible modificar estos valores por defecto según nuestras preferencias: por ejemplo, reemplazando el período de 20 velas definidas por defecto, y fijándolo a 100 velas. Análogamente, se puede modificar el argumento del precio o del indicador 'close' por otro valor, como p.ej. "**DOpen**(6)":

Average[20](Dopen(6))

Veamos algunos ejemplos de comportamiento de indicadores preexistentes:

Este programa calcula la media móvil exponencial sobre 20 velas (períodos) aplicada al precio de cierre:

```
RETURN ExponentialAverage[20] (Close)
```

Este otro programa calcula una media móvil ponderada sobre 20 velas aplicada al precio estándar:

```
mm = WeightedAverage[20] (TypicalPrice)
RETURN mm
```

Cálculo de una media móvil alisada con el procedimiento de Wilder sobre 100 velas aplicada al Volumen:

```
mm = WilderAverage[100] (Volume)
RETURN mm
```

Programamos a continuación un código para calcular el MACD (en histograma) sobre el precio al cierre. La línea del MACD se construye como la diferencia entre la media móvil exponencial sobre 12 períodos menos aquella calculada sobre 26 períodos.

A continuación, se realiza un alisado con una media móvil exponencial a 9 períodos, aplicada a la diferencia para obtener la línea de Señal. El MACD en histograma se crea así a partir de la diferencia entre la línea del MACD y la línea de Señal.

```
REM Cálculo de la línea MACD
LineaMACD = ExponentialAverage[12] (Close) - ExponentialAverage[26] (Close)
REM Cálculo de la línea de Señal MACD
LineaSignal = ExponentialAverage[9] (LineaMACD)
REM Cálculo de la diferencia entre la línea del MACD y su Señal
MACDHistograma = LineaMACD - LineaSignal
RETURN MACDHistograma
```

Optimización de variables

Al crear un indicador se introduce un cierto número de constantes. La opción de optimización de variables (situada en la zona superior derecha) permite atribuir un valor por defecto a una constante indefinida para actuar posteriormente sobre el valor de la constante a partir de la interfaz de parámetros (propiedades) del indicador.

La ventaja consiste en la posibilidad de poder alterar los parámetros del indicador sin necesidad de modificar el código.

Por ejemplo, en el cálculo de una media móvil de 20 períodos:

```
RETURN Average[20] (Close)
```



Para poder modificar el número de períodos directamente desde la ventana de propiedades, vamos a reemplazar el valor 20 por una variable 'n':

```
RETURN Average[n] (Close)
```

A continuación, pulsaremos en 'Añadir' en 'Optimización de variables' para que aparezca la ventana 'Definición de variable', que vamos a rellenar así:

Definición de variable

Nombre usado en programa	n
Etiqueta en ventana propiedades	n
Tipo	Entero
Restricción	> 0
Valor por defecto	20

Aceptar Cancelar

A continuación, pulsamos 'OK'. En la ventana de Propiedades del indicador (llamada en este caso 'Propiedades - Mi Media Móvil'), obtendremos así un nuevo parámetro que nos permitirá modificar el número de períodos de la media móvil:

Propiedades - MyMovingAverage

Añadir

MyMovingAverage (20)
> MyMovingAverage (20)

Añadir nueva zona de color

Color

Color alcista Color bajista

Anchura

Estilo

☒ Resaltar último precio

n 20

☐ Convertir en configuración por defecto

Modificar indicador

Cerrar

Aplicando este método para crear múltiples variables, será posible combinar simultáneamente varios parámetros del indicador.



Capítulo II: Funciones e instrucciones ProBuilder



Estructuras de control



Instrucción condicional IF

La instrucción **IF** permite escoger acciones condicionadas, permitiendo subordinar un resultado a la verificación de una o varias condiciones definidas previamente.

La estructura se compone de los elementos **IF**, **THEN**, **ELSE**, **ELSIF**, **ENDIF**, que se combinan en función de la complejidad de las condiciones que deseemos definir. Vamos a ver cómo utilizarla.

Una condición, un resultado (IF THEN ENDIF)

Se puede buscar una condición y definir una acción que tendrá lugar si la condición se cumple. Si la condición no se cumple, no habrá acción (se obtendrá el valor por defecto de 0).

En el siguiente ejemplo, si el último precio es superior a la MM de período 20, se obtendrá el valor 1.

<code>IF Close > Average[20](Close) THEN</code>	SI precio > media móvil de 20 períodos ENTONCES
<code>Resultado = 1</code>	Resultado = 1
<code>ENDIF</code>	SI NO Resultado = 0 (por defecto) FIN DE CONDICIÓN
<code>RETURN Resultado</code>	MOSTRAR Resultado



'RETURN' deberá estar seguido de la variable de almacenamiento utilizada (en el ejemplo, Resultado) para mostrar el resultado de la condición

Una condición, dos resultados (IF THEN ELSE ENDIF)

También podemos definir un resultado distinto del definido por defecto en caso de que la condición no se cumpla. Retomando el ejemplo previo: si el último precio es superior a la MM de 20 períodos, se muestra el valor 1. En caso contrario, se muestra el valor -1.

```
IF Close > Average[20](Close) THEN
    Resultado = 1
ELSE
    Resultado = -1
ENDIF
RETURN Result
```

Nota: Acabamos de crear un indicador binario. Para más detalles al respecto, ver la ver la sección ['Creación de un indicador binario o ternario'](#) de este mismo manual.

Condiciones imbricadas

Se pueden crear subcondiciones aplicables tras la verificación de una condición principal. Eso implica condiciones que deberán verificarse secuencialmente (una tras otra) en orden de aparición. Para ello, basta con imbricar las condiciones **IF**, recordando insertar tantos **ENDIF** como **IF**. Veámoslo en este ejemplo:

Condiciones dobles en medias móviles:

```
IF (Average[12](Close) - Average[20](Close) > 0) THEN
    IF ExponentialAverage[12](Close) - ExponentialAverage[20](Close) > 0 THEN
        Resultado = 1
    ELSE
        Resultado = -1
    ENDIF
ENDIF
RETURN Resultado
```

■ Condiciones Múltiples (IF THEN ELSE ELSIF ENDIF)

Pueden definirse varios resultados, estando cada uno de ellos asociado a una condición específica. El indicador devolverá varios estados: si la Condición 1 se cumple, se activará la Acción 1; en cambio, si la Condición 2 se cumple, se activará la Acción 2 ... y si no se cumple ninguna condición, se activará la Acción n.

Sintácticamente, esta estructura emplea las instrucciones **IF, THEN, ELSIF, THEN ELSE, ENDIF**.

Y se escribe de la siguiente manera:

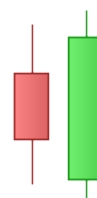
```
IF (Condición1) THEN
    (Acción1)
ELSIF (Condición2) THEN
    (Acción2)
ELSIF (Condición3) THEN
    (Acción3)
...
...
...
ELSE
    (Acción n)
ENDIF
```

Aunque la codificación es más compleja, también es posible reemplazar los **ELSIF** por **ELSE IF**. En tal caso, el bucle deberá terminar en tantos **ENDIF** como **IF** se hayan escrito. Si desea imbricar múltiples condiciones en su programa, recomendamos utilizar **ELSIF** de preferencia a **ELSE IF**.

Ejemplo: detección de desbordamientos alcistas y bajistas

Este indicador devolverá 1 si detecta un desbordamiento alcista; -1 si detecta un desbordamiento bajista; y 0 en los demás casos.

```
// Descripción de un desbordamiento alcista
Condition1 = Close[1] < Open[1]
Condition2 = Open < Close[1]
Condition3 = Close > Open[1]
Condition4 = Open < Close
```



```
// Descripción de un desbordamiento bajista
Condition5 = Close[1] > Open[1]
Condition6 = Close < Open
Condition7 = Open > Close[1]
Condition8 = Close < Open[1]
```



```
IF Condition1 AND Condition2 AND Condition3 AND Condition4 THEN
    a = 1
ELSIF Condition5 AND Condition6 AND Condition7 AND Condition8 THEN
    a = -1
ELSE
    a = 0
ENDIF
RETURN a
```

Ejemplo: Resistencia pivot Demarks

```
IF DClose(1) > DOpen(1) THEN
    Phigh = DHigh(1) + (DClose(1) - DLow(1)) / 2
    Plow = (DClose(1) + DLow(1)) / 2
ELSIF DClose(1) < DOpen(1) THEN
    Phigh = (DHigh(1) + DClose(1)) / 2
    Plow = DLow(1) - (DHigh(1) - DClose(1)) / 2
ELSE
    Phigh = DClose(1) + (DHigh(1) - DLow(1)) / 2
    Plow = DClose(1) - (DHigh(1) - DLow(1)) / 2
ENDIF
RETURN Phigh , Plow
```

Ejemplo: BarIndex

En el capítulo I de este manual habíamos presentado **BarIndex** como un contador del número de velas (o de barras) desde el inicio del histórico visible. **BarIndex** se utiliza a menudo asociado con **IF**. Por ejemplo, si queremos verificar si nuestro gráfico contiene más de 23 velas, escribiremos:

```
IF BarIndex <= 23 THEN
    a = 0
ELSIF BarIndex > 23 THEN
    a = 1
ENDIF
RETURN a
```

Bucle iterativo FOR

El bucle **FOR** se usa en una serie finita de elementos, permitiendo llamarlos uno a uno. Estos elementos pueden llamarse en cualquier orden, siempre que la serie en sí misma esté ordenada.

La estructura se compone de **FOR**, **TO**, **DOWNTO**, **DO**, **NEXT**. El uso de **TO** o de **DOWNTO** varía en función de la llamada en orden creciente o decreciente de los elementos. Es importante subrayar que el código situado entre el **FOR** y el **DO** son los extremos del recorrido del intervalo.

Avance creciente (FOR, TO, DO, NEXT)

```
FOR (Variable = ValorDeInicioDeSerie) TO ValorDeFinalDeSerie DO
    (Acción)
NEXT
```

Ejemplo: alisado de la media móvil de 12 períodos (MM12)

Vamos a crear una variable de almacenamiento (Resultado) que sumará una a una cada media móvil, de períodos 11, 12 y 13.

```
Resultado = 0
FOR Variable = 11 TO 13 DO
    Resultado = Average[Variable](Close) + Resultado
NEXT
REM Calculemos la media de las medias móviles dividiendo Resultado por 3 y
almacenando el resultado en AverageResult.
AverageResult = Resultado / 3
RETURN AverageResult
```

Veamos lo que sucede en cada paso:

Matemáticamente, buscamos hallar la media de las medias móviles aritméticas de períodos 11, 12 y 13.

Variable tomará sucesivamente los valores 11, 12 y 13

Resultado = 0

Variable = 11

Resultado recibe el valor del Resultado previo + MM11 = (0) + MM11 = (0 + MM11)

El programa pasa al valor siguiente del contador

Variable = 12

Resultado recibe el valor del Resultado previo + MM12 = (0 + M11) + MM12 = (0 + MM11 + MM12)

El programa pasa al valor siguiente del contador

Variable = 13

Resultado recibe el valor del Resultado previo + MM13 = (0 + M11 + M12) + M13 = (0 + M11 + M12 + M13)

El valor 13 es el último valor del contador.

Se cierra el bucle **"FOR"** con la instrucción **"NEXT"**.

Se muestra el resultado

En este código, 'variable' tomará inicialmente el primer valor de la serie, a continuación tomará el valor siguiente (el precedente + 1) y así sucesivamente hasta que 'variable' tome el valor del final de la serie. De este modo queda formado el bucle.

Ejemplo: Media sobre las últimas 20 velas de máximos:

<code>IF BarIndex < 20 THEN</code>	Si no hay más de 20 períodos en el histórico, entonces
<code>MMhigh = Undefined</code>	Se atribuye a MMhigh el valor por defecto 'indefinido'
<code>ELSE</code>	Si no
<code>FOR i = 0 TO 20 DO</code>	Para los valores entre 1 y 20
<code>SUMhigh = High[i]+SUMhigh</code>	Se suman los últimos 20 valores de máximos
<code>NEXT</code>	
<code>ENDIF</code>	
<code>MMhigh = SUMhigh / 20</code>	Se divide esta suma por 20 y se iguala a MMhigh
<code>RETURN MMhigh</code>	Se muestra MMhigh

■ **Avance decreciente (FOR, DOWNTO, DO, NEXT)**

El avance decreciente se sirve en cambio de las instrucciones: **FOR, DOWNTO, DO, NEXT**.

Se escribe de esta manera:

```
FOR (Variable = ValorDeFinalDeSerie) DOWNTO ValorDeInicioDeSerie DO
    (Acción)
NEXT
```

Retomando el ejemplo de la media móvil en las últimas 20 velas de precios máximos:

Comprobamos que invertimos los extremos del intervalo recorrido.

```
IF BarIndex = 0 THEN
    MMhigh = Undefined
ELSE
    FOR i = 20 DOWNTO 1 DO
        SUMhigh = High[i] + SUMhigh
    NEXT
ENDIF
MMhigh = SUMhigh / 20
RETURN Mmhigh
```

Bucle condicional WHILE

El bucle **WHILE** aplica una acción siempre que una condición se mantenga válida. Comprobaremos que este bucle tiene muchos aspectos comunes con la instrucción condicional simple **IF/THEN/ENDIF**.

Sintácticamente, esta estructura emplea las instrucciones **WHILE**, **DO** (opcional), **WEND**

La estructura se escribe de esta manera:

```
WHILE (Condición) DO
    (Acción 1)
    ...
    (Acción n)
WEND
```

Mostramos a continuación un ejemplo intuitivo:

```
Resultado = 0
WHILE Close > Average[20] (Close) DO
    Resultado = 1
WEND
RETURN Resultado
```

Ejemplo: indicador que calcula el número de períodos al alza consecutivos

```
Increase = (Close > Close[1])
Count = 0
WHILE Increase[Count] DO
    Count = Count + 1
WEND
RETURN Count
```

Observación general para la instrucción condicional WHILE

De modo análogo al caso de IF, el programa no tratará el bucle condicional escrito si la condición de validación es desconocida.

Veámoslo en un ejemplo:

```
Count = 0
WHILE i <> 11 DO
    i = i + 1
    Count = Count + 1
WEND
RETURN Count
```

La instrucción **WHILE** no conoce el valor de origen de i. Por ello, no puede verificar si se confirma que i es efectivamente igual a 10.

El bucle utilizará sus recursos para definir la variable i y atribuirle el valor 0 por defecto. De este modo, 'Count' no será procesado cada vez que el valor de retorno sea 0.

El código correcto sería:

```
i = 0
Count = 0
WHILE i <> 11 DO
    i = i + 1
    Count = Count + 1
WEND
RETURN Count
```

En este código, i está inicializada correctamente. El bucle funcionará correctamente, ya que la condición es válida.

■ BREAK

La instrucción **BREAK** permite hacer una salida forzada de un bucle **WHILE** o de un bucle **FOR**. Cabe la posibilidad de crear combinaciones con el comando **IF**, tanto del bucle **WHILE** como **FOR**.

■ Con WHILE

Cuando queremos salir de un bucle condicional **WHILE** sin esperar a encontrar una situación que deshaga la condición de bucle, utilizamos **BREAK** siguiendo esta estructura:

```
WHILE (Condición) DO
    (Acción)
    BREAK
WEND
```

Tomemos por ejemplo un indicador que acumula períodos alcistas y bajistas consecutivos:

```
REM Indicador de tendencia: acumula el número de subidas y bajadas
REM Aumentamos de 1 un contador inicializado a 0 en caso de subida (o -1 para una bajada)
Subida = (Close - Close[1]) > 0
Indicator = 0
i = 0
WHILE Subida[i] DO
    Indicator = Indicator + 1
    i = i + 1
    BREAK
WEND
RETURN Indicator
```

En este código, si no hubiésemos utilizado **BREAK**, el bucle habría continuado y el resultado sería un indicador de tendencia que acumularía el número de alzas consecutivas.

■ Con FOR

Cuando se busca salir de un bucle iterativo **FOR** sin llegar al último (o al primer) valor de la serie, se utiliza **BREAK** estructurado de esta manera:

```
FOR (Variable = ValorDeFinalDeSerie) TO ValorDeInicioDeSerie DO
    (Acción)
    BREAK
NEXT
```

Tomemos el ejemplo de un indicador que acumula las subidas consecutivas del volumen de las últimas 19 velas. Este indicador devolverá 0 si el volumen es bajista.

```
Indicator = 0
FOR i = 0 TO 19 DO
    IF (Volume[i] > Volume[i + 1]) THEN
        Indicator = Indicator + 1
    ELSE
        BREAK
    ENDIF
NEXT
RETURN Indicator
```

En este ejemplo, si no se hubiese utilizado **BREAK**, el bucle habría continuado hasta llegar a 19 (el último elemento de la serie) incluso si la condición de volumen no fuese válida.

Con **BREAK**, en cambio, en cuanto la condición deja de ser válida, el programa devuelve el resultado y se vuelve a situar a 0.

■ CONTINUE

La instrucción **CONTINUE** permite reemplazar la lectura del programa en la línea de inicio de un bucle **WHILE** o **FOR**. Suele combinarse a menudo con **BREAK** para dar una orden de salida del bucle (**BREAK**) o para mantenerse en el bucle (**CONTINUE**).

■ Con WHILE

Vamos a crear un programa que acumula la cantidad de velas con un nivel de apertura y cierre inferiores a los del período inmediatamente previo. Si la condición no se verifica, el contador volverá a 0.

```
Subida = Close > Close[1]
Contador = 0
WHILE Open < Open[1] DO
    IF Subida[Contador] THEN
        Contador = Contador + 1
        CONTINUE
    ENDIF
BREAK
WEND
RETURN Contador
```

A través de **CONTINUE**, y tras verificar que se cumple la condición del **IF**, el programa no sale del bucle **WHILE**, pudiendo así acumular el número de figuras que verifican esta condición. Sin la instrucción **CONTINUE**, el programa saldría del bucle independientemente de que la condición del **IF** se cumpla o no. En tal caso, no podríamos acumular la aparición de configuraciones, y el resultado sería binario (1,0).

■ Con FOR

Vamos a crear un programa que acumule el número de velas que tengan un cierre superior al de la inmediatamente anterior. Si la condición no se cumple, el contador volverá a 0.

```
Subida = Close > Close[1]
Contador = 0
FOR i = 1 TO BarIndex DO
    IF Subida[Contador] THEN
        Contador = Contador + 1
        CONTINUE
    ENDIF
BREAK
NEXT
RETURN Contador
```

FOR permite comprobar la condición en todo el histórico disponible. Mediante **CONTINUE**, cuando la condición del **IF** se cumple, nos mantenemos en el bucle **FOR** y continuamos con el valor del **i** siguiente. Ello permite acumular las configuraciones que cumplen la condición.

Sin la instrucción **CONTINUE**, el programa saldría del bucle independientemente de que la condición del **IF** se cumpla o no. No sería posible acumular las apariciones de las configuraciones buscadas, y el resultado sería binario (1,0).

■ ONCE

La instrucción **ONCE** permite ordenar a una variable que se dé ‘una SOLA VEZ’.

En todo programa, el lenguaje de programación leerá el código tantas veces como velas haya en el gráfico antes de devolver un resultado. Por ello, se debe tener siempre presente que **ONCE**:

- es procesado una sola y única vez por el programa, aunque éste realice relecturas durante su ejecución.
- cuando se produce la relectura del código por el lenguaje de programación, ésta conservará los valores calculados durante la lectura previa.

Para comprender bien el funcionamiento de este comando, es necesario tener una visión clara de cómo el lenguaje lee el código. Esto se ve con claridad en el ejemplo siguiente:

Tenemos aquí dos programas que devuelven respectivamente los valores 0 y 15, donde la única diferencia entre ambos es el comando **ONCE**:

Programa 1

```
1 Contador = 0
2 i = 0
3 IF i <= 5 THEN
4     Contador = Contador + i
5     i = i + 1
6 ENDIF
7 RETURN Contador
```

Programa 2

```
1 ONCE Contador = 0
2 ONCE i = 0
3 IF i <= 5 THEN
4     Contador = Contador + i
5     i = i + 1
6 ENDIF
7 RETURN Contador
```

Veamos en detalle cómo el programa lee los códigos.

Programa 1 :

el lenguaje lee L1 (Contador = 0; i = 0), a continuación L2, L3, L4, L5 y L6 (Contador = 0; i = 1), vuelve a L1 y relee todo el código de nuevo y de idéntica manera. Con **RETURN**, el lenguaje opera una salida del programa tras haber leído este último ‘n velas veces’. El resultado final es 0 (cero), al igual que tras la primera lectura.

Programa 2 :

el lenguaje lee L1 (Contador = 0; i = 0), a continuación L2, L3, L4, L5, L6 (Contador = 0; i = 1); llega a la línea ‘**RETURN**’, retoma el bucle a partir de L3 (**las líneas con ONCE sólo se procesan durante la primera lectura**), L4, L5, L6 (Contador = 1; i = 2), vuelve de nuevo (Contador = 3; i = 3) y así sucesivamente hasta que (Contador = 15; i = 6). Tras llegar a este resultado, la bucle en **IF** deja de procesarse puesto que la condición requerida para ello ya no se da; sólo le quedará leer L7. De ahí el nuevo resultado: 15.

Funciones matemáticas

Funciones comunes unarias y binarias

Examinemos ahora las funciones matemáticas principales.

Observe que a y b son ejemplos de argumentos decimales. Pueden reemplazarse por cualquier otra variable en su programa.

- **MIN(a, b)** : calcula el mínimo de a y de b
- **MAX(a, b)** : calcula el máximo de a y de b
- **ROUND(a)** : calcula un redondeo a la unidad de a
- **ABS(a)** : calcula el valor absoluto de a
- **SGN(a)** : indica el signo de a (1 si positivo, -1 si negativo)
- **SQUARE(a)** : calcula el cuadrado de a
- **SQRT(a)** : calcula la raíz cuadrada de a
- **LOG(a)** : calcula el logaritmo neperiano de a
- **EXP(a)** : calcula el exponencial (potencia) de a
- **COS(a)** : calcula el coseno de a
- **SIN(a)** : calcula el seno de a
- **TAN(a)** : calcula la tangente de a
- **ATAN(a)** : calcula la arcotangente de a

Vamos a codificar un ejemplo: la ley matemática normal, cuyo interés aquí reside en que se utiliza tanto en el cálculo del cuadrado, como de la raíz cuadrada o en el cálculo de la potencia exponencial:

```
REM Ley Normal aplicada en x = 10, Desviación Tipica = 6 y Esperanza = 8
REM Definamos en variable optimizada:
DesviacionTipica = 6
Esperanza = 8
x = 10
Indicator = EXP((1 / 2) * (SQUARE(x - Esperanza) / DesviacionTipica)) /
(DesviacionTipica * SQRT(2 / 3.14))
RETURN Indicator
```

Operadores matemáticos comunes

- **a < b** : a es estrictamente inferior a b
- **a <= b o a =< b** : a es inferior o igual a b
- **a > b** : a es estrictamente superior a b
- **a >= b o a => b** : a es superior o igual a b
- **a = b** : a es igual a b (o a recibe el valor b)
- **a <> b** : a es distinto de b

Funciones de comparaciones gráficas

- **a CROSSES OVER b** : a cruza b al alza
- **a CROSSES UNDER b** : a cruza b a la baja

Funciones sumatorias

- **cumsum** : Halla el sumatorio (suma acumulada) de todas las velas del gráfico

La sintaxis de uso de **cumsum** es :

```
cumsum (precio o indicador)
```

- **summation** : Halla la suma sobre un número de velas a definir

La suma se calcula a partir de la vela más reciente (de derecha a izquierda)

La sintaxis de uso de **summation** es :

```
summation[cantidad de velas] (precio o indicador)
```

Funciones estadísticas

La sintaxis de uso de estas funciones es análoga a la de los indicadores y de la función Summation:

```
lowest[cantidad de velas] (precio o indicador)
```

- **lowest** : da el valor mínimo en el período definido
- **highest** : da el valor máximo en el período definido
- **STD** : da la desviación típica en un valor para un período definido
- **STE** : da el error de desviación en un valor para un período definido

Operadores lógicos

Como en todo lenguaje informático, es preciso disponer de operadores lógicos para crear indicadores pertinentes. Veamos a continuación los 4 operadores lógicos de ProBuilder:

- **NOT(a)** : NO lógico
- **a OR b** : O lógico
- **a AND b** : Y lógico
- **a XOR b** : O exclusivo

Cálculo del indicador de tendencia: On Balance Volume (OBV) :

```
IF NOT((Close > Close[1]) OR (Close = Close[1])) THEN
    MiOBV = MiOBV - Volume
ELSE
    MiOBV = MiOBV + Volume
ENDIF
RETURN MiOBV
```

Instrucciones ProBuilder

- **RETURN** : devuelve el resultado
- **CustomClose** : envía un valor de precio configurable; por defecto, envía 'Close'
- **CALL** : llama una función previamente creada por el usuario
- **AS** : nombra los distintos resultados mostrados
- **COLOURED** : aplica a la curva mostrada un color a definir

■ RETURN

Ya hemos visto en el primer capítulo la importancia de la instrucción **RETURN**. Posee propiedades específicas que deben conocerse para evitar ciertos errores de programación.

Para utilizar correctamente **RETURN** en el codaje de un programa:

- debe emplearse una sola y única vez
- en la última línea de código
- puede usarse opcionalmente con otras funciones como AS y COLOURED
- para mostrar múltiples resultados, escribiremos RETURN seguido de los resultados que deseamos mostrar y separados por comas (ejemplo: RETURN a, b)

■ REM o //

REM (que también puede expresarse como //) permite insertar comentarios en el código. Estos comentarios le ayudarán a recordar el funcionamiento de la función programada. Los comentarios son leídos por el programa, pero obviamente no serán procesados por el código.

Veamos un ejemplo para ilustrar el concepto con más claridad:

```
REM Este programa calcula la media aritmética de 20 periodos al precio de cierre
RETURN Average[20] (Close)
```



No utilice NUNCA caracteres especiales (ejemplos: é,ñ,ç,ó...) en ProBuilder, ni siquiera dentro de un comentario **REM**

■ CustomClose

CustomClose es una constante que permite mostrar las constantes **Close**, **Open**, **High**, **Low** y otros valores, que pueden seleccionarse en la ventana de propiedades del indicador.

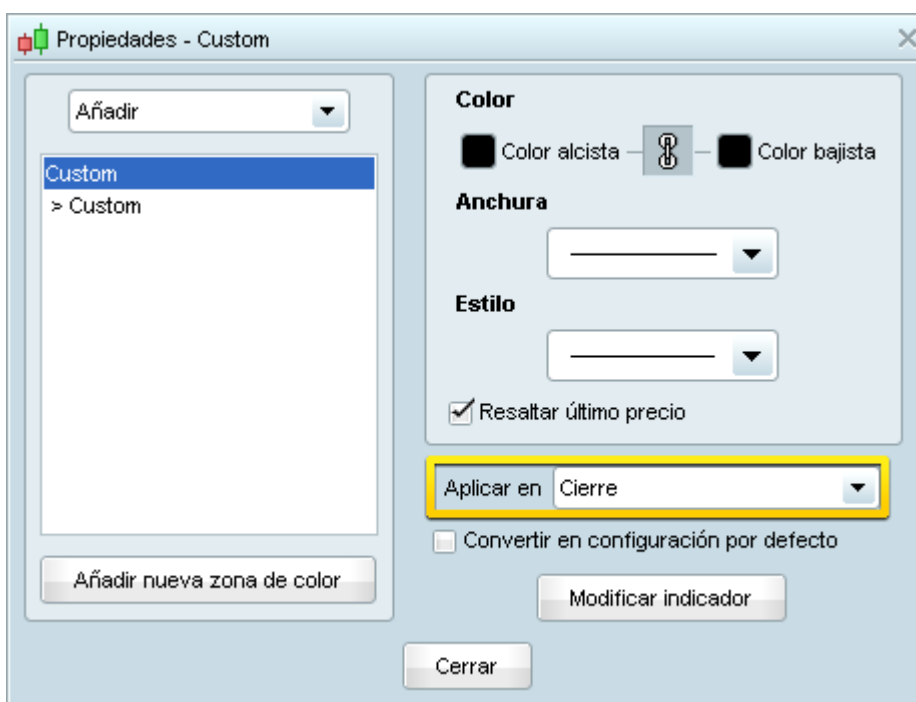
La sintaxis de uso es análoga a la de las constantes de precio que se adaptan a la vista del gráfico:

```
CustomClose[n]
```

Veamos un ejemplo sencillo:

```
RETURN CustomClose[2]
```

Al pulsar sobre el icono con forma de llave inglesa situado en el ángulo superior izquierdo del gráfico, comprobará que es posible configurar los precios utilizados para el cálculo (en la zona resaltada en amarillo de la imagen que se muestra a continuación)



CALL

CALL permite llamar a un indicador personalizado ya existente en la plataforma.

El método más rápido consiste en seleccionar directamente a partir de la categoría 'Indicadores personales' (en el botón 'Insertar función' al que se accede tras pulsar 'Nuevo Indicador') el indicador a utilizar.

Supongamos que ha codificado con el nombre HistoMACD el indicador de MACD en histograma.

Seleccione el indicador en la lista y pulse 'Añadir'. En la zona de programación aparecerá el Código

```
myHistoMACD = CALL HistoMACD
```

La plataforma ha nombrado directamente el antiguo indicador personalizado 'HistoMACD' como 'myHistoMACD'.

Esto significa que a efectos del resto del programa, si desea utilizar este indicador antiguamente llamado HistoMACD, deberá llamarlo con la denominación 'myHistoMACD'.

AS

La palabra clave **AS** nombra el resultado mostrado. Esta instrucción se utiliza conjuntamente con **RETURN** según la siguiente estructura:

```
RETURN Resultado1 AS "Nombre Curva", Resultado2 AS "Nombre Curva", ...
```









La ventaja consiste en facilitar la identificación de los elementos que componen el indicador creado.

Ejemplo:

```
a = ExponentialAverage[200] (Close)
b = WeightedAverage[200] (Close)
c = Average[200] (Close)
RETURN a AS "Media Exponencial", b AS "Media Ponderada", c AS "Media Aritmética"
```

COLOURED

COLOURED se utiliza tras el comando **RETURN** para colorear la línea visible con un color definido según la norma RGB (red, green, blue). A continuación indicamos los colores principales de esta norma:

Color	Valor RGB (rojo, verde, azul)	Español
	(0, 0, 0)	negro
	(255, 255, 255)	blanco
	(255, 0, 0)	rojo
	(0, 255, 0)	verde
	(0, 0, 255)	azul
	(255, 255, 0)	amarillo
	(0, 255, 255)	azul celeste
	(255, 0, 255)	añil

La sintaxis de uso del comando Coloured es la siguiente:

```
RETURN Indicator COLOURED(Red, Green, Blue)
```

El comando **AS** puede asociarse al comando **COLOURED**(r, g, b) respetando el siguiente orden:

```
RETURN Indicator COLOURED(Red, Green, Blue) AS "Nombre De Mi Curva"
```

Retomando el ejemplo previo, insertaremos **COLOURED** en la línea del 'RETURN'.

```
a = ExponentialAverage[200](Close)
b = WeightedAverage[200](Close)
c = Average[200](Close)
RETURN a COLOURED(255, 0, 0) AS "Media Movil Exponencial", b COLOURED(0, 255, 0)
AS "Weighted Media Ponderada", c COLOURED(0, 0, 255) AS "Media Movil Simple"
```

La imagen muestra la personalización de los colores en el gráfico resultante.



Capítulo III: Aplicaciones prácticas

Creación de un indicador binario o ternario: lógica y método

Por definición, un indicador binario o ternario no puede devolver más de dos o tres resultados posibles (generalmente 0 ,1 o -1) respectivamente. Su utilidad principal en el ámbito bursátil consiste en permitir la verificación inmediata de la condición que constituye el indicador.

Utilidad de un indicador binario o ternario:

- permite detectar las principales configuraciones de velas japonesas
- facilita la lectura del gráfico cuando se buscan varias condiciones simultáneas
- permite crear alertas de una sola condición sobre un indicador que incorpora varias ➡ ¡dispondrá así de una mayor cantidad de alertas!
- detecta condiciones complejas, también sobre datos históricos
- facilita la programación de un backtest (sistema o estrategia)

Los indicadores binarios o ternarios se construyen con ayuda de la función IF. Por ello, se recomienda releer la sección referente a esta función para aprovechar mejor las explicaciones ofrecidas a continuación.

Ilustremos la creación de estos indicadores personalizados para detectar configuraciones de precios:

Indicador binario: detección de un Martillo

```
// Detección de un Martillo
Martillo = Close>Open AND High = Close AND (Open-Low)>=3*(Close-Open)
IF Martillo THEN
    Resultado = 1
ELSE
    Resultado = 0
ENDIF
RETURN Resultado AS "Martillo"
```



Indicador ternario: detección de Cruces doradas y de Cruces mortales

```
a = ExponentialAverage[10](Close)
b = ExponentialAverage[20](Close)
c = 0
// Detección de Cruz dorada
IF a CROSSES OVER b THEN
    c = 1
ENDIF
// Detección de Cruz mortal
IF a CROSSES UNDER b THEN
    c = -1
ENDIF
RETURN c
```




Nota: Hemos mostrado las medias móviles exponenciales de 10 y 20 períodos aplicadas al precio de cierre para evidenciar la correspondencia de los resultados del indicador con más nitidez.

Encontrará otros ejemplos de indicadores de detección de configuraciones de precios en los ejercicios de este manual.

Crear indicadores STOP: siga sus posiciones en tiempo real

Pueden crearse indicadores que representen STOP (o lo que es lo mismo, niveles de salida potencial) definidos en función de parámetros personalizados.

Con el módulo ProBackTest, cuya información puede consultarse en un manual independiente, ya dispone de la posibilidad de definir los niveles de salida de un sistema o estrategia. No obstante, el interés de programar un indicador que sigue la cotización del valor reside en el hecho que:

- permite visualizar el stop como una línea que se actualiza en tiempo real directamente sobre el gráfico del precio
- no necesita definir las órdenes de compra ni de venta (lo que sí es necesario en la programación de un ProBackTest)
- puede asociar alertas en tiempo real (y recibirlas en pantalla o en su móvil) para ser informado inmediatamente en el momento en que la condición fijada se esté cumpliendo

La programación de los Stop le permitirá aplicar los principales comandos presentados en los capítulos precedentes.

Par ailleurs, dans le manuel ProBackTest vous pourrez trouver plusieurs exemples de stop à insérer dans des stratégies d'investissement.

Existen 4 tipologías de stop que examinaremos a continuación:

- **STOP profit estático**
- **STOP loss estático**
- **STOP de inactividad**
- **STOP de seguimiento (trailing stop)**

Los códigos que se proponen en los ejemplos presentados a continuación constituyen sugerencias para la construcción de indicadores de stop. Obviamente, deberá personalizarlos según las instrucciones descritas en los capítulos anteriores.

■ STOP profit estático

Un *STOP profit* o *Take-Profit*, designa un límite superior de salida de posición. Por definición, este límite es fijo: el inversor cerrará la posición con ganancias en el momento en que el precio cruce este límite de Stop.

El indicador que se presenta a continuación ofrece 2 niveles con toma de posición a partir del momento 'HoraInicio'.

- si Ud. es comprador, tiene una posición larga (long). Deberá considerar la curva superior que representa el 10% de ganancia –o lo que es lo mismo, el 110% el precio de compra.
- si Ud. es vendedor a descubierto, tiene una posición corta (short). Deberá considerar la curva inferior, que representa el 10% de ganancia –o lo que es lo mismo, el 90% del precio de venta a descubierto.

Veamos a continuación un ejemplo de Stop profit personalizable:

```
// Definición de variables optimizadas:
// HoraInicio = 100000 (en este ejemplo fijamos la hora de entrada a las 10h00min00s)
// Precio= Precio en la apertura de la posición
// Si tiene una posición larga (long), seguirá la curva superior. Si está en
posición corta (short), seguirá la curva inferior.
// AmplitudUp = 1.1 (representa la variación del precio, y se utiliza para trazar
el Stop profit en posiciones largas (long))
// AmplitudDown = 0.9 (representa la variación del precio, y se utiliza para
trazar el Stop profit en posiciones cortas (short))
IF Time = HoraInicio THEN
    StopLONG = AmplitudUp * Precio
    StopSHORT = AmplitudDown * Precio
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "TakeProfit LONG 10%", StopSHORT COLOURED(0,
255, 0) AS "TakeProfit SHORT 10%"
```

■ STOP loss estático

Un *STOP Loss* es el contrario del *STOP Take-Profit*; en lugar de definir un límite de salida con ganancias, se define un límite de pérdida máxima asumible, a partir del cual se cierra la posición. Al igual que el *Take-Profit*, el límite es fijo (estático).

El indicador que se presenta a continuación ofrece 2 niveles con toma de posición a partir del momento 'HoraInicio'.

- si Ud. es comprador, tiene una posición larga (long). Deberá considerar la curva superior que representa el 10% de pérdida –o lo que es lo mismo, el 90% el precio de compra
- si Ud. es vendedor a descubierto, tiene una posición corta (short). Deberá considerar la curva inferior, que representa el 10% de ganancia –o lo que es lo mismo, el 110% del precio de venta a descubierto.

Veamos a continuación un ejemplo de Stop profit personalizable:

```
// Definición de variables optimizadas:
// HoraInicio = 100000 (en este ejemplo fijamos la hora de entrada a las 10h00min00s)
// Si tiene una posición larga (long), seguirá la curva inferior. Si está en
posición corta (short), seguirá la curva superior.
// Precio= Precio en la apertura de la posición
// AmplitudUp = 1.1 (representa la variación del precio, y se utiliza para trazar
el Stop loss en posiciones cortas (short))
//AmplitudDown = 0.9 (representa la variación del precio, y se utiliza para
trazar el Stop loss en posiciones largas (long))
IF Time = HoraInicio THEN
    StopLONG = AmplitudUp * Precio
    StopSHORT = AmplitudDown * Precio
ENDIF
RETURN StopLONG COLOURED(0, 0, 0) AS "StopLoss BUY 10%", StopSHORT COLOURED(0,
255, 0) AS "StopLoss SELL 10%"
```

■ STOP de inactividad

Un STOP de inactividad cierra una posición cuando el precio se mueve en un rango demasiado estrecho y el objetivo de beneficio no se alcanza en un período determinado (expresado en número de velas).

El cierre de la posición tendrá lugar en el rango inferior si la posición es larga (long) o en el rango superior si la posición es corta (short).

El Stop mostrado a continuación es un indicador binario que devolverá el valor 1 si el STOP se activa, y el valor 0 en caso contrario.

Veamos un primer ejemplo:

```
// MiVolatilidad = 0.01 que corresponde a la brecha relativa de las bandas superior e inferior
IF IntradayBarIndex = 0 THEN
    ObjetivoShort = (1 - MiVolatilidad) * Close
    ObjetivoLong = (1 + MiVolatilidad) * Close
ENDIF
RETURN ObjetivoShort AS "ObjetivoShort", ObjetivoLong AS "ObjetivoLong"
```

Ejemplo de STOP de inactividad en gráfico intradiario:

```
// Definición de variables optimizadas:
REM La posición se abre a precio de mercado
// MiVolatilidad = 0.01 que corresponde a la brecha relativa de las bandas superior e inferior del rango definido
// NumeroDeVelas = 20 corresponde a la duración máxima en numero de velas permitida antes de forzar el cierre de la posición (resultado 1)
Resultado = 0
Cpt = 0
IF IntradayBarIndex = 0 THEN
    ObjetivoShort = (1 - MiVolatilidad) * Close
    ObjetivoLong = (1 + MiVolatilidad) * Close
ENDIF
FOR i = IntradayBarIndex DOWNT0 1 DO
    IF Close[i] >= ObjetivoShort AND Close[i] <= ObjetivoLong THEN
        Cpt = Cpt + 1
    ELSE
        Cpt = 0
    ENDIF
    IF Cpt = NumeroDeVelas THEN
        Resultado = 1
    ENDIF
NEXT
RETURN Resultado
```

Trailing STOP

Un *Trailing STOP* sigue de modo dinámico la evolución del precio, indicando el momento en que la posición ha de ser cerrada.

Proponemos a continuación dos tipologías de *Trailing STOP*, correspondientes a la versión dinámica del Stop Loss y del Stop Profit.

Trailing STOP LOSS (aplicable en intradía)

```
// Definición de variables optimizadas:
// HoraInicio = 090000 (en este ejemplo fijamos la hora de entrada a las
09h00min00s)
REM La posición se abre a precio de mercado
// Amplitud = 0.9 (representa un stop al 10%)
IF Time = HoraInicio THEN
    IF lowest[5](Close) < 1.2 * Low THEN
        IF lowest[5](Close) >= Close THEN
            Cut = Amplitud * lowest[5](Close)
        ELSE
            Cut = Amplitud * lowest[20](Close)
        ENDIF
    ELSE
        Cut = Amplitud * lowest[20](Close)
    ENDIF
ENDIF
RETURN Cut AS "Trailing Stop Loss"
```

Trailing STOP Profit (aplicable en intradía)

```
// Definición de variables optimizadas:
// HoraInicio = 090000 (en este ejemplo fijamos la hora de entrada a las
09h00min00s)
REM La posición se abre a precio de mercado
// Amplitud = 1.1 (representa un stop al 110%)
IF Time = HoraInicio THEN
    PrecioInicio = Close
ENDIF
Precio = PrecioInicio - AverageTrueRange[10]
TrailingStop = Amplitud * highest[15](Precio)
RETURN TrailingStop COLOURED (255, 0, 0) AS "Trailing take profit"
```

Capítulo IV: Ejercicios

Configuraciones de velas japonesas

GAP UP o GAP DOWN



En este tipo de configuraciones, el color de las velas es irrelevante.

Definimos la amplitud como variable optimizada de valor 0.001

El gap se define con estas dos condiciones:

- la apertura del día actual es estrictamente superior al cierre de la víspera, O BIEN estrictamente inferior a cierre de la víspera;
- el valor absoluto resultante de calcular $[(\text{apertura del día} - \text{cierre víspera}) / \text{cierre víspera}]$ es estrictamente superior a la amplitud.

```
// Definición de variable optimizada
// Amplitud (del Gap)
Amplitud = 0.001
// Primera condición de existencia del Gap
IF Open > Close[1] OR Open < Close[1] THEN
    // Segunda condición de existencia del Gap
    IF ABS((Open - Close[1]) / Close[1]) > Amplitud THEN
        // Búsqueda del Gap
        Detector = SGN(Open - Close[1])
    ELSE
        Detector = 0
    ENDIF
ELSE
    Detector = 0
ENDIF
// Resultado
RETURN Detector AS "Gap deteccion"
```

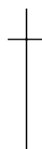
Doji (versión amplia)



El Doji se define como un rango estrictamente superior a 5 veces el valor absoluto de (Open – Close).

```
Doji = Range > ABS (Open - Close) * 5
RETURN Doji AS "Doji"
```

Doji (versión estricta)



En su versión estricta, el Doji se define por un Close = Open.

```
Doji = (Open = Close)
RETURN Doji AS "Doji"
```

Indicadores

BODY MOMENTUM

La definición matemática del Body Momentum es así:

$$\text{BodyMomentum} = 100 * \text{BodyUp} / (\text{BodyUp} + \text{BodyDown})$$

BodyUp es un contador del número de velas que cierran a un nivel de precios superior al de la apertura, durante un período definido (en el código de este ejemplo tomamos un período = 14). El contador BodyDown es análogo en sentido inverso (cierre inferior a apertura).

```
Periodos = 14
b = Close - Open
IF BarIndex > Periodos THEN
  Bup = 0
  Bdn = 0
  FOR i = 1 TO Periodos
    IF b[i] > 0 THEN
      Bup = Bup + 1
    ELIF b[i] < 0 THEN
      Bdn = Bdn + 1
    ENDIF
  NEXT
  BM = (Bup / (Bup + Bdn)) * 100
ELSE
  BM = Undefined
ENDIF
RETURN BM AS "Body Momentum"
```

■ OSCILADOR DE ONDAS ELLIOT

El oscilador de ondas Elliot representa la diferencia de medias móviles.

La media móvil corta representa la acción del precio, mientras que la media móvil larga representa la tendencia de fondo.

Cuando los precios forman una onda 3, aumentarán fuertemente. En consecuencia, el valor del oscilador aumentará sensiblemente.

Al llegar a una onda 5, los precios aumentarán más lentamente y el oscilador tomará un valor mucho menos elevado.

```
RETURN Average[5](MedianPrice) - Average[35](MedianPrice) AS "Oscilador Ondas Elliot"
```

■ Williams %R

Su funcionamiento es similar al de un estocástico. Para trazarlo, se definen primero dos curvas.

El %R queda así definido por $(\text{Close} - \text{LowestL}) / (\text{HighestH} - \text{LowestL}) * 100$

```
HighestH = highest[14](High)
LowestL = lowest[14](Low)
MiWilliams = (Close - LowestL) / (HighestH - LowestL) * 100
RETURN MiWilliams AS "Williams %R"
```

■ Bandas de Bollinger

Las bandas se componen de una media móvil de 20 períodos que se aplica al cierre. La media móvil se multiplica por el doble de la desviación típica sobre las 20 velas previas del precio al cierre para limitar las bandas superior e inferior.

```
a = Average[20](Close)
// Definicion de la desviacion tipica
DesviacionTipica = STD[20](Close)
Bsup = a + 2 * DesviacionTipica
Binf = a - 2 * DesviacionTipica
RETURN a AS "Average", Bsup AS "Bollinger Up", Binf AS "Bollinger Down"
```

Glosario

A

Código	Implementación	Función
<code>ABS</code>	<code>ABS(a)</code>	Función matemática "Valor Absoluto"
<code>AccumDistr</code>	<code>AccumDistr(price)</code>	Designa la Acumulación Distribución clásica
<code>ADX</code>	<code>ADX[N]</code>	Indicador Average Direccional Index
<code>ADXR</code>	<code>ADXR[N]</code>	Indicador Average Direccional Index Rate
<code>AND</code>	<code>a AND b</code>	Operador lógico Y
<code>AroonDown</code>	<code>AroonDown[P]</code>	Designa el Aroon Down
<code>AroonUp</code>	<code>AroonUp[P]</code>	Designa el Aroon Up
<code>ATAN</code>	<code>ATAN(a)</code>	Función matemática "Arctangente"
<code>AS</code>	<code>RETURN Result AS "ResultName"</code>	Instrucción utilizada para atribuir un nombre a una curva
<code>Average</code>	<code>Average[N](price)</code>	Media Móvil Aritmética
<code>AverageTrueRange</code>	<code>AverageTrueRange[N](price)</code>	Designa la Media móvil con alisado de Wilder del True Range

B

Código	Implementación	Función
<code>BarIndex</code>	<code>BarIndex</code>	Número de velas desde el inicio de la descarga de datos (en el gráfico en el caso de un indicador ProBuilder o para un sistema de trading en el caso de ProBacktest o ProInvest)
<code>BollingerBandWidth</code>	<code>BollingerBandWidth[N](price)</code>	Ancho de banda de Bollinger
<code>BollingerDown</code>	<code>BollingerDown[N](price)</code>	Soporte de la banda de Bollinger
<code>BollingerUp</code>	<code>BollingerUp[N](price)</code>	Resistencia de la banda de Bollinger
<code>BREAK</code>	<code>(FOR...DO...BREAK...NEXT)</code> o <code>(WHILE...DO...BREAK...WEND)</code>	Instrucción de salida forzada de bucle FOR o WHILE

C

Código	Implementación	Función
<code>CALL</code>	<code>myResult = CALL myFunction</code>	Llamada de función del usuario
<code>CCI</code>	<code>CCI[N](price)</code> o <code>CCI[N]</code> aplicado por defecto en <code>TypicalPrice</code>	Commodity Channel Index
<code>ChaikinOsc</code>	<code>ChaikinOsc[Ch1, Ch2](price)</code>	Designa el oscilador de Chaikin
<code>Chandle</code>	<code>Chandle[N](price)</code>	Designa el Chande Momentum Oscillator
<code>ChandeKrollStopUp</code>	<code>ChandeKrollStopUp[Pp, Qq, X]</code>	Stop de protección según Chande y Kroll en posición compradora
<code>ChandeKrollStopDown</code>	<code>ChandeKrollStopDown[Pp, Qq, X]</code>	Stop de protección según Chande y Kroll en posición vendedora
<code>Close</code>	<code>Close[N]</code>	Designa el precio de cierre de la vela actual o de n días previos
<code>COLOURED</code>	<code>RETURN Result</code> <code>COLOURED(R,G,B)</code>	Colorea una curva de un cierto color según la convención RGB
<code>COS</code>	<code>COS(a)</code>	Función Coseno
<code>CROSSES OVER</code>	<code>a CROSSES OVER b</code>	Operador booleano que verifica que una curva pase por encima de otra
<code>CROSSES UNDER</code>	<code>a CROSSES UNDER b</code>	Operador booleano que verifica que una curva pase por debajo de otra
<code>cumsum</code>	<code>cumsum(price)</code>	Sumatorio de un precio desde el inicio del histórico mostrado
<code>CurrentDayOfWeek</code>	<code>CurrentDayOfWeek</code>	Designa el día actual
<code>CurrentHour</code>	<code>CurrentHour</code>	Designa la hora actual
<code>CurrentMinute</code>	<code>CurrentMinute</code>	Designa el minuto actual
<code>CurrentMonth</code>	<code>CurrentMonth</code>	Designa el mes actual
<code>CurrentSecond</code>	<code>CurrentSecond</code>	Designa el segundo actual
<code>CurrentTime</code>	<code>CurrentTime</code>	Designa HoraMinuto actual
<code>CurrentYear</code>	<code>CurrentYear</code>	Designa el año actual
<code>CustomClose</code>	<code>CustomClose[N]</code>	Constante configurable en la ventana de propiedades al mostrar el gráfico (por defecto, cierre)
<code>Cycle</code>	<code>Cycle(price)</code>	Indicador Ciclo (en precio)

D

Código	Implementación	Función
Date	Date[N]	Designa la fecha de cierre de la vela actual
Day	Day[N]	Día de cierre de la vela actual
Days	Days[N]	Contador de días desde 1900
DayOfWeek	DayOfWeek[N]	Designa el día de la semana durante el cual la vela actual se ha cerrado
DClose	DClose(N)	Precio de cierre del enésimo día previo al de la vela actual
DEMA	DEMA[N](price)	Doble Media Móvil Exponencial
DHigh	DHigh(N)	Precio máximo del enésimo día previo al de la vela actual
DI	DI[N](price)	Designa el Demand Index (Índice de la Demanda)
DIminus	DIminus[N](price)	Designa el DI-
DIplus	DIplus[N](price)	Designa el DI+
DLow	DLow(N)	Precio mínimo del enésimo día previo al de la vela actual
DO	Ver FOR et WHILE	Instrucción opcional de los bucles FOR y WHILE para introducir la acción de inicio de bucle
DOpen	DOpen(N)	Precio de apertura del enésimo día previo al de la vela actual
DOWNT0	Ver FOR	Instrucción que se aplica sobre el bucle FOR para ordenar una lectura decreciente
DPO	DPO[N](price)	Designa el Detrented Price Oscillator

E

Código	Implementación	Función
EaseOfMovement	EaseOfMovement[I]	Designa el indicador Ease of Movement
ELSE	Ver IF/THEN/ELSE/ENDIF	Instrucción de llamada de la segunda condición a defecto de la primera salida de IF
ELSEIF	Ver IF/THEN/ELSE/ENDIF	Contracción de ELSE IF (imbricación en un conjunto de instrucciones condicionales de otro conjunto de instrucciones)
EMV	EMV[N]	Designa el indicador Ease of Movement Value
ENDIF	Ver IF/THEN/ELSE/ENDIF	Instrucción a introducir al final del conjunto de instrucciones condicionales
EndPointAverage	EndPointAverage[N](price)	Último punto de Media Móvil
EXP	EXP(a)	Función matemática "Exponencial"
ExponentialAverage	ExponentialAverage[N](price)	Media Móvil Exponencial

F - G

Código	Implementación	Función
FOR/TO/NEXT	FOR i=a TO b DO a NEXT	Bucle "Para" (Toma todos los valores del inicio al final designado, o vice versa (DOWNTO))
ForceIndex	ForceIndex(price)	Da el indicador Force Index, que determina quién controla el mercado (vendedor, comprador)

H

Código	Implementación	Función
High	High[N]	Designa el precio máximo alcanzado durante el período N
highest	highest[N](price)	Designa el máximo de un período sur un horizonte temporal dado
HistoricVolatility	HistoricVolatility[N](price)	Designa la volatilidad histórica (o volatilidad estadística)
Hour	Hour[N]	Designa la hora de cierre de cada vela del histórico

I - J - K

Código	Implementación	Función
IF/THEN/ENDIF	IF a THEN b ENDIF	Conjunto de instrucciones condicionales sin segunda condición
IF/THEN/ELSE/ENDIF	IF a THEN b ELSE c ENDIF	Conjunto de instrucciones condicionales
IntradayBarIndex	IntradayBarIndex[N]	Cuenta el número de velas en el gráfico intraday

L

Código	Implementación	Función
LinearRegression	LinearRegression[N](price)	Regresión lineal
LinearRegressionSlope	LinearRegressionSlope[N](price)	Pendiente de la regresión lineal
LOG	LOG(a)	Función matemática "logaritmo neperiano"
Low	Low[N]	Designa el mínimo alcanzado durante el período
lowest	lowest[N](price)	Designa el mínimo de un período dentro de un horizonte temporal dado

M

Código	Implementación	Función
MACD	MACD[S,L,Sl](price)	Designa al Moving Average Convergence Divergence (MACD) en histograma
MACDline	MACDLine[S,L](price)	Designa la línea del MACD
MassIndex	MassIndex[N]	Indicador Mass Index aplicado en N velas
MAX	MAX(a,b)	Función matemática "Máximo"
MedianPrice	MedianPrice	Media del precio máximo y del mínimo
MIN	MIN(a,b)	Función matemática "Mínimo"
Minute	Minute	Designa el minuto del instante del cierre de cada vela del histórico
MOD	a MOD b	Función matemática "Resto del cociente euclídeo"
Momentum	Momentum[I]	Designa el Momentum (precio de cierre actual – precio de cierre de la enésima vela precedente)
MoneyFlow	MoneyFlow[N](price)	Designa el MoneyFlow entre -1 y 1
MoneyFlowIndex	MoneyFlowIndex[N]	Designa el MoneyFlowIndex
Month	Month[N]	Designa el mes de cierre de cada vela del histórico

N

Código	Implementación	Función
NEXT	Ver FOR/TO/NEXT	Instrucción a introducir al final del bucle "Para" (FOR)
NOT	NOT a	Operador lógico NO

O

Código	Implementación	Función
OBV	OBV(price)	Designa el "On-Balance-Volume"
ONCE	ONCE VariableName = VariableValue	Instrucción que precede a otra que deseamos ejecutar una sola vez
Open	Open[N]	Designa el precio de apertura de la vela actual o celle de n días previos
OR	a OR b	Operador lógico O

P - Q

Código	Implementación	Función
PriceOscillator	PriceOscillator[S,L](price)	Indicador Percentage Price oscillator
PositiveVolumeIndex	PriceVolumeIndex(price)	Designa el indicador Positive Volume Index
PVT	PVT(price)	Designa el indicador "Price Volume Trend"

R

Código	Implementación	Función
R2	R2[N](price)	Designa el coeficiente R Cuadrado (error de precios en la regresión lineal)
Range	Range[N]	Devuelve el Range (rango, diferencia entre el precio máximo y mínimo de la vela actual)
REM	REM comentario	Precede un comentario (el código no lo toma en cuenta, pero facilitan la lectura al usuario)
Repulse	Repulse[N](price)	Devuelve el indicador Repulse (mide la fuerza alcista y bajista de cada vela)
RETURN	RETURN Resultado	Instrucción que envía el resultado
ROC	ROC[N](price)	Designa el "Price Rate of Change"
RSI	RSI[N](price)	Designa el oscilador "Relative Strength Index"
ROUND	ROUND(a)	Función matemática "Redondeo a la unidad" (parte entera)

S

Código	Implementación	Función
SAR	SAR[At,St,Lim]	Designa el Parabólico SAR
SARatdmf	SARatdmf[At,St,Lim](price)	Designa el Parabólico SAR en el módulo ATDMF
SIN	SIN(a)	Función matemática "Seno"
SGN	SGN(a)	Función matemática "Signo de"
SMI	SMI[N,SS,DS](price)	Designa el índice Estocástico Momentum (Stochastic Momentum Index)
SmoothedStochastic	SmoothedStochastic[N,K](price)	Designa un estocástico alisado
SQUARE	SQUARE(a)	Función matemática "Cuadrado" (potencia 2)
SQRT	SQRT(a)	Función matemática "Raíz cuadrada"
STD	STD[N](price)	Función estadística "Desviación Típica"
STE	STE[N](price)	Función estadística "Error típico"
Stochastic	Stochastic[N,K](price)	Línea %K del Estocástico
summation	summation[N](price)	Suma de un cierto precio de las N últimas velas
Supertrend	Supertrend[STF,N]	Designa el Super Trend

T

Código	Implementación	Función
TAN	TAN(a)	Función matemática "Tangente"
TEMA	TEMA[N](price)	Media Móvil Exponencial Triple
THEN	Ver IF/THEN/ELSE/ENDIF	Instrucción que sigue la primera condición del conjunto condicional "IF"
Time	Time[N]	Da la evolución de la HoraMinutoSegundo y permite llamar la hora en el programa
TimeSeriesAverage	TimeSeriesAverage[N](price)	Media Móvil de las series temporales
TO	Ver FOR/TO/NEXT	Instrucción "hasta " en el bucle "Para" (FOR)
Today	Today	Fecha actual
TotalPrice	TotalPrice[N]	(cierre + apertura + máximo + mínimo) / 4
TR	TR(price)	Designa el True Range
TriangularAverage	TriangularAverage[N](price)	Media Móvil Triangular
TRIX	TRIX[N](price)	Triple Media Móvil Exponencial
TypicalPrice	TypicalPrice[N]	Designa el precio Típico (Media de máximo, mínimo y cierre)

U

Código	Implementación	Función
Undefined	a = Undefined	Permite dejar una variable indefinida (es un tipo de variable)

V

Código	Implementación	Función
Variation	Variation(price)	Da la diferencia entre el cierre de la víspera y el cierre actual en %
Volatility	Volatility[S, L]	Designa la volatilidad de Chaikin
Volume	Volume[N]	Designa el volumen
VolumeOscillator	VolumeOscillator[S,L]	Designa el oscilador de volumen
VolumeROC	VolumeROC[N]	Designa el volumen del Rate Of Change (ROC)

W

Código	Implementación	Función
<code>WeightedAverage</code>	<code>WeightedAverage[N](price)</code>	Designa la Media Móvil Ponderada
<code>WeightedClose</code>	<code>WeightedClose[N]</code>	Da la Media entre el precio de cierre, máximo y mínimo, con ponderaciones respectivas de 2, 1 y 1
<code>WEND</code>	Ver WHILE/DO/WEND	Instrucción a introducir al final del bucle While/Do/Whend (mientras)
<code>WHILE/DO/WEND</code>	WHILE (condición) DO (acción) WEND	Bucle "Mientras"
<code>WilderAverage</code>	<code>WilderAverage[N](price)</code>	Da la Media Móvil de Wilder
<code>Williams</code>	<code>Williams[N](close)</code>	Calcula el %R de Williams
<code>WilliamsAccumDistr</code>	<code>WilliamsAccumDistr(price)</code>	Indicador Acumulación/Distribución de Williams

X

Código	Implementación	Función
<code>XOR</code>	a XOR b	Operador lógico O exclusivo

Y

Código	Implementación	Función
<code>Year</code>	<code>Year[N]</code>	Da la evolución de los años y permite llamar a los años en el programa
<code>Yesterday</code>	<code>Yesterday[N]</code>	Da la evolución del día de ayer y permite llamar a este último en el programa

Z

Código	Implementación	Función
<code>ZigZag</code>	<code>ZigZag[Zr](price)</code>	Designa los Zig-Zag de la teoría de las ondas de Elliott
<code>ZigZagPoint</code>	<code>ZigZagPoint[Zp](price)</code>	Designa los Zig-Zag de la teoría de las ondas de Elliott calculadas a Z puntos

Otros

Código	Función	Código	Función
<code>+</code>	Operador de adición	<code><></code>	Operador de diferencia
<code>-</code>	Operador de sustracción	<code><</code>	Operador de inferioridad estricta
<code>*</code>	Operador de multiplicación	<code>></code>	Operador de superioridad estricta
<code>/</code>	Operador de división decimal	<code><=</code>	Operador de inferioridad
<code>=</code>	Operador de igualdad	<code>>=</code>	Operador de superioridad

10.32	38.78	84.20	75.50	120.57	8.27	21.57	91.27	26.07	70.13	59.75	13.65	35.24	10.32	38.78	84.20	75.50	120.57	8.27	21.57	91.27
-1.06	-0.71	-0.22	+0.28	+1.4	-0.04	-1.71	-0.95	-0.03	+0.54	+1.12	-0.09	-1.46	-1.06	-0.71	-0.22	+0.28	+1.4	-0.04	-1.71	-0.95



ProRealTime.com

— La referencia de los programas de bolsa en línea —

