

GUIDE DE PROGRAMMATION

Fonctions de Base & Indicateurs (ProBuilder)

SOMMAIRE

| | |
|---|----|
| ■■■ Présentation de ProBuilder | 1 |
| ■■■ Chapitre I : Les notions fondamentales | 2 |
| ➡ Utiliser ProBuilder..... | 2 |
| › Création d'indicateur..... | 2 |
| › Raccourcis clavier de l'éditeur de code..... | 5 |
| ➡ Spécificités de programmation du langage ProBuilder..... | 6 |
| › Les spécificités..... | 6 |
| › Le modèle d'exécution..... | 7 |
| › Les variables..... | 7 |
| ➡ Les constantes financières ProBuilder..... | 8 |
| › Les constantes de prix et de volume adaptées à l'unité de temps du graphique..... | 8 |
| › Les constantes journalières de prix..... | 9 |
| › Les constantes temporelles..... | 9 |
| › Les constantes dérivées des prix..... | 13 |
| › La constante Undefined..... | 13 |
| ➡ Utilisation des indicateurs préexistants..... | 14 |
| ➡ Ajout de variables paramétrables..... | 17 |
| ■■■ Chapitre II : Fonctions et instructions ProBuilder | 19 |
| ➡ Structures de contrôle..... | 19 |
| › Instruction conditionnelle IF..... | 19 |
| • Une condition, un résultat (IF THEN ENDIF)..... | 19 |
| • Une condition, deux résultats (IF THEN ELSE ENDIF)..... | 19 |
| • Conditions imbriquées..... | 19 |
| • Conditions Multiples (IF THEN ELSIF ELSE ENDIF)..... | 20 |
| › Boucle itérative FOR..... | 22 |
| • Avancement croissant (FOR, TO, DO, NEXT)..... | 22 |
| • Avancement décroissant (FOR, DOWNT0, DO, NEXT)..... | 23 |
| › Boucle conditionnelle WHILE..... | 24 |
| › BREAK..... | 25 |
| • BREAK avec WHILE..... | 25 |
| • BREAK avec FOR..... | 26 |
| › CONTINUE..... | 27 |
| • CONTINUE avec WHILE..... | 27 |
| • CONTINUE avec FOR..... | 27 |
| › ONCE..... | 28 |
| ➡ Fonctions Mathématiques..... | 29 |
| › Fonctions usuelles unaires et binaires..... | 29 |
| › Opérateurs mathématiques usuels..... | 29 |
| › Fonctions de comparaisons graphiques..... | 29 |
| › Fonctions de sommation..... | 30 |
| › Fonctions Statistiques..... | 30 |

| | |
|--|-----------|
| ➡ Opérateurs logiques..... | 30 |
| ➡ Mots-clefs ProBuilder..... | 31 |
| › RETURN..... | 31 |
| › Commentaires..... | 31 |
| › CustomClose..... | 31 |
| › CALCULATEONLASTBARS..... | 31 |
| › CALL..... | 32 |
| › AS..... | 32 |
| › COLOURED..... | 33 |
| ➡ Commandes de dessin..... | 35 |
| › Les paramètres supplémentaires..... | 38 |
| ➡ Les instructions multi-périodes..... | 41 |
| › Liste des périodes disponibles..... | 43 |
| ➡ Les tableaux de données..... | 44 |
| ➡ PRINT..... | 46 |
| ▀▀ Chapitre III : Applications pratiques | 48 |
| ➡ Créer un indicateur binaire ou ternaire : pourquoi et comment ?..... | 48 |
| ▀▀ Chapitre IV : exercices | 50 |
| ➡ Figures de chandeliers..... | 50 |
| ➡ Indicateurs..... | 52 |
| ▀▀ Glossaire | 54 |

Avertissement : ProRealTime n'exerce pas le service de Conseil en Investissement Financier. Ce document n'est en aucun cas une offre de conseil en investissement ni une incitation quelconque à acheter ou vendre des instruments financiers. Les exemples présentés dans ce manuel sont à but pédagogique. Pour votre propre trading, vous êtes entièrement libre dans le choix de vos critères. Les performances passées ne présagent pas de l'avenir. Tout système de trading peut vous exposer à un risque de perte supérieur à votre investissement initial.

Présentation de ProBuilder

ProBuilder est le langage de programmation de ProRealTime. Ce dernier sert à concevoir des indicateurs techniques personnalisés, des stratégies de trading (ProBacktest) ou des scans personnalisés (ProScreener). ProBacktest et ProScreener font l'objet de manuels individuels à cause de certaines spécificités de programmation.

Ce langage est de type BASIC, très simple d'utilisation et exhaustif dans les possibilités offertes.

Vous allez pouvoir construire vos propres programmes qui utilisent les cotations de n'importe quel instrument inclus dans l'offre ProRealTime, à partir des éléments de base :

- le cours d'ouverture de chaque barre : **Open**
- le cours de clôture de chaque barre : **Close**
- le plus haut de chaque barre : **High**
- le plus bas de chaque barre : **Low**
- le nombre de titres échangés : **Volume**

Les barres, ou chandeliers, sont les représentations graphiques standards des cotations reçues en temps réel. ProRealTime vous offre bien entendu la possibilité de personnaliser le type de style graphique, vous proposant, parmi d'autres, des vues telles que Renko, Kagi, Heikin-Ashi.

ProBuilder évalue les données de chaque barre de prix depuis la plus ancienne jusqu'à la plus récente, et exécute la formule développée dans le langage afin de déterminer la valeur des indicateurs sur la barre en question.

Les indicateurs développés sous ProBuilder peuvent être affichés sur le graphique du prix ou bien dans un graphique individuel.

Dans ce document, vous assimilerez au fur et à mesure les commandes permettant de programmer dans ce langage grâce à une vision théorique claire et des exemples concrets les illustrant.

A la fin de ce manuel, vous y trouverez un glossaire qui vous donnera une vue de l'ensemble des commandes de ProBuilder, des indicateurs déjà codés et d'autres fonctions complétant ce que vous aurez appris durant votre lecture.

Les utilisateurs plus habitués à la programmation pourront passer directement à la lecture du chapitre II ou bien consulter le glossaire afin de retrouver rapidement l'explication relative à la fonction recherchée.

Pour ceux moins habitués à programmer, nous conseillons le visionnage de la vidéo intitulée « [Créer un indicateur dans ProBuilder](#) » et de lire l'intégralité du manuel.

Pour toute question complémentaire sur le fonctionnement de ProBuilder, vous pouvez les poser auprès de notre communauté ProRealTime sur le [forum ProRealCode](#), vous y retrouverez aussi une [documentation en ligne](#) avec de nombreux exemples.

Votre chargé de clientèle dédié peut aussi vous aider avec vos questions, n'hésitez pas à demander.

En vous souhaitant nos meilleurs vœux de réussite et une bonne lecture.

L'équipe ProRealTime.

Chapitre I : Les notions fondamentales

Utiliser ProBuilder

Création d'indicateur

La zone de programmation d'un indicateur est disponible à partir du bouton « Indicateurs » qui se trouve sur chaque graphique de votre plateforme ProRealTime ou depuis le menu Affichage > Indicateurs/Backtest.



Vous accédez ensuite à la fenêtre de gestion des indicateurs. Vous pourrez :

- Afficher un indicateur prédéfini.
- Créer un indicateur personnalisé, qui pourra ensuite être appliqué à n'importe quelle valeur.

Dans le second cas cliquez sur « Créer », pour accéder à la fenêtre de programmation.

Vous avez alors la possibilité :

- De programmer directement un indicateur dans la zone de texte réservée au code.
- D'utiliser la fonction d'aide « Insérer Fonction » (icône f_x), qui permet de trouver dans une nouvelle fenêtre une bibliothèque des fonctions disponibles, séparées en huit catégories, afin de vous assister lors de la programmation.

```

Mon 1er indicateur

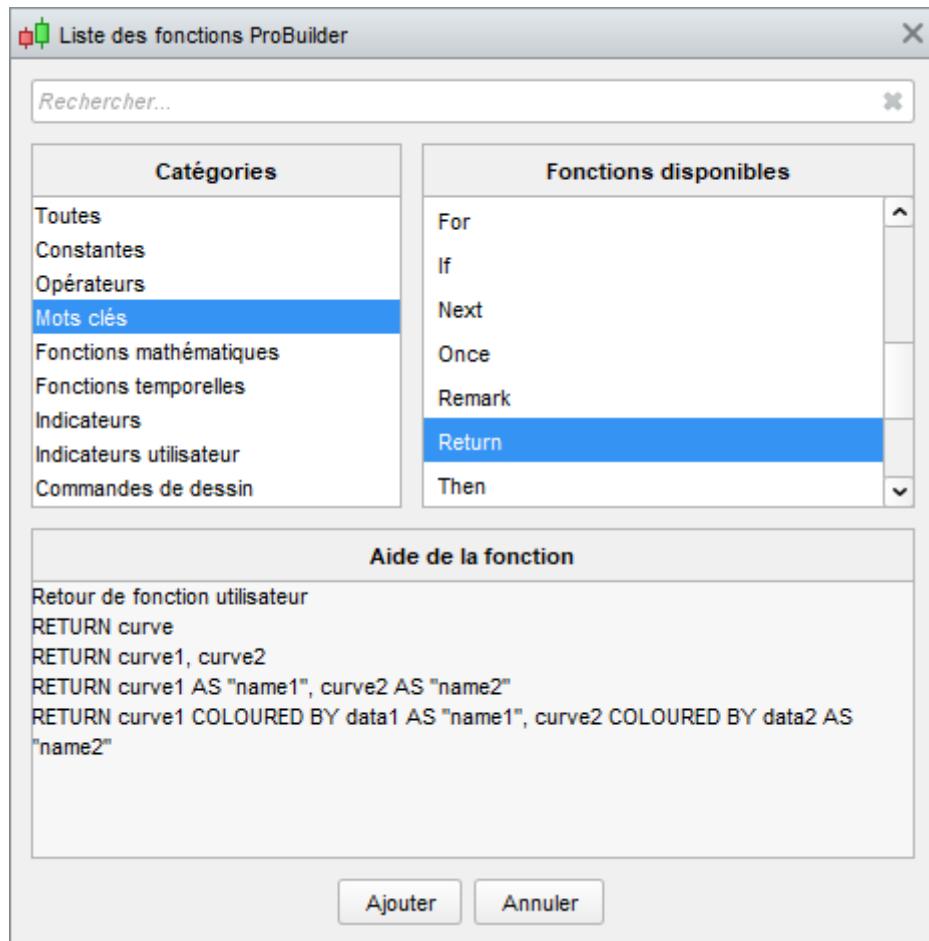
Variables 0

1 MyAverage = ( Close + Close[1] + Close[2] ) / 3
2 RETURN MyAverage

sur prix nouveau panneau Appliquer à APPLE - 1 heure Maintenir la fenêtre ouverte
Formations vidéo

```

Prenons comme exemple le premier élément caractéristique des indicateurs ProBuilder, c'est-à-dire la fonction « **RETURN** », (disponible dans la section « Liste des fonctions ProBuilder » f_x - voir image ci-dessous).

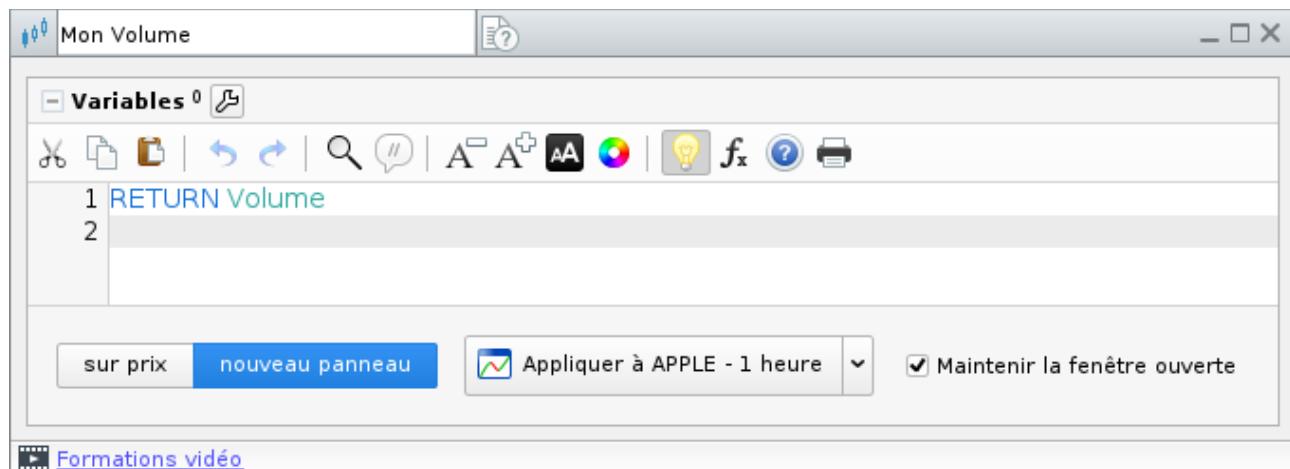


Sélectionnez donc le mot « **RETURN** » et cliquez sur « Ajouter » : la commande s'ajoutera bien à la zone de programmation.

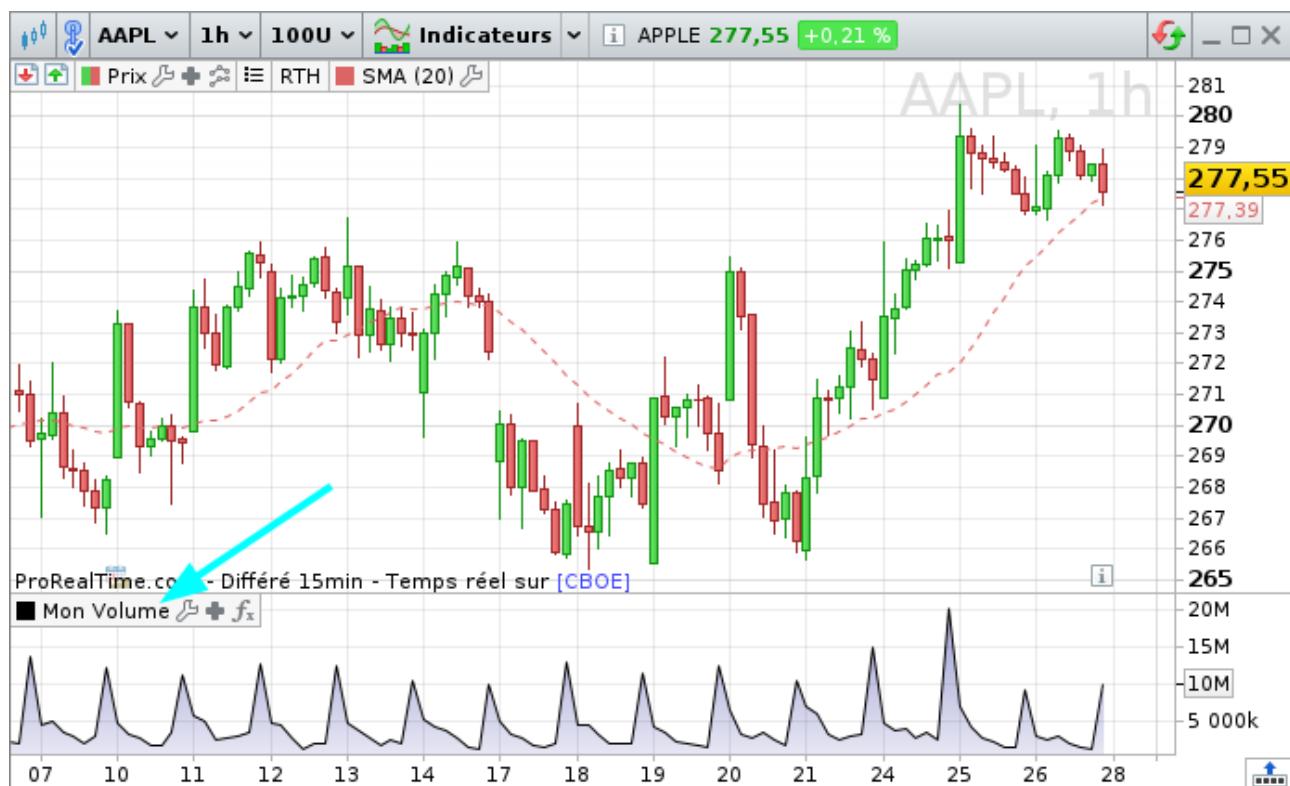


RETURN vous permet d'afficher le résultat de votre indicateur

Supposons que l'on veuille créer un indicateur affichant le Volume. Si vous avez déjà inséré le mot **RETURN**, il suffit alors d'aller une nouvelle fois sur « Insérer fonction », de cliquer sur « Constantes » dans la section « Catégories », puis sur le côté droit, section « Fonctions disponibles », cliquer sur « **Volume** ». Enfin, cliquez sur « Ajouter ». N'oubliez pas d'insérer un espace entre chaque instruction.



Avant de cliquer sur le bouton « Appliquer », précisez en haut de la fenêtre le nom de votre indicateur. Enfin, cliquez sur « Appliquer » et vous verrez s'afficher le graphique avec votre indicateur.



Raccourcis clavier de l'éditeur de code

La fenêtre de programmation de la plateforme ProRealTime possède plusieurs fonctionnalités pratiques utilisables via des raccourcis clavier :

- Tout sélectionner (Ctrl+A) : Sélectionne tout le texte présent dans l'éditeur de code
- Copier (Ctrl + C) : Copie le texte sélectionné
- Couper (Ctrl + X) : Coupe le texte sélectionné
- Coller (Ctrl + V) : Colle le texte copié
- Annuler (Ctrl + Z) : Annule la dernière action faite dans l'éditeur de code
- Refaire (Ctrl + Y) : Refait la dernière action faite dans l'éditeur de code
- Rechercher / Remplacer (Ctrl + F) : Cherche un texte dans l'éditeur de code / Remplace un texte dans l'éditeur de code
- Commenter / Dé-commenter (Ctrl + R) : Commente le code sélectionné / Dé-commente le code sélectionné : le code commenté sera précédé de « // » et coloré en gris. Il ne sera pas pris en compte lors de l'exécution du code.
- Auto-Complétion (Ctrl+Espace) : Permet d'afficher des suggestions d'instructions ou de mots-clefs

Pour les utilisateurs de MAC, les mêmes raccourcis clavier peuvent être utilisés. Il suffit dans ce cas de remplacer la touche « Ctrl » par la touche « Command ».

La plupart de ces raccourcis peuvent aussi être utilisés via un clic droit dans l'éditeur de code.

Spécificités de programmation du langage ProBuilder

Les spécificités

Le langage ProBuilder vous permet de manipuler de nombreuses commandes classiques ainsi que des outils plus élaborés spécifiques à l'analyse technique, qui vous donneront la possibilité de programmer des indicateurs du plus simple au plus sophistiqué.

Les principes clés à connaître sur le langage ProBuilder sont :

- Il n'est **pas nécessaire de déclarer les variables**.
- Il n'est **pas nécessaire de typer les variables**.
- Il n'y a **pas de différence entre majuscule et minuscule**.
- On utilise le même symbole pour **l'affectation et l'égalité mathématique**.

Qu'est-ce que cela signifie ?

- Déclarer une variable X, c'est indiquer son existence. Dans ProBuilder, vous pouvez directement utiliser X sans avoir défini antérieurement son existence. Prenons un exemple en écrivant :

Avec déclaration : Soit la variable X, On attribue à X la valeur 5

Sans déclaration : On attribue à X la valeur 5 (donc implicitement, X existe et vaut 5)

En ProBuilder il suffit d'écrire : X=5

- Typer une variable, c'est-à-dire définir la nature de la variable : est-elle un entier (ex : 3 ; 8 ; 21 ; 643 ; ...) un nombre décimal (ex : 1.76453534535...), un booléen (VRAI, FAUX),...

- Dans ProBuilder, vous pouvez écrire vos commandes aussi bien avec des majuscules qu'avec des minuscules. Par exemple, l'ensemble de commandes **IF / THEN / ELSE / ENDIF** pourra indifféremment être écrite **iF / tHeN / ELse / endIf**.

- Affecter une valeur à une variable, c'est lui attribuer une valeur. Pour mieux comprendre le principe d'affectation, il faut que vous considériez une variable comme une boîte vide qui attend qu'on lui mette quelque chose à l'intérieur. Le schéma ci-dessous vous illustre ce principe avec la valeur Volume affectée à la variable X :

X ← Volume

Voyez bien qu'on lit de droite à gauche la phrase : Volume est affecté à X.

Maintenant, pour l'écrire en code ProBuilder, on va simplement remplacer la flèche par un signe =

X = Volume

Le même symbole = est utilisé :

- Pour l'affectation d'une variable (comme dans l'exemple précédent).
- Comme opérateur mathématique de comparaison (1+ 1= 2 est équivalent à 2 = 1 + 1).

Le modèle d'exécution

Contrairement aux langages de programmation classiques qui s'exécutent une fois puis s'arrêtent, **ProBuilder s'exécute une fois par chandelier** en commençant par le chandelier le plus ancien.

Lorsqu'il atteint le chandelier en cours de construction, le comportement change en fonction du moteur ProBuilder utilisé (Indicateur, ProBacktest, ProOrder AutoTrading, ProScreener) :

- En **Indicateur** le code est réévalué à chaque tick.
- En **ProBacktest** et en **ProOrder** le code est évalué à la clôture du chandelier.
- En **ProScreener** le code est réexécuté depuis le 1^{er} chandelier dès que le marché a été intégralement scanné. Cela permet d'avoir la fenêtre de ProScreener à jour en permanence.

Les variables

Dans les langages de programmation classiques les variables contiennent une seule valeur. En ProBuilder, les variables fonctionnent différemment. **Elles historisent leurs valeurs à chaque chandelier rencontré.** L'historique s'agrandit donc au fur et à mesure que le code progresse sur les chandliers historiques.

Chaque valeur de l'historique est accessible à tout moment avec l'opérateur « **[n]** » permettant de récupérer la valeur à n chandelier dans le passé par rapport au chandelier courant. Il est aussi possible d'aller chercher la n-ième valeur de l'historique de la manière suivante « **[BarIndex+1-n]** »

- MaVariable[3] : Valeur de MaVariable 3 chandliers avant le chandelier courant.
- MaVariable[BarIndex-2] : Valeur de MaVariable au 3ème chandelier de l'historique

Bien que les variables ressemblent à des listes d'autres langages de programmation leur fonctionnement est bien différent :

- Il est impossible de modifier le passé : MaVariable[3] = 42 n'est pas une instruction valide, l'historique peut être lu mais ne peut pas être modifié.
- Seulement la valeur au chandelier courant peut être éditée : MaVariable = 42 est valide.
- Une variable non éditée lors du chandelier courant prend la valeur qu'elle avait au chandelier précédent, comme si l'instruction suivante était insérée en début de code pour chaque variable utilisée dans le code : MaVariable = MaVariable[1]

Il existe des variables de type tableau (`$MonTableau[MonIndice]`) qui seront abordées en détails plus loin dans ce manuel. Contrairement aux variables classiques de probuilder, **les tableaux ne sont pas historisés**, l'opération suivante n'est pas valide « `$MonTableau[MonIndice][3]` » **elle ne permet pas** de lire la valeur de `$MonTableau[MonIndice]` 3 chandliers dans le passé.



Il est essentiel d'avoir une bonne compréhension du modèle d'exécution et du concept de variables historisées pour utiliser ProBuilder à son plein potentiel !

Les constantes financières ProBuilder

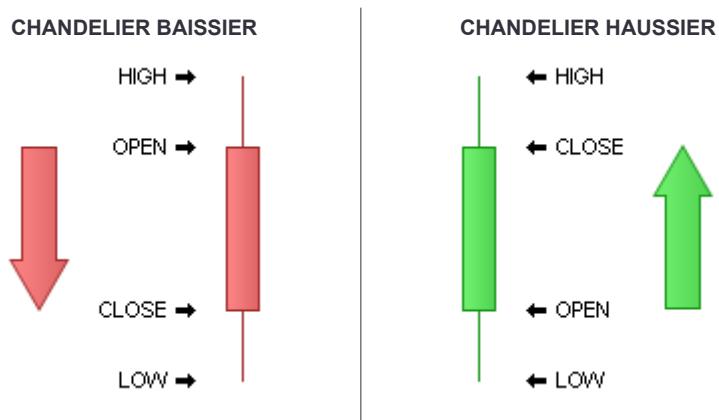
Avant de commencer à coder vos indicateurs personnels, il est nécessaire de connaître les éléments à partir desquels vous pourrez constituer votre code, tels que les prix d'ouverture et de clôture, le volume, etc... Ce sont les « fondamentaux » de l'analyse technique, et l'essentiel à connaître pour coder des indicateurs.

Vous pourrez ainsi les combiner afin de faire ressortir certains aspects de l'information fournie par les marchés financiers. On peut les regrouper en 5 catégories :

Les constantes de prix et de volume adaptées à l'unité de temps du graphique

Ce sont les constantes « classiques » les plus utilisées. Elles reportent par défaut les valeurs de la barre en cours (quelle que soit l'unité de temps du graphique) et se présentent de la façon suivante :

- **Open** : le cours d'ouverture de la barre courante.
- **High** : le cours le plus haut de la barre courante.
- **Low** : le cours le plus bas de la barre courante.
- **Close** : le cours de clôture de la barre courante.
- **Volume** : le nombre de titres ou de lots échangés sur la barre courante.



Exemple : Range de la barre courante

a = **High**

b = **Low**

MyRange = a - b

RETURN MyRange // La constante « Range » existe pour simplifier l'accès à cette valeur.

Pour faire appel aux valeurs des barres précédentes, il suffit de rajouter entre crochets le nombre de barres à partir de la barre courante.

Prenons par exemple la constante prix de clôture. L'appel du cours se fait de la façon suivante :

Valeur de la clôture de la barre courante :

Close

Valeur de la clôture de la barre qui précède la courante :

Close[1]

Valeur de la clôture de la n-ième barre qui précède la courante :

Close[n]

Cette règle vaut pour n'importe quelle constante ou variable. Par exemple, le prix d'ouverture de la 2ème barre précédant la barre courante, sera appelé par : **Open[2]**.

La valeur qui sera reportée dépendra de la période affichée sur le graphique.

Les constantes journalières de prix

Contrairement aux constantes adaptées à l'unité de temps du graphique, les constantes journalières de prix se réfèrent aux valeurs de la journée, indépendamment de la période affichée sur le graphique.

Une autre différence à constater avec les constantes adaptées à l'unité de temps est que les constantes journalières utilisent des parenthèses pour obtenir leurs valeurs des jours précédents.

- **DOpen(n)** : prix d'ouverture de la $n^{\text{ème}}$ journée antérieure à celle de la barre courante.
- **DHigh(n)** : prix le plus haut de la $n^{\text{ème}}$ journée antérieure à celle de la barre courante.
- **DLow(n)** : prix le plus bas de la $n^{\text{ème}}$ journée antérieure à celle de la barre courante.
- **DClose(n)** : prix de clôture de la $n^{\text{ème}}$ journée antérieure à celle de la barre courante.

Remarque : si « n » est égal à 0, « n » fera référence à la journée en cours. Les valeurs maximales et minimales n'étant pas encore définitives pour $n=0$, nous obtiendrons des résultats qui pourront évoluer au cours de la journée en fonction des minimums et maximums atteints par la valeur.



Pour les constantes adaptées à l'unité de temps on utilise des crochets, pour les constantes journalières on utilise des parenthèses.

Close[3] ➔ Le prix de clôture 3 chandeliers auparavant

DClose(3) ➔ Le prix de clôture il y a 3 jours

Les constantes temporelles

Le temps est une composante parfois négligée de l'analyse technique. Pourtant les traders connaissent bien l'importance de certains moments de la journée, ou de certaines dates de l'année. Il est donc possible de limiter l'analyse de son indicateur à des moments spécifiques en utilisant les constantes suivantes :

- **Date** : Date codée sous la forme AAAAMMMJJ indiquant la date de clôture de chaque barre.

Les constantes temporelles sont considérées par ProBuilder comme des entiers. La constante Date, par exemple, doit être présentée comme un unique nombre composé de 8 chiffres.

Écrivons alors le programme :

RETURN Date

Supposons que nous soyons le 4 juillet 2020. L'indicateur issu du programme ci-dessus va nous renvoyer le résultat suivant 20200704.

Pour lire une date, il suffit donc de la lire de la manière suivante :

20200704 = 2020 années 07 mois et 04 jours.

Attention, dans l'écriture d'une date au format AAAAMMMJJ, MM doit être compris entre 1 et 12 et JJ doit être compris entre 1 et 31.

- **Time** : HeureMinuteSeconde codée sous la forme HHMMSS indiquant l'heure de clôture de chaque barre.

Faisons par exemple :

RETURN Time

On obtient une courbe liant toutes les heures de clôture de chaque barre :



Pour lire une heure, il suffit donc de lire de la manière suivante :

160000 = 16 heures 00 minutes et 00 secondes.

Attention, dans l'écriture d'une heure au format HHMMSS, HH doit être compris entre 0 et 23, MM doit être compris entre 0 et 59 et SS doit être compris lui aussi entre 0 et 59.

Il est possible de combiner dans un même indicateur **Time** et **Date** afin de restreindre le résultat à un moment spécifique. Dans l'exemple qui suit, nous voulons limiter notre indicateur au premier octobre 2025 à 9h00 et 1 sec.

```
a = (Date = 20251001)
b = (Time = 090001)
RETURN (a AND b)
```

De la même façon fonctionnent les constantes suivantes :

- **Timestamp** : Date et heure **UNIX** (nombre de secondes depuis le 1^{er} janvier 1970) de la clôture de chaque barre.
- **Second** : Seconde de la clôture de chaque barre (entre 0 et 59).
- **Minute** : Minute de la clôture de chaque barre (entre 0 et 59).
- **Hour** : Heure de la clôture de chaque barre (entre 0 et 23).
- **Day** : Jour du mois de la clôture de chaque barre (entre 1 et 28 ou 29 ou 30 ou 31).
- **Month** : Mois de la clôture de chaque barre (entre 1 et 12).
- **Year** : Année de la clôture de chaque barre.
- **DayOfWeek** : Jour de la semaine à la clôture de chaque barre (0=dimanche, 1=lundi, 2=mardi, 3=mercredi, 4=jeudi, 5=vendredi, 6=samedi).

Il existe aussi leurs dérivés en **Open** :

- **OpenTimestamp** : Date et heure **UNIX** de l'ouverture de chaque barre.
- **OpenSecond** : Seconde de l'ouverture de chaque barre (entre 0 et 59).
- **OpenMinute** : Minute de l'ouverture de chaque barre (entre 0 et 59).
- **OpenHour** : Heure de l'ouverture de chaque barre (entre 0 et 23).
- **OpenDay** : Jour du mois de l'ouverture de chaque barre (entre 1 et 28 ou 29 ou 30 ou 31).
- **OpenMonth** : Mois de l'ouverture de chaque barre (entre 1 et 12).
- **OpenYear** : Année de l'ouverture de chaque barre.
- **OpenDayOfWeek** : Jour de la semaine à l'ouverture de chaque barre (0=dimanche, 1=lundi, 2=mardi, 3=mercredi, 4=jeudi, 5=vendredi, 6=samedi).
- **OpenTime** : HeureMinuteSeconde codée sous la forme HHMMSS indiquant l'heure d'ouverture de chaque barre.
- **OpenDate** : Date (AAAA-MM-JJ) d'ouverture de la barre courante.

Exemple d'utilisation de ces constantes :

RETURN (Hour > 17) AND (Day = 30)

- **CurrentHour** : Heure actuelle (celle du marché).
- **CurrentMinute** : Minute actuelle (celle du marché).
- **CurrentMonth** : Mois actuel (celui du marché).
- **CurrentSecond** : Seconde actuelle (celle du marché).
- **CurrentTime** : HeureMinuteSeconde actuelle (celle du marché).
- **CurrentYear** : Année actuelle (celle du marché).
- **CurrentDayOfWeek** : Jour de la semaine actuelle selon le fuseau horaire du marché.

L'image suivante met en évidence cette différence appliquée aux constantes **CurrentTime** et **Time**. Pour simplifier, les constantes en « Current » font abstraction de l'axe des temps et ne considèrent que la valeur affichée dans l'encadré blanc.



Time indique l'heure de clôture de chaque barre

CurrentTime indique l'heure du marché

Si vous souhaitez régler vos indicateurs par rapport à des compteurs (nombres de jours écoulés, nombre des barres etc...), les constantes **Days**, **BarIndex** et **IntradayBarIndex** sont à votre disposition.

■ **Days** : Compteur de jours depuis 1900.

Cette constante est utile lorsqu'on souhaite connaître le nombre de jours qui se sont écoulés, particulièrement lorsqu'on travaille en vues quantitatives, comme (x) ticks ou (x) volumes.

L'exemple suivant permettra donc d'afficher le passage d'une journée à l'autre de cotations, lorsqu'on se trouve dans l'une des vues citées.

`RETURN Days`

(Attention à ne pas confondre les deux constantes “**Day**” et “**Days**”).

■ **BarIndex** : Compteur de barres depuis le début de l'historique affiché.

Le compteur part de la barre la plus à gauche de l'historique chargé et compte toutes les barres jusqu'à celle la plus à droite – barre en cours incluse. La première barre affichée (la plus à gauche) est considérée comme la barre 0. **BarIndex** s'utilise dans la majorité des cas avec l'instruction **IF** présentée plus loin dans le manuel.

■ **IntradayBarIndex** : Compteur de barres intraday.

Le compteur affiche le nombre de barres depuis le début de la journée et est réinitialisé à zéro tous les débuts de journée. La première barre du compteur est considérée comme la barre 0.

Comparons donc les deux constantes en créant deux indicateurs séparés :

`RETURN BarIndex`

et

`RETURN IntradayBarIndex`



On remarque bien la remise à zéro du compteur de barres tous les débuts de journée pour **IntradayBarIndex**.

Les constantes dérivées des prix

- **Range** : différence entre **High** et **Low**
• $\text{Range} = \text{High} - \text{Low}$
- **TypicalPrice** : moyenne entre **High**, **Low** et **Close**
• $\text{TypicalPrice} = (\text{High} + \text{Low} + \text{Close}) / 3$
- **WeightedClose** : moyenne pondérée de **High**, **Low**, et **Close**
• $\text{WeightedClose} = (\text{High} + \text{Low} + 2 * \text{Close}) / 4$
- **MedianPrice** : la moyenne entre **High** et **Low**
• $\text{MedianPrice} = (\text{High} + \text{Low}) / 2$
- **TotalPrice** : la moyenne entre **Open**, **High**, **Low** et **Close**
• $\text{TotalPrice} = (\text{Open} + \text{High} + \text{Low} + \text{Close}) / 4$

Le **Range** représente la volatilité de la barre courante.

Le **WeightedClose** insiste sur l'importance du cours de clôture.

Les constantes **TypicalPrice** et **TotalPrice** reflètent mieux la psychologie du marché intra-barre courante car elles prennent en compte trois et quatre niveaux de cours atteints pendant ce chandelier.

Le **MedianPrice** représente le prix moyen des extrêums.

Range en % :

```
pctRange = (Range / Range[1] - 1) * 100
```

```
RETURN pctRange
```

La constante **Undefined**

Undefined permet d'indiquer à l'indicateur de ne pas afficher un résultat pour certaines variables.

- **Undefined** : donnée indéfinie (équivalent d'une case vide).

Vous pouvez retrouver un exemple d'application plus loin dans le manuel.

Utilisation des indicateurs préexistants

Nous avons jusqu'à présent observé les possibilités offertes par ProBuilder en termes de constantes et leur comportement lors de l'accès aux barres du passé. Le même comportement s'applique au fonctionnement des indicateurs préexistants (et par la suite nous verrons que ceux que vous programmerez fonctionneront selon le même principe).

Les indicateurs ProBuilder se composent de trois éléments dont la syntaxe est :

NomDeFonction [calculé sur n barres] (sur telle variable)

Lorsque l'on utilise le bouton « Insérer Fonction » pour rechercher une fonction ProBuilder, des valeurs par défaut sont placées pour la période et pour l'argument prix ou indicateur. Exemple pour une moyenne mobile de 20 périodes :

Average[20](Close)

On peut bien sûr les modifier selon nos préférences ; par exemple, on peut remplacer les 20 barres définies par défaut par n'importe quel autre nombre de barres (exemple : **Average[10]**, **Average[15]**, **Average[30]**, ..., **Average[n]**). De même, on peut modifier **Close** par **Open** ou par indicateur comme par exemple le **RSI** (**Relative Strength Index**). Nous obtiendrons, par exemple :

Average[20](RSI[5](Close))

Nous calculons ainsi la moyenne mobile 20 périodes d'un RSI calculé sur 5 périodes

Regardons quelques exemples de comportements d'indicateurs préexistants :

Programme calculant la moyenne mobile exponentielle sur 20 bougies appliquée au prix de clôture :

RETURN ExponentialAverage[20](Close)

Calcul d'une moyenne mobile pondérée sur 20 barres appliquée au prix typique

RETURN WeightedAverage[20](TypicalPrice)

Calcul d'une moyenne mobile à lissage de Wilder sur 100 barres appliquée au Volume

RETURN WilderAverage[100](Volume)

Calcul du MACD (en histogramme) sur le prix de clôture.

La ligne du MACD se construit comme la différence entre la moyenne mobile exponentielle sur 12 périodes moins celle sur 26 périodes. On effectue ensuite un lissage avec une moyenne mobile exponentielle à 9 périodes, appliquée à la différence pour obtenir la ligne de Signal. Le MACD en histogramme se constitue alors de la différence entre la ligne du MACD et la ligne de Signal.

```
// Calcul de la ligne MACD
LigneMACD = ExponentialAverage[12](Close) - ExponentialAverage[26](Close)
// Calcul de la ligne du signal MACD
LigneSignal = ExponentialAverage[9](LigneMACD)
// Calcul de la différence entre la ligne du MACD et son Signal
MACDHistogramme = LigneMACD - LigneSignal
RETURN MACDHistogramme STYLE (HISTOGRAM)
```

Calcul d'une moyenne à deux paramètres

Vous avez aussi la possibilité de paramétriser la fonction moyenne (**Average**) avec un deuxième paramètre afin de choisir le type de moyenne mobile utilisée. Nous obtenons la formule suivante :

Average[Nbre de périodes, Type de moyenne]

Le paramètre Type de moyenne désigne, comme son nom l'indique, le type de moyenne qui sera sollicité. Elles sont au nombre de 9 et indexées de 0 à 8 :

| | | |
|-----------------|--------------------|---------------|
| 0=Simple | 4=Triangulaire | 8=Zero Retard |
| 1=Exponentielle | 5=Moindres Carrés | |
| 2=Pondérée | 6=Série Temporelle | |
| 3=Wilder | 7=Hull | |

Calcul des lignes Ichimoku

L'indicateur Ichimoku ayant de nombreuses lignes le représentant, certaines de ces lignes ont été introduites dans le langage ProBuilder pour vous permettre d'utiliser pleinement son potentiel.

Les lignes se déclinent de la manière suivante :

- `TenkanSen[TenkanPeriod, KijunPeriod, Senkou-SpanBPeriod]`
- `KijunSen[TenkanPeriod, KijunPeriod, Senkou-SpanBPeriod]`
- `SenkouSpanA[TenkanPeriod, KijunPeriod, Senkou-SpanBPeriod]`
- `SenkouSpanB[TenkanPeriod, KijunPeriod, Senkou-SpanBPeriod]`

Avec pour chaque ligne les paramètres habituels d'Ichimoku :

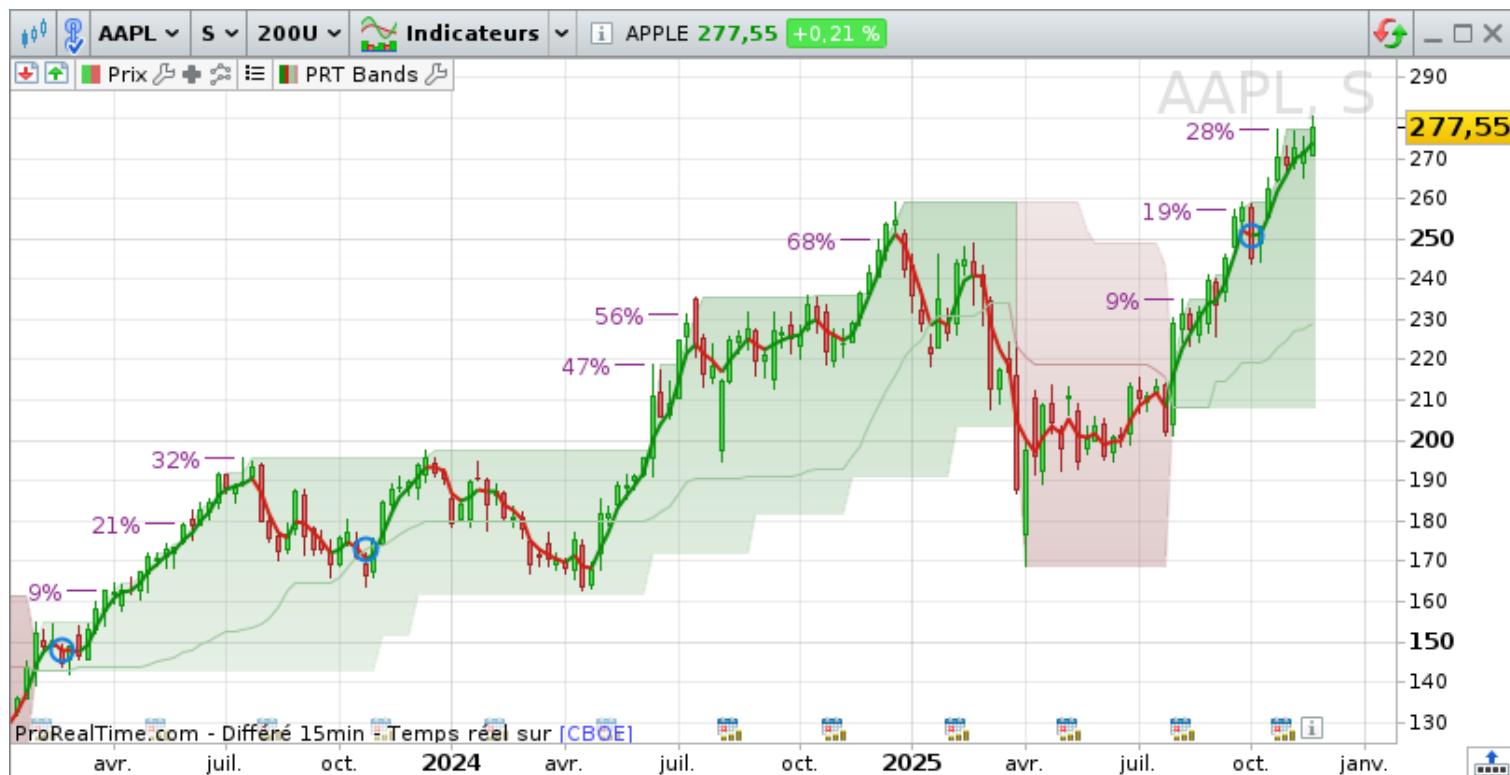
- `TenkanPeriod`: droite d'alerte, (point haut + point bas)/2 sur les ***n*** dernières périodes
- `KijunPeriod`: droite de signal, (point haut + point bas)/2 sur les ***n*** dernières périodes
- `Senkou-SpanBPeriod`: projection point moyen long terme, (point haut + point bas)/2 sur les ***n*** dernières périodes

Calcul des lignes PRT Bands

PRT Bands est un indicateur visuel qui simplifie la détection et le suivi des tendances. Il est exclusif à la plateforme ProRealTime.

Il peut vous aider à :

- détecter un retournement de tendances
- identifier et suivre une tendance haussière
- mesurer l'intensité de la tendance
- trouver des points potentiels d'entrée et de sortie



Voici les différentes données de PRT Bands disponibles dans le langage ProBuilder :

- **PRTBANDSUP**: renvoie la valeur de la ligne supérieure de l'indicateur
- **PRTBANDSDOWN**: renvoie la valeur de la ligne inférieure de l'indicateur
- **PRTBANDSSHORTTERM**: renvoie la valeur de la ligne (épaisse) court terme de l'indicateur
- **PRTBANDSMEDIUMTERM**: renvoie la valeur de la ligne (fine) moyen terme de l'indicateur

[En savoir plus sur l'indicateur PRT Bands](#)

Ajout de variables paramétrables

Lorsque l'on code un indicateur, on introduit un certain nombre de constantes. L'option de variables paramétrables, en haut à gauche, vous permet d'attribuer une valeur par défaut à une variable non définie et d'agir ensuite sur la valeur de cette variable à partir de l'interface des paramètres de l'indicateur.

L'avantage réside dans la possibilité de pouvoir modifier les paramètres de l'indicateur sans modifier le code.

Calculons par exemple une moyenne mobile de période 20 :

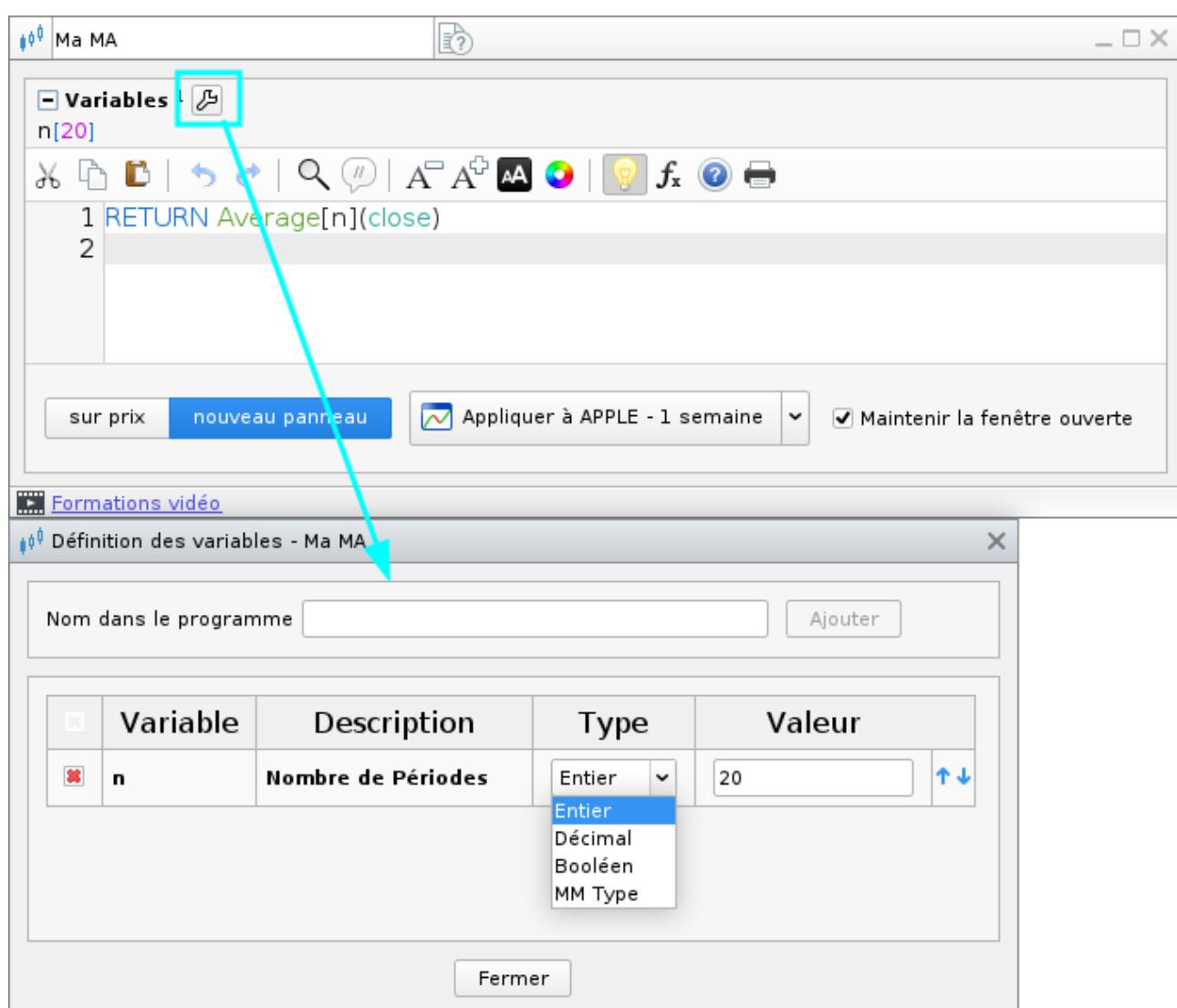
```
RETURN Average[20](Close)
```

Afin de pouvoir modifier le nombre de périodes de calcul directement à partir de l'interface, remplacez 20 par une variable 'n' :

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

Cliquez ensuite sur la clé à côté de « Variables » et vous verrez alors s'afficher la fenêtre « Définition des variables ».

Entrez le nom de votre variable, ici « n », et appuyez sur « Entrée » (ou cliquer « Ajouter »), vous pourrez ensuite renseigner un Type et une Valeur par défaut. Remplissez comme suit :



Dans la fenêtre de Propriété de l'indicateur, vous obtiendriez donc un nouveau paramètre, qui vous permettra d'agir sur la période de la moyenne mobile :



Liste des types disponibles pour les variables paramétrables :

- **Entier** : nombre entier compris entre -2 000 000 000 et 2 000 000 000 (ex : 450)
- **Décimale** : nombre décimal avec une précision de 5 chiffres significatifs (ex : 1.03247)
- **Booléen** : Vrai (1) ou Faux (0)
- **Type de moyenne mobile** : permet de définir la valeur du second paramètre pour le calcul de moyenne mobile de l'indicateur **Average** (voir plus haut).

Bien entendu, il est possible de créer plusieurs variables vous donnant ainsi la possibilité de modifier plusieurs paramètres en même temps.

Chapitre II : Fonctions et instructions ProBuilder

Structures de contrôle

Instruction conditionnelle IF

L'instruction **IF** sert à faire un choix d'actions conditionnées, c'est-à-dire à subordonner un résultat à la vérification d'une ou plusieurs conditions définies.

La structure se compose des éléments **IF**, **THEN**, **ELSE**, **ELSIF**, **ENDIF**, qui se combinent selon la complexité des conditions que nous voulons définir. Nous allons passer en revue leur mode d'emploi.

Une condition, un résultat (IF THEN ENDIF)

Nous avons la possibilité de rechercher une condition et de définir une action si la condition est vérifiée. En revanche, si la condition n'est pas satisfaite, rien ne se passe.

Dans l'exemple, si le dernier prix est supérieur à la MM de période 20, alors on affiche la valeur 1.

```
Result=0                                     Le Result est égal à 0.  
IF Close > Average[20](Close) THEN        Si le prix de clôture est > à la moyenne mobile de 20 périodes  
  Result = 1                                ALORS Result sera égal à 1  
ENDIF                                         FIN DE CONDITION  
RETURN Result
```

Une condition, deux résultats (IF THEN ELSE ENDIF)

Nous pouvons également choisir de définir un résultat au cas où la condition n'est pas vérifiée. Reprenons l'exemple précédent : si le dernier prix est supérieur à la MM de période 20, on affiche la valeur 1.

Autrement, on affiche -1.

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

NB : Nous venons de créer un indicateur binaire. Pour en savoir plus, voir la section sur les indicateurs binaires et ternaires plus loin dans ce manuel.

Conditions imbriquées

Il est possible de créer des sous-conditions à la suite de la validation d'une condition principale, c'est-à-dire des conditions qui doivent se vérifier l'une après l'autre (dans l'ordre de parution). Pour ce faire, il suffit d'imbriquer les **IF**, mais en faisant attention à insérer autant de **ENDIF** que de **IF**. Regardons l'exemple :

Double conditions sur moyennes mobiles :

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

Conditions Multiples (IF THEN ELSIF ELSE ENDIF)

Il est possible de définir plusieurs résultats associés chacun à une condition spécifique. L'indicateur reporte donc plusieurs états : si la Condition1 est vérifiée alors on active l'Action 1 ; autrement, si la Condition 2 est vérifiée, on active l'Action 2 ...si aucune condition n'est vérifiée, on active l'Action n.

Syntaxiquement, cette structure utilise les instructions : **IF, THEN, ELSIF, THEN ... ELSE, ENDIF**.

Elle s'écrit de la manière suivante :

```
IF (Condition1) THEN
    (Action1)
ELSIF (Condition2) THEN
    (Action2)
ELSIF (Condition3) THEN
    (Action3)
...
ELSE
    (Action n)
ENDIF
```

Il est possible, mais l'écriture est plus lourde, de remplacer les **ELSIF** par des **ELSE IF**. Il faudra alors terminer la série par autant d'**ENDIF** que d'**IF** écrits. Il vous est donc conseillé, si vous souhaitez imbriquer de multiples conditions dans votre programme, d'utiliser **ELSIF** plutôt que **ELSE IF**.

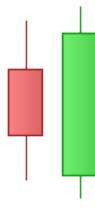
Exemple : détection des avalements haussiers et baissiers

Cet indicateur va retourner 1 si un avalement haussier est détecté, -1 si un avalement baissier est détecté et 0 le reste du temps.

```
// Description d'un avalement haussier
Condition1 = Close[1] < Open[1]
Condition2 = Open < Close[1]
Condition3 = Close > Open[1]
Condition4 = Open < Close

// Description d'un avalement baissier
Condition5 = Close[1] > Open[1]
Condition6 = Open > Close[1]
Condition7 = Close < Open[1]
Condition8 = Close < Open

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
```



Exemple : pivot Demarks Résistance

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

Exemple : BarIndex

Dans le chapitre I de ce manuel, **BarIndex** vous a été présenté comme compteur du nombre des barres depuis le début de l'historique affiché. **BarIndex** est souvent utilisé en association avec **IF**. Par exemple, si on cherche à savoir si notre graphique contient moins ou plus de 23 barres, on écrira :

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

Remarque : Ce code est montré à titre d'exemple pour décrire le fonctionnement de l'instruction **IF**. Mais il existe une façon plus simple et lisible d'obtenir les mêmes résultats :

```
RETURN BarIndex > 23
```

Boucle itérative FOR

La boucle **FOR** est utilisée lorsqu'on souhaite parcourir un par un, une liste finie et ordonnée de nombres (1,2,3,...,6,7 ou 7,6,...,3,2,1).

La structure se compose des mots-clefs **FOR**, **TO**, **DOWNT0**, **DO**, **NEXT**. L'utilisation de **TO** ou **DOWNT0** varie en fonction de l'appel en ordre croissant ou décroissant des éléments. Il est important de souligner que ce qui se trouve entre le **FOR** et le **DO** sont les bornes de l'intervalle à balayer.

Avancement croissant (**FOR**, **TO**, **DO**, **NEXT**)

```
FOR Variable = ValeurDeDebutDeSerie TO ValeurDeFinDeSerie DO
    (Action)
NEXT
```

Exemple : lissage de Moyenne Mobile de période 12 (MM12)

Nous allons créer une variable de stockage (Result) qui va sommer une par une chaque moyenne mobile, de période 11, 12 et 13.

```
Result = 0
FOR Variable = 11 TO 13 DO
    Result = Result + Average[Variable](Close)
NEXT
// Faisons la moyenne des moyennes mobiles en divisant Result par 3 et en stockant le résultat dans AverageResult.
AverageResult = Result / 3
RETURN AverageResult
```

Visualisons ce qui se passe étape par étape :

Mathématiquement, on veut faire la moyenne des moyennes mobiles arithmétiques de période 11, 12 et 13.

Variable va donc prendre successivement les valeurs 11, 12 puis 13

```
Result = 0
Variable = 11
```

Result reçoit la valeur du précédent **Result** + MM11 soit : $(0) + MM11 = (0 + MM11)$

L'instruction **NEXT** nous fait passer à la valeur suivante du compteur

```
Variable = 12
```

Result reçoit la valeur du précédent **Result** + MM12 soit : $(0 + MM11) + MM12 = (0 + MM11 + MM12)$

L'instruction **NEXT** nous fait passer à la valeur suivante du compteur

```
Variable = 13
```

Result reçoit la valeur du précédent **Result** + MM13 soit : $(0 + MM11 + MM12) + MM13 = (0 + MM11 + MM12 + MM13)$

La valeur 13 est la dernière valeur du compteur.

NEXT ferme la boucle **FOR** car il n'y a plus de valeur suivante.

On affiche **Result** / 3

Ce code signifie tout simplement que **Variable** va dans un premier temps prendre la valeur de début de série, puis **Variable** prendra la valeur suivante (la précédente + 1) et ainsi de suite jusqu'à ce que **Variable** dépasse ou soit égale à la valeur de fin de la série. Alors, la boucle se termine.

Exemple : Moyenne sur les 5 dernières barres des plus hauts

```

SUMhigh = 0
IF BarIndex < 4 THEN
    MMhigh = Undefined
ELSE
    FOR i = 0 TO 4 DO
        SUMhigh = High[i] + SUMhigh
    NEXT
ENDIF
MMhigh = SUMhigh / 5
RETURN MMhigh

```

S'il n'y a pas plus de 5 périodes sur l'historique
Alors on attribue à MMhigh la valeur par défaut « rien »
Sinon
Pour les valeurs entre 0 et 4
On somme les 5 dernières valeurs des plus hauts
On moyenne cette somme par 5 et on l'affecte à MMhigh
On affiche MMhigh

Remarque : Ce code est montré a titre d'exemple pour décrire le fonctionnement de la boucle **FOR**. Mais il est important de retenir qu'il faut éviter d'utiliser les boucles **FOR** lorsque d'autres alternatives existent. Par exemple, ce code est plus lisible, donne les mêmes résultats et est plus optimisé. Il s'exécutera donc beaucoup plus rapidement :

Return Average[5](High)

Avancement décroissant (FOR, DOWNT0, DO, NEXT)

L'avancement décroissant utilise en revanche les instructions : **FOR, DOWNT0, DO, NEXT**.

Elle s'écrit de la manière suivante :

```

FOR Variable = ValeurDeFinDeSérie DOWNT0 ValeurDeDébutDeSérie DO
    (Action)
NEXT

```

Reprendons l'exemple de la moyenne mobile sur les 5 dernières barres des prix les plus hauts :

On remarquera que l'on a juste inversé les bornes de l'intervalle balayé.

```

SUMhigh = 0
IF BarIndex < 4 THEN
    MMhigh = Undefined
ELSE
    FOR i = 4 DOWNT0 0 DO
        SUMhigh = High[i] + SUMhigh
    NEXT
ENDIF
MMhigh = SUMhigh / 5
RETURN MMhigh

```

Boucle conditionnelle WHILE

La boucle **WHILE** sert à appliquer des actions tant qu'une condition reste valide. Vous verrez que cette boucle a de grandes similitudes avec l'instruction conditionnelle simple **IF/THEN/ENDIF**.

Syntaxiquement, cette structure utilise les instructions : **WHILE**, **(DO** facultatif), **WEND**

La structure s'écrit de la manière suivante :

```
WHILE (Condition) DO
  (Action 1)
  ...
  (Action n)
WEND
```

Ce code permet de mettre en évidence le nombre de barres séparant le chandelier actuel d'un chandelier précédent plus élevé, dans une limite de 30 périodes.

```
i = 1
WHILE high > high[i] AND i < 30 DO
  i = i + 1
WEND
RETURN i
```

Exemple : indicateur calculant le nombre de périodes de hausse consécutives

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

Remarque : Ce code est montré à titre d'exemple pour décrire le fonctionnement de la boucle **WHILE**. Mais il est important de retenir qu'il faut éviter d'utiliser les boucles **WHILE** lorsque d'autres alternatives existent. Par exemple, ce code est beaucoup plus optimisé, s'exécute plus rapidement et donne les même résultats. Ce code utilise habilement le fait que le code ProBuilder est exécuté sur chaque chandelier historique : incrémenter un compteur à chaque fois que la condition est vérifiée fonctionne aussi bien que regarder dans le passé à chaque bougies, par contre, c'est beaucoup plus efficient.

```
Increase = Close > Close[1]
IF Increase THEN
  Count = Count + 1
ELSE
  Count = 0
ENDIF
RETURN Count
```

BREAK

L'instruction **BREAK** permet de faire une sortie forcée d'une boucle **WHILE** ou d'une boucle **FOR**. Des combinaisons avec la commande **IF** sont possibles, que ce soit dans une boucle **WHILE** ou dans une boucle **FOR**.

BREAK avec WHILE

Lorsqu'on cherche à sortir d'une boucle conditionnelle **WHILE**, on utilise **BREAK** de la manière suivante :

```
WHILE (Condition) DO
    (Action)
    IF (ConditionBreak) THEN
        BREAK
    ENDIF
WEND
```

L'utilisation de **BREAK** dans une boucle **WHILE** n'a d'intérêt que si l'on veut tester une condition supplémentaire dont la valeur ne peut être connue que dans le corps de la boucle **WHILE**. Prenons pour exemple un stochastique basé sur oscillateur qui n'est calculé qu'en tendance haussière :

```
ret = 0
Increase = Close > Close[1]
i = 0
WHILE Increase[i] or BarIndex > 0 DO
    i = i + 1
    // Si high - low, on sort de la boucle pour éviter une division par zéro.
    IF high[i] = low[i] Then
        BREAK
    ENDIF
    osc = (close - low) / (high - low)
    ret = AVERAGE[i](osc)
WEND
RETURN ret
```

BREAK avec **FOR**

Lorsqu'on cherche à sortir d'une boucle itérative **FOR**, sans arriver à la dernière valeur de la série, on utilise **BREAK** de cette manière :

```
FOR Variable = ValeurDeDebutDeSerie TO ValeurDeFinDeSerie DO
    (Action)
    BREAK
NEXT
```

Prenons pour exemple un indicateur cumulant le nombre de hausses consécutives du volume dans les 19 dernières barres. Cet indicateur rendra zéro si le volume est baissier.

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

Dans ce code, si on n'avait pas utilisé **BREAK**, la boucle aurait continué jusqu'à 19 (dernier élément de la série) même si la condition de volume n'est pas valide.

Avec **BREAK**, en revanche, dès que la condition n'est plus validée, il retourne le résultat.

CONTINUE

L'instruction **CONTINUE** permet de finir l'itération courante d'une boucle **WHILE** ou **FOR**. Il est souvent utilisé en association avec **BREAK**, pour donner l'ordre soit de sortir de la boucle (**BREAK**) soit d'y rester (**CONTINUE**).

CONTINUE avec WHILE

Créons un programme cumulant le nombre de chandeliers ayant une clôture supérieure et une ouverture inférieure à celles de la veille. Si la condition n'est pas vérifiée, le compteur reviendra à zéro.

```
Increase = Close > Close[1]
condition = Open > Open[1]
Count = 0
WHILE condition[Count] DO
    IF Increase[Count] THEN
        Count = Count + 1
    CONTINUE
    ENDIF
    BREAK
WEND
RETURN Count
```

Grâce à **CONTINUE**, lorsque la condition du **IF** est vérifiée, on ne sort pas de la boucle **WHILE**, ce qui permet de cumuler le nombre de chandeliers vérifiant cette condition. Sans l'instruction **CONTINUE**, le programme sortirait de la boucle, que la condition du **IF** soit vérifiée ou non. On ne pourrait donc pas cumuler les apparitions des conditions et le résultat serait binaire (1,0).

CONTINUE avec FOR

Créons un programme cumulant le nombre de chandeliers ayant une clôture supérieure à la veille. Si la condition n'est pas vérifiée, le compteur reviendra à zéro.

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

FOR permet de tester la condition sur tout l'historique disponible. Grâce à **CONTINUE**, lorsque la condition du **IF** est vérifiée, on ne sort pas de la boucle **FOR** et on continue avec la valeur du **i** suivant. Ceci permet de cumuler le nombre de figures vérifiant cette condition.

Sans l'instruction **CONTINUE**, le programme sortirait de la boucle, que la condition du **IF** soit vérifiée ou non. On ne pourrait donc pas cumuler les apparitions de figures et le résultat serait binaire (1,0).



Il est important de toujours s'assurer d'avoir une condition de sortie valide pour les boucles de type **FOR** et **WHILE** pour garantir le bon fonctionnement de votre code et éviter les boucles infinies.

ONCE

L'instruction **ONCE** sert à initialiser « **une seule fois** » une variable à une valeur donnée.

Sachant que pour tout programme, le langage va lire autant de fois le code qu'il y a de barres sur le graphique avant de retourner un résultat, il faudra donc retenir que :

- **ONCE** n'est traité par le programme que la première fois que l'instruction est rencontrée.
- Les fois suivantes, l'instruction sera ignorée.
- Il est aussi possible d'utiliser **ONCE** dans des instructions **IF**, **FOR** et **WHILE**.

Pour bien comprendre comment fonctionne cette commande, il faut percevoir la manière dont le langage lit le code; d'où l'utilité de l'exemple suivant.

Voici deux programmes qui retournent respectivement 0 et 15 et dont la seule différence est l'ajout de la commande **ONCE** :

Programme 1

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

Programme 2

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

Voyons voir comment le langage a lu les codes.

Programme 1 :

Le langage va lire L1 (Count = 0 ; i = 0), puis L2, L3, L4, L5 et L6 (Count = 0 ; i = 1), revient à L1 et relira tout exactement de la même façon. Le résultat affiché est 0 (zéro), comme à la suite de la première lecture.

Programme 2 :

Le langage va lire L1 (Count = 0 ; i = 0), puis L2, L3, L4, L5, L6 (Count = 0 ; i = 1) ; arrivé à la ligne de **RETURN**, il recommence la boucle depuis L3 (**les lignes avec ONCE ne sont traitées que la première fois**), L4, L5, L6 (Count = 1 ; i = 2), puis revient à nouveau (Count = 3 ; i = 3) et ainsi de suite jusqu'à (Count = 15 ; i = 6). Arrivé à ce résultat, l'instruction **IF** n'est plus traitée car la condition ne vaut plus; ne lui restera à lire que L7. D'où le résultat : 15.

Fonctions Mathématiques

Fonctions usuelles unaires et binaires

Intéressons-nous maintenant aux fonctions mathématiques.

À noter que *a* et *b* sont des exemples d'arguments décimaux. Ils peuvent être remplacés par n'importe quelle variable dans votre programme.

- **MIN**(*a*, *b*) : calcule le minimum de *a* et de *b*
- **MAX**(*a*, *b*) : calcule le maximum de *a* et de *b*
- **ROUND**(*a*, *n*) : calcule un arrondi à l'unité de *a* avec une précision de *n* chiffres après la virgule
- **ABS**(*a*) : calcule la valeur absolue de *a*
- **SGN**(*a*) : donne le signe de *a* (1 pour positif, -1 pour négatif, 0 si nulle)
- **SQUARE**(*a*) : calcule le carré de *a*
- **SQRT**(*a*) : calcule la racine carrée de *a*
- **LOG**(*a*) : calcule le logarithme népérien de *a*
- **POW**(*a*,*b*) : calcule *a* à la puissance *b*
- **EXP**(*a*) : calcule l'exponentiel de *a*
- **COS**(*a*) / **SIN**(*a*) / **TAN**(*a*) : calcule le cosinus/sinus/tangente de *a* (en degrés)
- **ACOS**(*a*) / **ASIN**(*a*) / **ATAN**(*a*) : calcule l'arc-cosinus/arc-sinus/arc-tangente (en degrés) de *a*
- **FLOOR**(*a*, *n*) : retourne le plus grand nombre rond inférieur à *a* avec une précision de *n*
- **CEIL**(*a*, *n*) : retourne le plus petit nombre rond supérieur à *a* avec une précision de *n*
- **RANDOM**(*a*,*b*) : génère un nombre entier aléatoire compris entre *a* et *b* (inclus)

Codons par exemple la loi mathématique normale, intéressante car elle utilise à la fois la mise au carré, la mise en racine carrée et l'exponentielle :

```
// Loi Normale appliquée en x = 10, Ecart-type = 6 et Espérance = 8
// Posons en variable optimisée :
Ecarttype = 6
Esperance = 8
x = 10
Indicator = EXP( - (1 / 2) * SQUARE(x - Esperance) / Ecarttype) /
(Ecarttype * SQRT(2 / 3.1415 ))
RETURN Indicator
```

Opérateurs mathématiques usuels

- ***a* < *b*** : *a* est strictement inférieur à *b*
- ***a* <= *b* ou *a* =< *b*** : *a* est inférieur ou égal à *b*
- ***a* > *b*** : *a* est strictement supérieur à *b*
- ***a* >= *b* ou *a* => *b*** : *a* est supérieur ou égal à *b*
- ***a* = *b*** : *a* est égal à *b* (ou *a* reçoit la valeur *b*)
- ***a* <> *b*** : *a* est différent de *b*

Fonctions de comparaisons graphiques

- ***a* CROSSES OVER *b*** : la courbe de *a* franchit la courbe de *b* à la hausse
- ***a* CROSSES UNDER *b*** : la courbe de *a* franchit la courbe de *b* à la baisse

Fonctions de sommation

- **cumsum** : Calcule la somme de toutes les barres du graphique

La syntaxe d'utilisation de **cumsum** est :

cumsum(prix ou indicateur)

Par exemple : **CUMSUM(Close)** donne la somme des **Close** de chaque bougie.

- **summation** : Calcule la somme sur un nombre de barres à définir

La somme est effectuée à partir de la barre la plus récente (de droite à gauche)

La syntaxe d'utilisation de **summation** est :

summation[nombre de barres](prix ou indicateur)

Par exemple : **SUMMATION[20](Open)** donne la somme des **Open** des 20 dernières bougies.

Fonctions Statistiques

La syntaxe d'utilisation de ces fonctions est la même que celle des indicateurs et de la fonction **Summation** à savoir :

lowest[nombre de barres](prix ou indicateur)

- **Lowest** : donne la valeur la plus basse sur la période définie
- **Highest** : donne la valeur la plus élevée sur la période définie
- **STD** : donne l'écart-type sur une valeur pour une période définie
- **STE** : donne l'erreur-type sur une valeur pour une période définie

Opérateurs logiques

De même, comme tout langage informatique, il est nécessaire d'avoir à disposition des opérateurs logiques afin de créer des indicateurs pertinents. Vous trouverez ci-dessous les 4 opérateurs logiques de ProBuilder :

- **NOT a** : NON logique
- **a OR b** : OU logique
- **a AND b** : ET logique
- **a XOR b** : OU exclusif (a OU b mais pas a ET b)

Mots-clefs ProBuilder

- **RETURN** : affiche le résultat de votre indicateur
- **CALL** : appelle une fonction précédemment créée par l'utilisateur
- **AS** : nomme les différents résultats affichés
- **COLOURED** : colorie le tracé affiché d'une couleur à définir

RETURN

Nous avons déjà pu voir dans le premier chapitre, l'importance de l'instruction **RETURN**. Elle a des propriétés particulières qu'il faut connaître pour éviter certaines erreurs de programmation.

Pour une utilisation correcte dans l'écriture d'un programme, **RETURN** s'utilise :

- Une seule et unique fois
- A la dernière ligne de code
- Optionnellement avec d'autres fonctions telles qu'**AS**, **COLOURED** et **STYLE**
- Pour afficher plusieurs résultats, on écrit **RETURN** suivi par les résultats qu'on veut afficher séparés par une virgule (exemple : **RETURN** a,b)

Commentaires

// ou //** permettent de placer dans le code des commentaires. Elles servent principalement à vous souvenir comment fonctionne une fonction que vous auriez codée. Ces remarques seront lues mais évidemment pas traitées par le code. Illustrons l'idée par l'exemple suivant :

```
// ce programme retourne la moyenne mobile arithmétique de période 20 sur le prix de clôture
```

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



N'utilisez pas les caractères spéciaux (exemples : é,ù,ç,ê,-,_,&...) dans ProBuilder (cela ne s'applique pas aux commentaires).

CustomClose

CustomClose est une variable qui permet d'afficher les constantes **Close**, **Open**, **High**, **Low** et d'autres valeurs, qui peuvent être sélectionnées dans la fenêtre des propriétés de l'indicateur.

Sa syntaxe d'utilisation est la même que les constantes de prix qui s'adaptent à la vue du graphique :

CustomClose[n]

Prenons un exemple simple :

```
RETURN CustomClose[2]
```

En cliquant sur « Configurer » depuis l'étiquette de prix sur le coin supérieur gauche du graphique vous verrez qu'il est possible de configurer les prix utilisés pour le calcul.

CALCULATEONLASTBARS

Ce paramètre permet d'augmenter la vitesse à laquelle un indicateur sera calculé en définissant le nombre de barres utilisables pour le calcul de l'indicateur. L'affichage se fera en commençant par la barre la plus récente.

Exemple : **DEPPARAM CALCULATEONLASTBARS = 200**

Attention : l'utilisation de l'instruction **DEPPARAM** doit se faire obligatoirement au début du code.

CALL

CALL permet d'appeler un indicateur personnalisé déjà présent sur votre plateforme.

Le moyen le plus rapide consiste à sélectionner directement à partir de la catégorie « Indicateurs utilisateur » (dans le menu « Insérer fonction ») l'indicateur à employer.

Imaginons que vous avez codé sous le nom de **HistoMACD** l'indicateur du MACD en histogramme.

Sélectionnez votre indicateur et cliquez sur « Ajouter » et dans la zone de programmation apparaîtra :

```
myHistoMACD = CALL "HistoMACD"
```

ProBuilder a donné le nom « **myHistoMACD** » à la variable représentant votre indicateur **"HistoMACD"**

Voici un exemple lorsque plusieurs variables sont renvoyées par votre **CALL** :

```
myExponentialMovingAverage, mySimpleMovingAverage = CALL "Averages"
```

AS

Le mot-clé **AS** sert à nommer le résultat affiché. Cette instruction est utilisée avec **RETURN** selon la structure suivante :

```
RETURN Result1 AS "Curve Name1", Result2 AS "Curve Name2", ...
```

Ce mot-clé facilite l'identification des éléments composants l'indicateur créé.

Exemple :

```
a = ExponentialAverage[200](Close)
```

```
b = WeightedAverage[200](Close)
```

```
c = Average[200](Close)
```

```
RETURN a AS "Exponential Average", b AS "Weighted Average", c AS "Arithmetic Average"
```

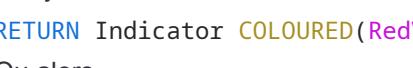
COLOURED

COLOURED est utilisé après la commande **RETURN** pour colorier la valeur affichée d'une certaine couleur, définie selon la norme RGB (Red, Green, Blue) ou en utilisant des couleurs pré-définies.

Les 140 couleurs pré-définies peuvent être retrouvées dans la documentation suivante :

[W3 School : HTML Color Names](#)

On peut donner les principales couleurs de la norme RGB ainsi que leur nom HTML pré-défini :

| COULEUR | VALEUR RGB (entre 0 et 255) (ROUGE, VERT, BLEU) | HTML Color Name |
|---|--|-----------------|
|  | (0, 0, 0) | black |
|  | (255, 255, 255) | white |
|  | (255, 0, 0) | red |
|  | (0, 255, 0) | green |
|  | (0, 0, 255) | blue |
|  | (255, 255, 0) | yellow |
|  | (0, 255, 255) | cyan |
|  | (255, 0, 255) | magenta |

La syntaxe d'utilisation de la commande **COLOURED** est la suivante :

`RETURN Indicator COLOURED(RedValue, GreenValue, BlueValue)`

Ou alors

`RETURN Indicator COLOURED("cyan")`

De manière optionnelle, vous pouvez contrôler l'opacité de votre courbe avec le paramètre alpha (compris entre 0 et 255) :

`RETURN Indicator COLOURED(RedValue, GreenValue, BlueValue, AlphaValue)`

La commande **AS** peut être associée à la commande **COLOURED(., ., .)** :

`RETURN Indicator COLOURED(RedValue, GreenValue, BlueValue) AS "Nom De Ma Courbe"`

Reprends l'exemple précédent et insérons **COLOURED** à la ligne du "**RETURN**".

`a = ExponentialAverage[200](Close)`

`b = WeightedAverage[200](Close)`

`c = Average[200](Close)`

`RETURN a COLOURED("red") AS "Exponential Moving Average", b COLOURED("green") AS "WeightedMoving Average", c COLOURED("blue") AS "Simple Moving Average"`

L'image vous montre la personnalisation des couleurs dans le résultat.



Commandes de dessin

Ces commandes vous permettent de dessiner des objets sur les graphiques mais aussi de personnaliser vos bougies, les barres de vos graphiques ainsi que les couleurs de l'ensemble de ces éléments.

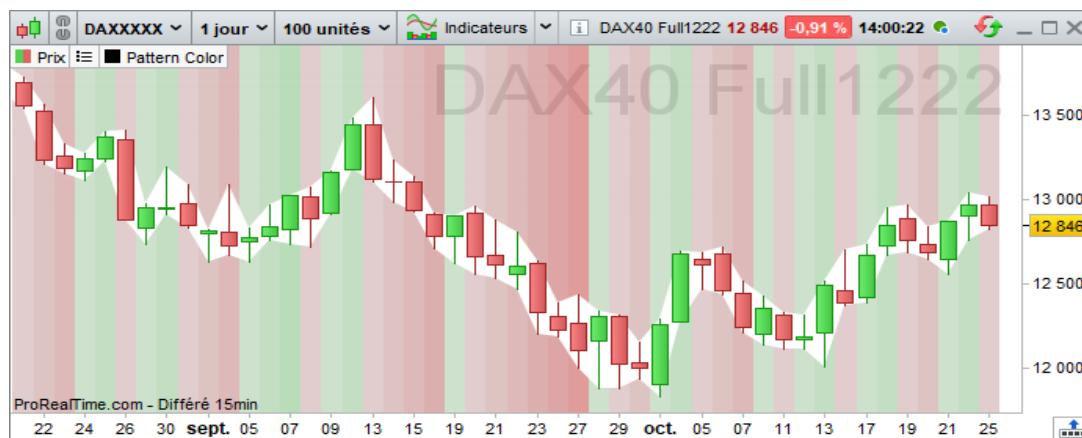
Pour chaque instruction ci-dessous, la couleur peut être définie de manière similaire à la couleur de votre courbe (instruction **COLOURED** ci-dessus) avec soit une couleur prédéfinie (HTML Color Names) entre guillemets, soit un triplet (R,V,B) sur lesquelles vous pouvez appliquer un paramètre d'opacité alpha : ("HTML Color Names",alpha) ou (R,V,B,alpha).

- **BACKGROUNDCOLOR(R,V,B,a)** : Vous permet de colorer l'arrière-plan des graphiques ou des barres spécifiques (comme les jours pairs ou impairs). La zone colorée commence à mi-chemin entre la barre précédente et la barre suivante.

Exemple : `BACKGROUNDCOLOR(0, 127, 255, 25)`

Il est possible d'utiliser une variable pour les couleurs si vous voulez que la couleur de fond change en fonction de vos conditions.

Exemple : `BACKGROUNDCOLOR(0, color, 255, 25)`



- **COLORBETWEEN** : Vous permet de remplir l'espace entre deux valeurs d'une certaine couleur.

Exemple : `COLORBETWEEN(Open, Close, "white")`

- **DRAWBARCHART** : Dessine une barre personnalisée sur le graphique. Open, high, low et close peuvent être des constantes ou des variables.

Exemple : `DRAWBARCHART(Open, High, Low, Close) COLOURED (0, 255, 0)`

- **DRAWCANDLE** : Dessine un chandelier personnalisé sur le graphique. Open, high, low et close peuvent être des constantes ou des variables.

Exemple : `DRAWCANDLE(Open, High, Low, Close) COLOURED ("black")`



Pour toutes les instructions de dessin ci-dessous, l'axe x est exprimé par défaut en numéro de barre (**BARINDEX**) et l'axe y correspond à l'échelle verticale des valeurs de votre graphique. Vous pouvez cependant changer ce comportement avec le mot clé **ANCHOR** décrit plus loin.

■ **DRAWARROW** : Dessine une flèche pointant vers la droite. Vous devez définir un point pour la flèche (axes x et y). Vous pouvez également choisir une couleur.

Exemple : `DRAWARROW(x1, y1) COLOURED (R, V, B, a)`

■ **DRAWARROWUP** : Dessine une flèche pointant vers le haut. Vous devez définir un point pour la flèche (axes x et y). Vous pouvez également choisir une couleur.

Exemple : `DRAWARROWUP(x1, y1) COLOURED (R, V, B, a)`

Ceci est utile pour ajouter des signaux d'achat visuels.

■ **DRAWARROWDOWN** : Dessine une flèche pointant vers le bas. Vous devez définir un point pour la flèche (axes x et y). Vous pouvez également choisir une couleur.

Exemple : `DRAWARROWDOWN(x1, y1) COLOURED (R, V, B, a)`

Ceci est utile pour ajouter des signaux de vente visuels.



■ **DRAWRECTANGLE** : Dessine un rectangle sur le graphique.

Exemple : `DRAWRECTANGLE(x1, y1, x2, y2) COLOURED (R, V, B, a)`

■ **DRAWTRIANGLE** : Dessine un triangle sur le graphique.

Exemple : `DRAWTRIANGLE(x1, y1, x2, y2, x3, y3) COLOURED (R, V, B, a)`

■ **DRAWELLIPSE** : Dessine une ellipse sur le graphique.

Exemple : `DRAWELLIPSE(x1, y1, x2, y2) COLOURED (R, V, B, a)`



• **DRAWPOINT** : Dessine un point sur le graphique.

Exemple : `DRAWPOINT (x1, y1, pointSize) COLOURED (R, V, B, a)`

• **DRAWLINE** : Dessine une ligne sur le graphique.

Exemple : `DRAWLINE (x1, y1, x2, y2) COLOURED (R, V, B, a)`

• **DRAWHLINE** : Dessine une ligne horizontale sur le graphique.

Exemple : `DRAWHLINE (y1) COLOURED (R, V, B, a)`

• **DRAWVLINE** : Dessine une ligne verticale sur le graphique.

Exemple : `DRAWVLINE (x1) COLOURED (R, V, B, a)`

• **DRAWSEGMENT** : Dessine un segment sur le graphique.

Exemple : `DRAWSEGMENT (x1, y1, x2, y2) COLOURED (R, V, B, a)`

Exemple : `DRAWSEGMENT (Barindex, Close, Barindex[5], Close[5])`



• **DRAWRAY** : Dessine une demi droite sur le graphique

Exemple : `DRAWRAY(x1, y1, x2, y2)`

• **DRAWTTEXT** : Ajoute un champ de texte au graphique avec le texte de votre choix à un emplacement spécifié. Ce texte peut-être configuré à l'aide de différents paramètres de style.

La syntaxe `#MaVariable#` vous permet d'afficher la valeur de MaVariable dans un texte.

Exemple : `DRAWTTEXT(value, x1, y1, font, fontStyle, fontSize) COLOURED (R, V, B, a)`

Exemple : `DRAWTTEXT ("Votre texte", x1, y1, SERIF, BOLD, 10) COLOURED (R, V, B, a)`

Example: `DRAWTTEXT ("Ma variable est #MaVariable#", x1, y1) COLOURED ("green")`



- Voici les différentes valeurs possibles pour les paramètres de **police** et **style de police**, la taille de la police est comprise en 1 et 30 :

| Police | Style de police |
|------------|-----------------|
| DIALOG | STANDARD |
| MONOSPACED | BOLD |
| SANSERIF | BOLDITALIC |
| SERIF | ITALIC |

- DRAWONLASTBARONLY** : Paramètre qui vous permet de dessiner des objets durant la dernière barre seulement. Ce paramètre peut être utilisé avec « **CALCULATEONLASTBARS** » pour optimiser les calculs.

Exemple : `DEFPARAM DRAWONLASTBARONLY = true`

Les paramètres supplémentaires

Pour certaines de ces commandes de dessin, on peut appliquer différentes instructions supplémentaires sans ordre précis :

BORDERCOLOR

Cette instruction permet de définir la couleur de la bordure d'un objet dessiné (hors lignes et flèches).

Exemple 1 : `DRAWRECTANGLE(Barindex, Close, Barindex[5], Close[5]) BORDERCOLOR(r,g,b,a)`

Exemple 2 : `DRAWRECTANGLE(Barindex, Close, Barindex[5], Close[5]) BORDERCOLOR("red")`

ANCHOR

Cette instruction permet de définir le point d'ancre de l'objet lorsque l'on souhaite le dessiner à partir d'une référence autre que les chandeliers.

`DRAWTEXT(Close,n,p) ANCHOR(referencePoint, horizontalShift, verticalShift)`

Elle peut prendre plusieurs valeurs en paramètres :

- Paramètre 1** : la position de l'ancre

| Valeur | Description |
|-------------|-------------------------------------|
| TOPLEFT | Fixé en haut à gauche du graphique |
| TOP | Fixé en haut du graphique (milieu) |
| TOPRIGHT | Fixé en haut à droite du graphique |
| RIGHT | Fixé à droite du graphique (milieu) |
| BOTTOMRIGHT | Fixé en bas à droite du graphique |
| BOTTOM | Fixé en bas du graphique (milieu) |
| BOTTOMLEFT | Fixé en bas à gauche du graphique |
| LEFT | Fixé à gauche du graphique (milieu) |
| MIDDLE | Fixé au centre du graphique |

• **Paramètre 2** : le type de valeur pour régler le positionnement sur l'axe horizontal

• **INDEX** : Les valeurs renseignées dans le dessin de l'objet pour l'axe horizontal feront référence au barindex des chandeliers

• **XSHIFT** : Les valeurs renseignées dans le dessin de l'objet pour l'axe horizontal feront référence à une valeur de décalage en pixel (positif ou négatif par rapport à un repère orthonormé)

• **Paramètre 3** : le type de valeur pour régler le positionnement sur l'axe vertical

• **VALUE** : Les valeurs renseignées dans le dessin de l'objet pour l'axe vertical feront référence à un prix

• **YSHIFT** : Les valeurs renseignées dans le dessin de l'objet pour l'axe vertical feront référence à une valeur de décalage en pixel (positif ou négatif par rapport à un repère orthonormé)

Exemples :

• `DRAWTEXT("Ma Variable vaut : #Var#", -20, -50) ANCHOR(TOPRIGHT, XSHIFT, YSHIFT)`

Affiche la valeur de la variable Var en haut à droite du graphique avec un décalage de -20 sur l'axe horizontal et -50 sur l'axe vertical.

• `DRAWTEXT("Top", Barindex-10, -20) ANCHOR(TOP, INDEX, YSHIFT)`

Dessine le texte "Top" en haut du graphique avec un décalage de -20 sur l'axe vertical et positionné dans la continuité du 10^{ème} barindex avant le dernier.

STYLE

Cette instruction permet de définir un style pour les objets (hors flèches) ou pour les valeurs retournées.

`DRAWRECTANGLE(x1, y1, x2, y2) STYLE(style, lineWidth)`

Il existe différents styles :

• **DOTTEDLINE**: ce style transforme la ligne en ligne pointillée, il y a 5 configurations différentes qui représentent 5 longueurs de pointillé différentes : **DOTTEDLINE**, **DOTTEDLINE1**, **DOTTEDLINE2**, **DOTTEDLINE3**, **DOTTEDLINE4**

• **LINE**: ce style rétabli le style de ligne par défaut (ligne pleine)

• **HISTOGRAM**: ce style, uniquement applicable dans l'instruction **RETURN** d'un indicateur, affiche les valeurs retournées sous forme d'histogramme.

• **POINT**: ce style, uniquement applicable dans l'instruction **RETURN** d'un indicateur, affiche les valeurs retournées sous forme de point.

• **lineWidth** qui définit l'épaisseur du trait, prendra une valeur comprise entre 1 (le plus fin) et 5 (le plus épais).



Remarque : pour les fonctions de dessins il est possible de préciser une date plutôt qu'un index de chandelier grâce à la fonction **DateToBarIndex** qui permet de transformer une date vers le barindex associé le plus proche.

L'instruction s'écrit sous la forme suivante :

DateToBarIndex(Date)

Les formats de date attendus :

- AAAA/ Exemple : 2022
- AAAAMM / Exemple : 202208
- AAAAMJJ / Exemple : 20220815
- AAAAMJJHH / Exemple : 2022081517
- AAAAMJJHHMM / Exemple : 202208151730
- AAAAMJJHHMMSS / Exemple : 20220815173020

Les instructions multi-périodes

ProBuilder vous permet de travailler sur différentes périodes lors de vos Backtests, Indicateurs et ProScreener, vous donnant ainsi accès à des données plus complètes lors de la conception de vos codes. L'instruction est structurée de la manière suivante :

TIMEFRAME(X UnitéDeTemps , Mode)

Avec comme paramètres :

- **UnitéDeTemps** : Le type de période choisi (voir [Liste des périodes disponibles](#))
- **X** : La valeur associée à la période choisi
- **Mode** : Le mode de calcul choisi (optionnel)

Exemple : **TIMEFRAME(1 Hour)**

Il vous est possible d'utiliser les instructions multi-timeframe uniquement pour appeler des unités de temps supérieures à votre unité de temps de base(unité de temps du graphique).

Les unités de temps secondaires appelées doivent aussi être un **multiple** de l'unité de temps de base.

Ainsi sur un graphique 10 minutes:

On peut appeler une unité de temps 20 minutes, 1 heure, 1 jour.

On ne peut pas appeler une unité de temps 5 minutes ou 17 minutes.

Pour entrer dans une unité de temps supérieure il faut donc utiliser l'instruction:

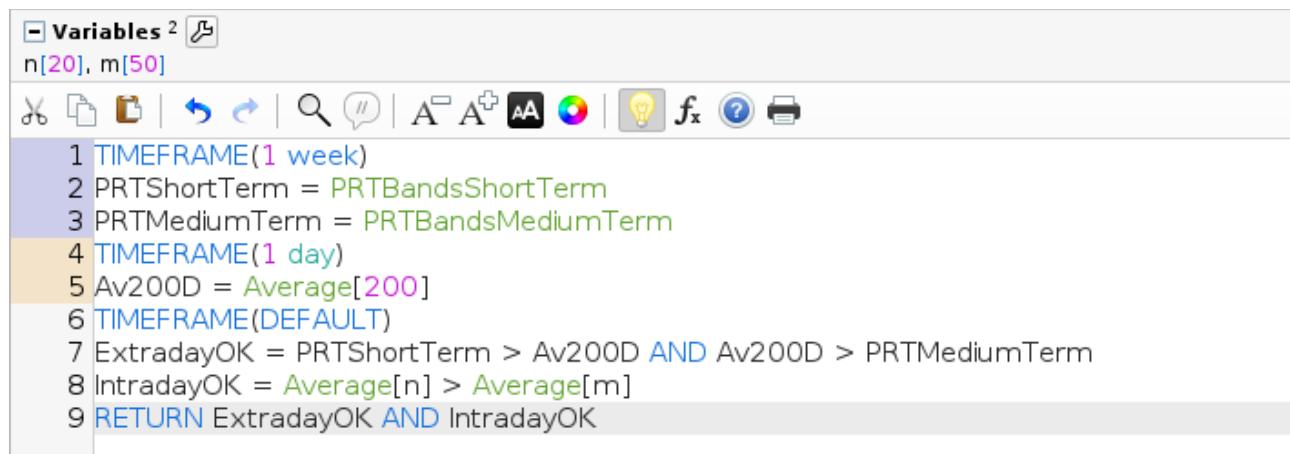
TIMEFRAME(X UnitéDeTemps)

Pour revenir sur l'unité de temps de base du graphique, on utilisera la commande suivante:

TIMEFRAME(DEFAULT)

On peut aussi lui indiquer l'unité de temps du graphique de base.

L'éditeur de la plateforme colore le fond des blocs de code dans des **TimeFrame** supérieures afin de vous aider à visualiser les morceaux de codes calculés dans chaque unité de temps différente.



```
Variables 2
n[20], m[50]
TIMEFRAME(1 week)
PRTShortTerm = PRTBandsShortTerm
PRTMediumTerm = PRTBandsMediumTerm
TIMEFRAME(1 day)
Av200D = Average[200]
TIMEFRAME(DEFAULT)
ExtradayOK = PRTShortTerm > Av200D AND Av200D > PRTMediumTerm
IntradayOK = Average[n] > Average[m]
RETURN ExtradayOK AND IntradayOK
```

Il est aussi possible d'utiliser deux modes de calcul dans une unité de temps supérieure afin d'avoir plus de flexibilité dans vos calculs :

TIMEFRAME(X UnitéDeTemps , DEFAULT)

TIMEFRAME(X UnitéDeTemps , UPDATEONCLOSE)

DEFAULT: c'est le mode par défaut du **TimeFrame** (mode utilisé lorsque le second paramètre n'est pas indiqué), les calculs dans les unités de temps supérieures sont réalisés à chaque nouveau prix reçu dans l'unité de temps de base du graphique.

UPDATEONCLOSE: les calculs contenu dans une unité de temps dans ce mode sont réalisés lors de la clôture du chandelier de l'unité de temps supérieure.

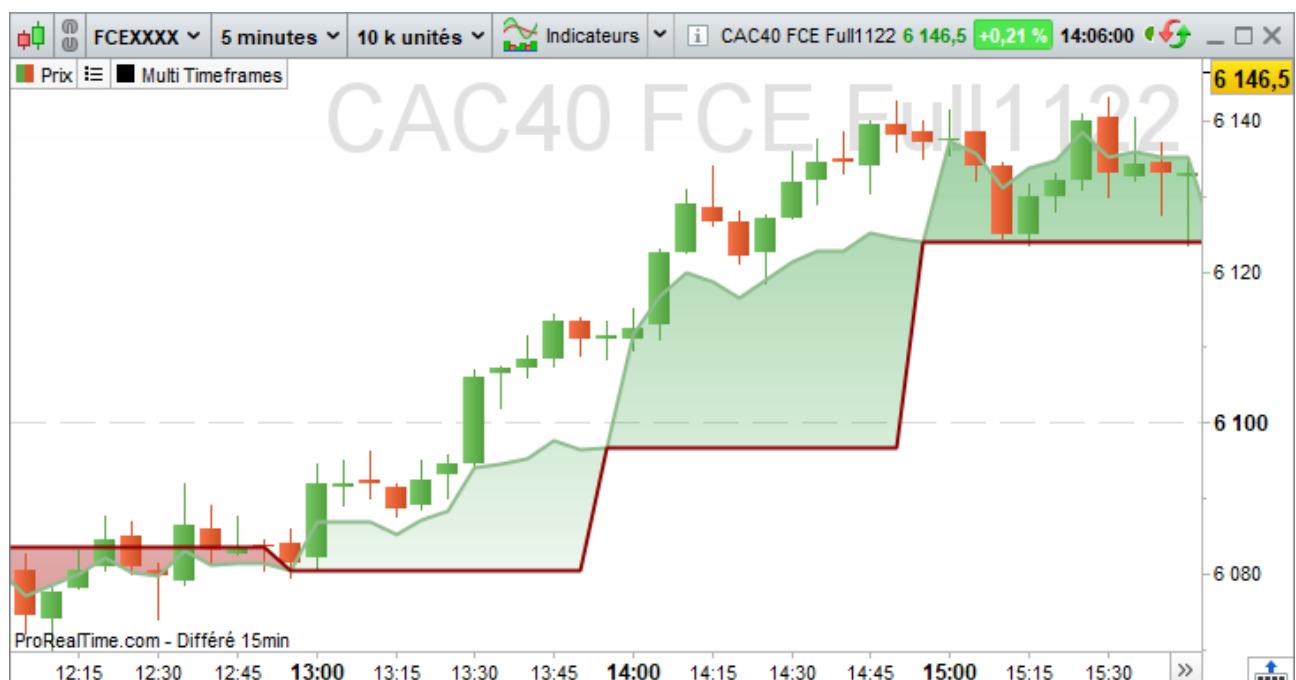
Voilà un exemple de code montrant la différence entre les deux modes de calcul:

```
// calcul d'un prix moyen entre l'ouverture et la clôture dans les deux modes disponibles.

TIMEFRAME(1 Hour)
MidPriceDefault=(Open+Close)/2

TIMEFRAME(1 Hour, UPDATEONCLOSE)
MidPriceUpdateOnClose=(Open+Close)/2

Return MidPriceDefault as "Prix Moyen mode Default" COLOURED ("DarkSeaGreen")
, MidPriceUpdateOnClose as "Prix Moyen mode UpdateOnClose" COLOURED ("DarkRed")
```



Ici on remarque que mon **MidPriceDefault** (en vert) vient se mettre à jour à chaque chandelier 5 minutes, alors que le **MidPriceUpdateOnClose** (en rouge) est mis à jour chaque chandelier 1 heure clôturé.

Remarque sur l'utilisation de l'instruction TIMEFRAME:

- Une variable calculée dans une unité de temps ne peut pas être écrasée par un calcul dans une autre unité de temps, par contre les variables peuvent être utilisées dans toutes les unités de temps contenues dans un même code.
- Il y a une limite de 10 instructions **TIMEFRAME** intraday (unité de temps inférieure au journalier) pour le trading automatique ainsi que le backtest.
- Pour le ProScreener, seul le mode **DEFAULT** est disponible, il n'est donc pas nécessaire de le préciser. De plus et afin de garantir la performance des calculs sur de nombreuses valeurs en temps réel, seule une liste prédéfinie d'unités de temps disponibles est autorisée pour ce module. Pour plus d'informations, nous vous invitons à lire le [Manuel de Programmation - Scan de Marché \(ProScreener\)](#).

Liste des périodes disponibles

| Périodes | Exemple |
|------------------------|-----------------------|
| Tick / Ticks | TIMEFRAME(1 Tick) |
| sec / Second / Seconds | TIMEFRAME(10 Seconds) |
| mn / Minute / Minutes | TIMEFRAME(5 Minutes) |
| Hour / Hours | TIMEFRAME(1 Hour) |
| Day / Days | TIMEFRAME(5 Days) |
| Week / Weeks | TIMEFRAME(1 Week) |
| Month / Months | TIMEFRAME(2 Months) |
| Year / Years | TIMEFRAME(1 Year) |

Les tableaux de données

Afin de pouvoir stocker plusieurs valeurs sur un même chandelier ou justement de ne stocker des valeurs que lorsque cela est nécessaire, nous vous proposons d'utiliser des tableaux de données plutôt que des variables.

Un code peut contenir autant de tableaux que nécessaires, ceux-ci pouvant contenir jusqu'à un million de valeurs maximum.

Un tableau est toujours préfixé du symbole \$.

Syntaxe d'une variable

A

Syntaxe d'un tableau

\$A

Un tableau commence de l'index 0 jusqu'à l'index 999 999

| | | | | | | | | | |
|--------|---|---|---|---|---|---|---|-----|---------|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... | 999 999 |
| Valeur | | | | | | | | | |

Pour insérer une valeur dans un tableau, il suffit d'utiliser

\$Tableau[Index] = valeur

Par exemple si on veut insérer la valeur du calcul de la moyenne mobile de période 20 à l'index 0 du tableau A, on écrira :

\$A[0] = Average[20](Close)

Pour lire la valeur d'un index du tableau, on utilisera, sur le même principe :

\$Tableau[Index]

Par exemple si on veut créer une condition qui vérifie que le close est supérieur à la valeur du premier index du tableau A:

Condition = Close > \$A[0]

A l'insertion d'une valeur à un index **n** d'un tableau, ProBuilder initialisera les valeurs à zéro pour tous les indices non-définis de 0 à **n-1** afin de faciliter l'utilisation des données contenues dans ce tableau.

Fonctions spécifiques

Plusieurs fonctions spécifiques aux tableaux sont disponibles afin de faciliter leur manipulation et utilisation:

- **ArrayMax**(\$Tableau) : retourne la valeur la plus haute du tableau qui a été définie. Les zéros remplis automatiquement par ProBuilder ne sont pas pris en compte.
- **ArrayMin**(\$Tableau) : retourne la valeur la plus petite du tableau qui a été définie. Les zéros remplis automatiquement par ProBuilder ne sont pas pris en compte.
- **ArraySort**(\$Tableau, **MODE**) : Trie le tableau de manière croissante (mode=**ASCEND**) ou de manière décroissante (mode=**DESCEND**). Les zéros remplis automatiquement par ProBuilder seront alors supprimés.
- **IsSet**(\$Tableau[index]) : retourne 1 si l'indice du tableau a été défini, 0 s'il n'a pas été défini. Les zéros remplis automatiquement par ProBuilder ne sont pas considérés comme ayant été définis donc la fonction retournera 0 sur ces indices.
- **LastSet**(\$Tableau) : retourne l'index défini le plus élevé du tableau, si aucun index n'a été défini dans le tableau, la fonction retournera -1.
- **UnSet**(\$Tableau) : Remet à 0 le tableau en supprimant complètement son contenu.

Remarque : contrairement aux variables et autres calculs effectués dans notre langage, les tableaux de données ne sont pas historisés. Il n'est donc pas possible de récupérer la valeur d'une cellule d'un tableau calculé sur un chandelier antérieur.

Si vous voulez en savoir plus, nous vous conseillons [ce lien](#) de notre partenaire ProRealCode qui détaille l'utilisation des tableaux à travers différents exemples (Page en anglais).

PRINT

PRINT est une fonctionnalité vous permettant d'afficher le contenu de vos variables en format tableaux avec le **BarIndex** et la **Date** correspondante

Elle est très utile pour créer et analyser des codes ProBuilder. La syntaxe est la suivante :

```
PRINT Var AS "Entête" FILLCOLOR (r,g,b,a) COLOURED (r,g,b,a)
```

Voici un code d'exemple et une sortie possible pour ce code :

```
Av20 = Average[20]
Av50 = Average[50]
CrossingOver = Av20 CROSSES OVER Av50
CrossingUnder = Av20 CROSSES UNDER Av50
IF CrossingOver OR CrossingUnder THEN
    IF CrossingOver THEN
        r = 35
        g = 200
    ELSIF CrossingUnder THEN
        r = 200
        g = 35
    ENDIF
    PRINT Av20 AS "Average 20"
    PRINT Av50 AS "Average 50"
    PRINT Av20[1] AS "Average 20[1]" COLOURED(0,0,250)
    PRINT Av50[1] AS "Average 50[1]" COLOURED(0,0,250)
    PRINT CrossingOver - CrossingUnder AS "CrossingType" FILLCOLOR(r,g,75,126)
    COLOURED(0,0,250)
ENDIF
RETURN CrossingOver - CrossingUnder
```

Print - 1 heure - APPLE

Mode dynamique

Temps réel

Modifier

| BarIndex | Date | Average 20 | Average 50 | CrossingType | Average 20[1] | Average 50[1] |
|----------|--------------------------|------------|------------|--------------|---------------|---------------|
| 681 | lun. 24 nov. 2025 11:00 | 270,757 | 270,627 | 1 | 270,549 | 270,5854 |
| 650 | lun. 17 nov. 2025 15:00 | 271,626 | 271,714 | -1 | 272,0105 | 271,7844 |
| 598 | jeu. 6 nov. 2025 12:00 | 269,688 | 269,6338 | 1 | 269,481 | 269,5666 |
| 590 | mer. 5 nov. 2025 11:00 | 268,9945 | 269,0964 | -1 | 269,076 | 269,0046 |
| 505 | lun. 20 oct. 2025 10:00 | 249,9825 | 249,5992 | 1 | 249,353 | 249,471 |
| 457 | jeu. 9 oct. 2025 11:00 | 256,2965 | 256,3726 | -1 | 256,458 | 256,4006 |
| 338 | mar. 16 sept. 2025 11:00 | 234,5965 | 234,1222 | 1 | 234,102 | 234,129 |
| 312 | mer. 10 sept. 2025 13:00 | 234,5165 | 234,862 | -1 | 235,18 | 234,9854 |
| 247 | mer. 27 août 2025 11:00 | 227,953 | 227,8376 | 1 | 227,8195 | 227,8496 |
| 211 | mer. 20 août 2025 10:00 | 230,3795 | 230,476 | -1 | 230,602 | 230,4654 |
| 148 | jeu. 7 août 2025 10:00 | 208,067 | 207,8536 | 1 | 207,3755 | 207,7216 |
| 103 | mar. 29 juil. 2025 14:00 | 213,0995 | 213,1106 | -1 | 213,1885 | 213,093 |
| 54 | ven. 18 juil. 2025 14:00 | 210,198 | 210,1434 | 1 | 210,1055 | 210,109 |
| 53 | ven. 18 juil. 2025 13:00 | 210,1055 | 210,109 | -1 | 210,0765 | 210,0514 |

Nb résultats :14

Variables :5 / 10

Option :

- **FILLCOLOR** : Permet de définir la couleur de fond de la cellule affichée par le **Print**.
- **COLOURED** : Permet de définir la couleur du texte de la cellule affichée par le **Print**.
- **AS**: Permet de choisir l'entête de la colonne dédiée à la variable (il est impossible d'avoir deux colonnes avec le même nom).

Les couleurs peuvent être définies avec des paramètres RGBA comme avec les noms de couleur de la convention W3C, nous vous renvoyons vers la section de coloration (**COLOURED**) des mots clés ProBuilder qui suit les mêmes règles.

Remarque : Comme vu dans l'exemple, une ligne n'est ajoutée au tableau que lorsque l'instruction est rencontrée, **PRINT** est donc très pratique à utiliser dans les boucles **IF**. Lorsque les codes ProBuilder se complexifient il arrive que certaines variables doivent être non nulles ou positives. Pour s'en assurer il est pratique d'utiliser l'instruction **PRINT**

```
// ...
IF MaVariablePositive < 0 THEN
    PRINT MaVariablePositive AS "Erreur : SQRT d'une variable négative"
ENDIF
AutreVariable = SQRT(MaVariablePositive)
// ...
```

Dans ce cas là si tout se passe bien aucun tableau ne s'affichera par contre vous serez assuré que **MaVariablePositive** est bien positive sur tout l'historique et qu'elle ne cause pas de soucis en aval de votre code.

Il est possible d'afficher jusqu'à 10 valeurs différentes calculées lors de l'exécution d'un même code (au-delà des dix premières valeurs, les suivantes seront ignorées, les **PRINT** qui ne sont jamais rencontrés dans le code au cours de l'exécution ne comptent pas)

La fenêtre des valeurs affichées peut contenir un historique de 200 calculs les plus récentes, qui sont mis à jour au gré des calculs effectués par votre code en temps réel.

Chapitre III : Applications pratiques

Créer un indicateur binaire ou ternaire : pourquoi et comment ?

Un indicateur binaire ou ternaire est par définition un indicateur ne pouvant retourner que deux ou trois résultats possibles (d'habitude 0, 1 ou -1). Son utilité principale dans un contexte boursier est de rendre immédiatement identifiable la vérification de la condition qui constitue l'indicateur.

Utilité d'un indicateur binaire ou ternaire :

- Permettre la détection des principales figures de chandeliers japonais
- Faciliter la lecture du graphique lorsqu'on cherche à repérer plusieurs conditions
- Pouvoir mettre des alertes classiques à une condition sur un indicateur qui en incorpore plusieurs
- Détecter des conditions complexes aussi sur l'historique
- Faciliter la réalisation d'un backtest

Les indicateurs binaires ou ternaires sont construits à l'aide de la fonction **IF**. Nous vous conseillons la relecture de la section relative avant de continuer la lecture.

Illustrons la création de ces indicateurs pour détecter des patterns de prix :

Indicateur binaire : détection du marteau

```
Hammer = Close > Open AND High = Close AND (Open-Low) >= 3*(Close-Open)
IF Hammer THEN
    Result = 1
ELSE
    Result = 0
ENDIF
RETURN Result AS "Hammer"
```

Ce code simplifié donnera aussi les mêmes résultats :

```
Hammer = Close > Open AND High = Close AND ( Open - Low ) >= 3 * ( Close - Open )
RETURN Hammer AS "Hammer"
```

Indicateur ternaire : détection de croix dorées et croix mortelles

```
EA10 = ExponentialAverage[10](Close)
EA20 = ExponentialAverage[20](Close)
```

```
GoldenCross = EA10 CROSSES OVER EA20
```

```
DeathCross = EA10 CROSSES UNDER EA20
```

// Les variables booléennes valent soit 0 soit 1 donc, lorsque mutuellement exclusive, soustraire 2 variables booléennes donne une variable à 3 états (0-1=-1, 0-0=0, ou 1-0=1)

```
Cross = GoldenCross - DeathCross
```

```
RETURN Cross STYLE (HISTOGRAM) COLOURED(DeathCross*210,GoldenCross*210,100)
```



Note : nous avons affiché les moyennes mobiles exponentielles de période 10 et 20 appliquées au prix de clôture pour bien mettre en évidence les correspondances des résultats de l'indicateur.

Vous pouvez retrouver d'autres indicateurs de détection des figures de prix dans la section « Applications Pratiques » plus loin dans ce manuel.

Chapitre IV : exercices

Figures de chandeliers

■ GAP UP ou DOWN



La couleur des chandeliers n'a pas d'importance

On définit en variable paramétrable l'amplitude égale à 0.001

Un gap se définit par deux conditions :

- (l'ouverture du jour est strictement supérieure à la clôture de la veille) ou (l'ouverture du jour est strictement inférieure à la clôture de la veille)
- la valeur absolue de ((l'ouverture du jour – la clôture de la veille) / la clôture de la veille) est strictement supérieure à l'amplitude

ONCE Amplitude = 0.001

```
// useDailyCandle permet de choisir d'utiliser DHigh et DLow au lieu de High et Low
// Les gaps sont calculés sur des chandeliers journalier de manière générale mais il peut
être intéressant de les calculer sur d'autres unités de temps.
```

ONCE useDailyCandle = 1

```
IF useDailyCandle THEN
    GapUpc1 = DLow(0) > DHigh(1)
    GapUpc2 = ABS((DLow(0) - DHigh(1)) / DHigh(1)) > Amplitude
    GapDownc1 = DHigh(0) < DLow(1)
    GapDownc2 = ABS((DHigh(0) - DLow(1)) / DLow(1)) > Amplitude
ELSE
    GapUpc1 = Low > High[1]
    GapUpc2 = ABS((Low - High[1]) / High[1]) > Amplitude
    GapDownc1 = High < Low[1]
    GapDownc2 = ABS((High - Low[1]) / Low[1]) > Amplitude
ENDIF
GapUp = GapUpc1 AND GapUpc2
GapDown = GapDownc1 AND GapDownc2
// Ajout d'une ligne horizontale pour mettre en évidence le niveau 0 (pas de gap)
DRAWHLINE(0) STYLE (DOTTEDLINE2) COLOURED("blue")
// Les variables booléennes valent soit 0 soit 1 donc, lorsque mutuellement exclusive,
soustraire 2 variables booléennes donne une variable à 3 états (0-1=-1, 0-0=0, ou 1-0=1)
RETURN GapUp - GapDown AS "Gap" STYLE (HISTOGRAM) COLOURED(GapDown*210, GapUp*210, 100)
```

 **Doji (version souple)**

On définit le doji par un range strictement supérieur à 5 fois la valeur absolue de (Open – Close).

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

```
RETURN Doji AS "Doji"
```

 **Doji (version stricte)**

On définit le doji par un close égal à l'Open.

```
Doji = (Open = Close)
```

```
RETURN Doji AS "Doji"
```

Indicateurs

• BODY MOMENTUM

Le Body Momentum est défini mathématiquement par :

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

BodyUp (resp. BodyDown) est un compteur du nombre de barres clôturent plus haut (resp. bas) que son ouverture, et ce sur une période définie (prenons période = 14)

ONCE Periods = 14

```
// Comme ( Close > Open ) et ( Close < Open ) sont des variables booléennes, elle valent soit 0 (faux) soit 1 (vrai).
```

```
// Donc utiliser l'indicateur summation sur ces variables booléennes nous permet de calculer le nombre de fois où la variable est vraie sur la plage voulue (ici 14)
```

```
BodyUp = summation[Periods]( Close > Open )
```

```
BodyDown = summation[Periods]( Close < Open )
```

```
BodyM = (BodyUp / (BodyUp + BodyDown)) * 100
```

```
RETURN BodyM AS "Body Momentum"
```

• ELLIOT WAVE OSCILLATOR

L'oscillateur Elliot représente la différence entre deux moyennes mobiles.

La moyenne mobile courte représente l'action des prix tandis que la moyenne longue représente la tendance de fond.

Lorsque les prix forment une vague 3, les prix grimpent fortement ce qui produit une valeur importante sur l'oscillateur.

Dans une vague 5, les prix grimpent plus lentement et l'oscillateur donne alors une valeur beaucoup plus faible.

```
RETURN Average[5](MedianPrice) - Average[35](MedianPrice) AS "Elliot Wave Oscillator"
```

💡 Williams %R

Voici un oscillateur dont le fonctionnement est similaire à l'oscillateur stochastique. Pour le tracer, on commence par définir dans un premier temps 2 courbes :

- 1) la courbe des plus hauts du plus haut sur 14 périodes
- 2) la courbe des plus bas du plus bas sur 14 périodes

Le %R se définit alors par $(\text{Close} - \text{LowestL}) / (\text{HighestH} - \text{LowestL}) * 100$

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

💡 Bandes de Bollinger

On définit ces bandes par un encadrement de la moyenne mobile arithmétique sur 20 barres appliquée au prix de clôture.

La moyenne mobile est bornée au-dessus (resp. en-dessous) par + (resp. -) 2 fois l'écart-type prise sur les 20 barres précédentes du prix de clôture.

```
ONCE Per = 20
ONCE nbSTD = 2

Bmid = Average[Per](Close)
StdDeviation = STD[Per](Close)

Bsup = Bmid + nbSTD * StdDeviation
Binf = Bmid - nbSTD * StdDeviation

RETURN Bmid AS "Average", Bsup AS "Bollinger Up", Binf AS "Bollinger Down"
```

Vous pouvez consulter notre communauté ProRealTime sur le [forum ProRealCode](#) afin d'y retrouver une [documentation en ligne](#) ainsi que de nombreux exemples.

Glossaire

A

| CODE | IMPLÉMENTATION | FONCTION |
|-------------------------|----------------------------------|---|
| ABS | ABS(a) | Fonction Mathématique « Valeur Absolue ». |
| AccumDistr | AccumDistr(price) | Désigne l'Accumulation Distribution classique. |
| ACOS | ACOS(a) | Fonction mathématique « Arc cosinus » (retourne un angle en degré). |
| AdaptiveAverage | AdaptiveAverage[x,y,z](price) | Indicateur de Moyenne Adaptative. |
| ADX | ADX[N] | Indicateur Average Directional Index. |
| ADXR | ADXR[N] | Indicateur Average Directional Index Rate. |
| AND | a AND b | Opérateur logique ET. |
| ArrayMax | ArrayMax(\$MyArray) | Retourne la valeur la plus grande. |
| ArrayMin | ArrayMin(\$MyArray) | Retourne la valeur la plus petite. |
| ArraySort | ArraySort(\$MyArray, ASCEND) | Tri le tableau par ordre croissant (ASCEND) ou décroissant (DESCEND). |
| AroonDown | AroonDown[P] | Désigne l'Aroon Down. |
| AroonUp | AroonUp[P] | Désigne l'Aroon Up. |
| ATAN | ATAN(a) | Fonction mathématique « Arc tangente » (retourne un angle en degré). |
| ANCHOR | ANCHOR(direction, index, yshift) | Fonction d'ancrage des dessins. |
| AS | RETURN x AS "ResultName" | Instruction servant à nommer une courbe. |
| ASIN | ASIN(a) | Fonction mathématique « Arc sinus » (retourne un angle en degré). |
| Average | Average[N](price) | Moyenne Mobile Arithmétique. |
| AverageTrueRange | AverageTrueRange[N](price) | Désigne la moyenne mobile par lissage de Wilder du True Range. |

B

| CODE | IMPLÉMENTATION | FONCTION |
|---------------------------|--|--|
| BACKGROUNDCOLOR | BACKGROUNDCOLOR(R,V,B,a) | Vous permet de colorer l'arrière-plan des graphiques ou des barres spécifiques (comme les jours pairs ou impairs). |
| BarIndex | BarIndex | Nombre de barres depuis la première barre de données chargée (dans un graphique dans le cas d'un indicateur ProBuilder ou pour un système de trading dans le cas d'un ProBacktest ou ProInvest). |
| BarsSince | BarsSince(condition,occurrence) | Renvoie le nombre de chandeliers depuis que la n ème occurrence de la condition a été remplie(n=0 par défaut dernière occurrence,n=1 avant-dernière occurrence). |
| Bold | DRAWTEXT(« text »,barindex,close,Serif,Bold, 10) | Style gras à appliquer au texte. |
| BoldItalic | DRAWTEXT(« text »,barindex,close,Serif,BoldItalic, 10) | Style gras italique à appliquer au texte. |
| BollingerBandWidth | BollingerBandWidth[N](price) | Bande passante de Bollinger. |
| BollingerDown | BollingerDown[N](price) | Support de la bande de Bollinger. |
| BollingerUp | BollingerUp[N](price) | Résistance de la bande de Bollinger. |
| BOTTOM | ANCHOR(BOTTOM,INDEX,YSHIFT) | Ancre en bas du graphique. |
| BOTTOMLEFT | ANCHOR(BOTTOMLEFT,INDEX,YSHIFT) | Ancre en bas à gauche du graphique. |
| BOTTOMRIGHT | ANCHOR(BOTTOMRIGHT,INDEX,YSHIFT) | Ancre en bas à droite du graphique. |
| BORDERCOLOR | BORDERCOLOR("red") | Ajoute une bordure colorée à l'objet associé. |
| BREAK | (FOR/DO/BREAK/NEXT) ou (WHILE/DO/BREAK/WEND) | Instruction de sortie forcée de boucle FOR ou WHILE. |

C

| CODE | IMPLÉMENTATION | FONCTION |
|----------------------------------|---|--|
| <code>CALCULATEONLASTBARS</code> | <code>DEFPARAM CalculateOnLastBars = 200</code> | Permet d'augmenter la vitesse à laquelle un indicateur sera calculé en définissant le nombre de barres présentant le résultat. |
| <code>CALL</code> | <code>myResult = CALL myFunction</code> | Appel de fonction utilisateur. |
| <code>CCI</code> | <code>CCI[N](price)</code> | Donne le Commodity Channel Index. |
| <code>CEIL</code> | <code>CEIL(N, M)</code> | Renvoie le plus petit nombre supérieur à N appliqué sur la décimale M. |
| <code>ChaikinOsc</code> | <code>ChaikinOsc[Ch1, Ch2](price)</code> | Désigne l'oscillateur de Chaikin. |
| <code>Chandle</code> | <code>Chandle[N](price)</code> | Désigne le Chande Momentum Oscillator. |
| <code>ChandeKrollStopUp</code> | <code>ChandeKrollStopUp[Pp, Qq, X]</code> | Stop de protection selon Chande et Kroll en position acheteuse. |
| <code>ChandeKrollStopDown</code> | <code>ChandeKrollStopDown[Pp, Qq, X]</code> | Stop de protection selon Chande et Kroll en position vendeuse. |
| <code>Close</code> | <code>Close[N]</code> | Désigne le prix de clôture de la barre courante ou celui du n-ième chandelier précédent. |
| <code>COLOURED</code> | <code>RETURN x COLOURED(R,G,B)</code> | Colorie une courbe d'une certaine couleur selon la convention RGB. |
| <code>COLORBETWEEN</code> | <code>COLORBETWEEN(a, b, color)</code> | Colorie l'espace entre deux valeurs. |
| <code>COS</code> | <code>COS(a)</code> | Fonction cosinus (argument a en degrés). |
| <code>CROSSES OVER</code> | <code>a CROSSES OVER b</code> | Opérateur booléen vérifiant qu'une courbe passe au-dessus d'une autre. |
| <code>CROSSES UNDER</code> | <code>a CROSSES UNDER b</code> | Opérateur booléen vérifiant qu'une courbe passe en-dessous d'une autre. |
| <code>Cumsum</code> | <code>Cumsum(price)</code> | Sommation d'un prix depuis le début de l'historique affiché. |
| <code>CurrentDayOfWeek</code> | <code>CurrentDayOfWeek</code> | Désigne le jour actuel. |
| <code>CurrentHour</code> | <code>CurrentHour</code> | Désigne l'heure actuelle. |
| <code>CurrentMinute</code> | <code>CurrentMinute</code> | Désigne la minute actuelle. |
| <code>CurrentMonth</code> | <code>CurrentMonth</code> | Désigne le mois actuel. |
| <code>CurrentSecond</code> | <code>CurrentSecond</code> | Désigne la seconde actuelle. |
| <code>CurrentTime</code> | <code>CurrentTime</code> | Désigne HeureMinute actuelle. |
| <code>CurrentYear</code> | <code>CurrentYear</code> | Désigne l'année actuelle. |
| <code>CustomClose</code> | <code>CustomClose[N]</code> | Constante paramétrable dans la fenêtre de propriétés. |
| <code>Cycle</code> | <code>Cycle(price)</code> | Indicateur Cycle. |

D

| CODE | IMPLÉMENTATION | FONCTION |
|-----------------------------|---|---|
| <code>Date</code> | <code>Date[N]</code> | Désigne la date de clôture de la barre courante. |
| <code>DATETOBARINDEX</code> | <code>DATETOBARINDEX(date)</code> | Permet d'utiliser une date pour les fonctions de dessin. |
| <code>Day</code> | <code>Day[N]</code> | Jour de clôture de la barre courante. |
| <code>Days</code> | <code>Days[N]</code> | Compteur de jours depuis 1900. |
| <code>Days</code> | <code>TIMEFRAME(X Days)</code> | Définit la période à « X Jours » pour la suite des calculs du code. |
| <code>DayOfWeek</code> | <code>DayOfWeek[N]</code> | Désigne le jour de la semaine durant lequel la barre courante a clos. |
| <code>DClose</code> | <code>DClose(N)</code> | Prix de clôture de la n-ième journée antérieure à celle de la barre courante. |
| <code>Decimals</code> | <code>Decimals</code> | Retourne le nombre de décimales du ticker. |
| <code>DEMA</code> | <code>DEMA[N](price)</code> | Double Moyenne Mobile Exponentielle. |
| <code>DHigh</code> | <code>DHigh(N)</code> | Prix le plus haut de la n-ième journée antérieure à celle de la barre courante. |
| <code>Dialog</code> | <code>DRAWTEXT(`text`,barindex,close,D ialog,Bold, 10)</code> | Police Dialog appliquée au texte. |
| <code>DI</code> | <code>DI[N](price)</code> | Désigne le Demand Index. |
| <code>DIminus</code> | <code>DIminus[N](price)</code> | Désigne le DI- |
| <code>DIplus</code> | <code>DIplus[N](price)</code> | Désigne le DI+ |

| CODE | IMPLÉMENTATION | FONCTION |
|---------------------------|--|---|
| DivergenceCCI | DivergenceCCI[Div1,Div2,Div3,Div4] | Indicateur de détection de divergences entre le prix et le CCI. |
| DivergenceMACD | DivergenceMACD[Div1,Div2,Div3,Div4](close) | Indicateur de détection de divergences entre le prix et le MACD. |
| DivergenceRSI | DivergenceRSI[Div1,Div2,Div3,Div4](close) | Indicateur de détection de divergences entre le prix et le RSI. |
| DLow | DLow(N) | Prix le plus bas de la n-ième journée antérieure à celle de la barre courante. |
| DO | Voir FOR et WHILE | Instruction facultative des FOR et WHILE pour l'action de bouclage. |
| DonchianChannelCenter | DonchianChannelCenter[N] | Canal du milieu de l'indicateur Donchian pour N périodes. |
| DonchianChannelDown | DonchianChannelDown[N] | Canal inférieur de l'indicateur Donchian pour N périodes. |
| DonchianChannelUp | DonchianChannelUp[N] | Canal supérieur de l'indicateur Donchian pour N périodes. |
| DOpen | DOpen(N) | Prix d'ouverture de la n-ième journée antérieure à celle de la barre courante. |
| DOTTEDLINE | STYLE(DOTTEDLINE1/2/3/4, width) | Style applicable aux traits d'un objet. |
| DOWNT0 | Voir FOR | Instruction sur boucle FOR pour une lecture décroissante. |
| DPO | DPO[N](price) | Désigne le Detrended Price Oscillator. |
| DRAWARROW | DRAWARROW(x1,y1) | Dessine une flèche pointant à la droite du point de référence. |
| DRAWARROWDOWN | DRAWARROWDOWN(x1,y1) | Dessine une flèche vers le bas pointant sur le point de référence. |
| DRAWARROWUP | DRAWARROWUP(x1,y1) | Dessine une flèche vers le haut pointant sur le point de référence. |
| DRAWBARCHART | DRAWBARCHART(open,high,low,close) | Dessine une barre personnalisée sur le graphique. Open, high, low et close peuvent être des constantes ou des variables. |
| DRAWCANDLE | DRAWCANDLE(open,high,low,close) | Dessine un chandelier personnalisé sur le graphique. Open, high, low et close peuvent être des constantes ou des variables. |
| DRAWEELLIPSE | DRAWEELLIPSE(x1,y1,x2,y2) | Dessine une ellipse sur le graphique. |
| DRAWHLINE | DRAWHLINE(y1) | Dessine une ligne horizontale sur le graphique. |
| DRAWLINE | DRAWLINE(x1,y1,x2,y2) | Dessine une ligne sur le graphique. |
| DRAWONLASTBARONLY | DEFPARAM DrawOnLastBarOnly = true | Paramètre qui vous permet de dessiner des objets sur la dernière barre seulement. |
| DRAWPOINT | DRAWPOINT(x1,y1, optional size) | Dessine un point sur le graphique. |
| DRAWRAY | DRAWRAY(x1,y1,x2,y2) | Dessine une demi-droite sur le graphique. |
| DRAWRECTANGLE | DRAWRECTANGLE(x1,y1,x2,y2) | Dessine un rectangle sur le graphique. |
| DRAWSEGMENT | DRAWSEGMENT(x1,y1,x2,y2) | Dessine un segment sur le graphique. |
| DRAWTXT | DRAWTXT("your text", x1, y1) | Ajoute un champ de texte au graphique avec le texte de votre choix à un emplacement spécifié. |
| DRAWTRIANGLE | DRAWTRIANGLE(x1, y1, x2, y2, x3, y3) | Dessine un triangle sur le graphique. |
| DRAWVLINE | DRAWVLINE(x1) | Dessine une ligne verticale sur le graphique. |
| DynamicZoneRSIDown | DynamicZoneRSIDown[rsiN, N] | Bandes inférieures de l'indicateur Dynamic Zone RSI. |
| DynamicZoneRSIUp | DynamicZoneRSIUp[rsiN, N] | Bandes supérieures de l'indicateur Dynamic Zone RSI. |
| DynamicZoneStochasticDown | DynamicZoneStochasticDown[N] | Bandes inférieures de l'indicateur Dynamic Zone Stochastic. |
| DynamicZoneStochasticUp | DynamicZoneStochasticUp[N] | Bandes supérieures de l'indicateur Dynamic Zone Stochastic. |

E

| CODE | IMPLÉMENTATION | FONCTION |
|-------------------|-----------------------------|--|
| EaseOfMovement | EaseOfMovement[] | Désigne l'indicateur Ease of Movement. |
| ElderrayBearPower | ElderrayBearPower[N](close) | Désigne l'indicateur Elderray Bear Power. |
| ElderrayBullPower | ElderrayBullPower[N](close) | Désigne l'indicateur Elderray Bull Power. |
| ELSE | Voir IF/THEN/ELSE/ENDIF | Instruction d'appel de la seconde condition à défaut de la première issue du IF. |
| ELSIF | Voir IF/THEN/ELSE/ENDIF | Contraction de ELSE IF. |

| CODE | IMPLÉMENTATION | FONCTION |
|--------------------|------------------------------|--|
| EMV | EMV[N] | Désigne l'indicateur Ease of Movement Value. |
| ENDIF | Voir IF/THEN/ELSE/ENDIF | Instruction de clôture des instructions conditionnelles. |
| EndPointAverage | EndPointAverage[N](price) | Moyenne Mobile au dernier point. |
| EXP | EXP(a) | Fonction Mathématique « Exponentielle ». |
| ExponentialAverage | ExponentialAverage[N](price) | Moyenne Mobile Exponentielle. |

F – G – H

| CODE | IMPLÉMENTATION | FONCTION |
|-----------------------|--|---|
| FILLCOLOR | PRINT x FILLCOLOR (r,g,b) | Permet de contrôler la couleur de fond du tableau PRINT cellule par cellule. |
| FractalDimensionIndex | FractalDimensionIndex[N](close) | Désigne l'indicateur Fractal Dimension Index. |
| FOR/TO/NEXT | FOR i =a TO b DO a NEXT | Prend les valeurs désignées du début à la fin ou vice versa. |
| ForceIndex | ForceIndex(price) | Indicateur Force Index déterminant qui contrôle le marché. |
| FLOOR | FLOOR(N, m) | Renvoie le plus grand nombre inférieur à N avec une précision de m chiffres après la virgule. |
| GetTimeFrame | GetTimeFrame | Renvoie la période actuelle du code (en secondes). |
| High | High[N] | Désigne le plus haut cours atteint durant la période N. |
| Highest | Highest[N](price) | Désigne le plus haut cours sur un horizon donné. |
| HighestBars | HighestBars[N] | Renvoie le décalage de chandelier de la dernière valeur la plus élevée. |
| HISTOGRAM | RETURN close STYLE(HISTOGRAM, lineWidth) | Applique le style histogramme sur la valeur retournée. |
| HistoricVolatility | HistoricVolatility[N](price) | Désigne la volatilité historique ou statistique. |
| Hour | Hour[N] | Désigne l'heure de clôture de chaque barre. |
| Hours | TIMEFRAME(X Hours) | Définit la période à « X Heures » pour la suite des calculs du code. |
| HullAverage | HullAverage[N](close) | Désigne l'indicateur Hull Average. |

I - J - K

| CODE | IMPLÉMENTATION | FONCTION |
|--------------------|---|---|
| IF/THEN/ENDIF | IF a THEN b ENDIF | Ensemble d'instructions conditionnelles sans deuxième condition. |
| IF/THEN/ELSE/ENDIF | IF a THEN b ELSE c ENDIF | Ensemble d'instructions conditionnelles. |
| IntradayBarIndex | IntradayBarIndex[N] | Compte le nombre de chandeliers sur le graphique intraday. |
| IsSet | IsSet(\$MyArray[index]) | Retourne 1 si l'indice (index) du tableau a été défini, 0 si il n'a pas été défini. |
| INDEX | ANCHOR(TOPLEFT,INDEX,YSHIFT) | Définit la valeur du point de l'objet pour l'axe horizontale comme étant un barindex. |
| Italic | DRAWTEXT(« text »,barindex,close,S erif,Italic, 10) | Style italique à appliquer au texte. |
| KeltnerBandCenter | KeltnerBandCenter[N] | Bande centrale de l'indicateur Keltner pour N période. |
| KeltnerBandDown | KeltnerBandDown[N] | Bande inférieure de l'indicateur Keltner pour N période. |
| KeltnerBandUp | KeltnerBandUp[N] | Bande supérieure de l'indicateur Keltner pour N période. |
| KijunSen | KijunSen[T,K,S] | Retourne la valeur KijunSen de l'indicateur Ichimoku. |

L

| CODE | IMPLÉMENTATION | FONCTION |
|---------|---------------------------|---|
| LastSet | LastSet(\$MyArray) | Retourne l'index défini le plus élevé du tableau. |
| LEFT | ANCHOR(LEFT,INDEX,YSHIFT) | Ancrage à gauche du graphique. |
| LINE | STYLE(LINE, lineWidth) | Style de ligne standard. |

| CODE | IMPLÉMENTATION | FONCTION |
|-----------------------|---------------------------------|--|
| LinearRegression | LinearRegression[N](price) | Droite de régression linéaire. |
| LinearRegressionSlope | LinearRegressionSlope[N](price) | Pente de la droite de régression linéaire. |
| LOG | LOG(a) | Fonction mathématique « logarithme népérien ». |
| Low | Low[N] | Désigne le plus bas atteint durant la période. |
| Lowest | Lowest[N](price) | Désigne le plus bas d'une période sur un horizon donné. |
| LowestBars | LowestBars[N] | Renvoie le décalage de chandelier de la dernière valeur la plus basse. |

M - N

| CODE | IMPLÉMENTATION | FONCTION |
|---------------------|--|---|
| MACD | MACD[S,L,Si](price) | Moving Average Convergence Divergence (MACD). |
| MACDline | MACDLine[S,L,Si](price) | Désigne la ligne du MACD. |
| MACDSignal | MACDSignal[S,L,Si](price) | Désigne le Signal du MACD. |
| MassIndex | MassIndex[N] | Indicateur Mass Index appliqu  sur N barres. |
| MAX | MAX(a,b) | Fonction math matique « Maximum ». |
| MedianPrice | MedianPrice | Moyenne du prix le plus haut et du plus bas. |
| MIDDLE | ANCHOR(MIDDLE,INDEX,YSHIFT) | Ancrage au milieu du graphique. |
| MIN | MIN(a,b) | Fonction Math matique « Minimum ». |
| Minute | Minute | D signe la minute du moment de la cl ture de chaque barre de l'historique. |
| Minutes | TIMEFRAME(X Minutes) | D finit la p riode   X Minutes pour la suite des calculs du code. |
| MOD | a MOD b | Fonction Math matique « Reste de la division euclidienne ». |
| Momentum | Momentum[N] | Momentum (prix de cl ture – prix de cl ture de la n-i me barre pr c dente). |
| MoneyFlow | MoneyFlow[N](price) | Donne le MoneyFlow entre -1 et 1. |
| MoneyFlowIndex | MoneyFlowIndex[N] | D signe le MoneyFlowIndex. |
| Monospaced | DRAWTEXT(« text »,barindex,close,M onospaced,italic, 10) | Police Monospaced appliqu e au texte. |
| Month | Month[N] | D signe le mois de la cl ture de chaque barre de l'historique. |
| Months | TIMEFRAME(X Months) | D finit la p riode   X Mois pour la suite des calculs du code. |
| NegativeVolumeIndex | NegativeVolumeIndex[N] | D signe l'indice de volume n gatif. |
| NEXT | Voir FOR/TO/NEXT | Instruction   placer   la fin de la boucle « FOR ». |
| NOT | NOT a | Op rateur logique NON. |

O

| CODE | IMPL MENTATION | FONCTION |
|---------------|-----------------------------------|--|
| OBV | OBV(price) | D signe l' « On-Balance-Volume ». |
| ONCE | ONCE VariableName = VariableValue | Instruction qui en pr c de une autre qu'on ne veut r aliser qu'une seule fois. |
| Open | Open[N] | Prix d'ouverture du n-i me chandelier pr c dent. |
| OpenDay | OpenDay[N] | Jour d'ouverture du n-i me chandelier pr c dent. |
| OpenDayOfWeek | OpenDay[N] | Jour de la semaine de l'ouverture du n-i me chandelier pr c dent. |
| OpenHour | OpenHour[N] | Heure d'ouverture du n-i me chandelier pr c dent. |
| OpenMinute | OpenMinute[N] | Minute d'ouverture du n-i me chandelier pr c dent. |
| OpenMonth | OpenMonth[N] | Mois d'ouverture du n-i me chandelier pr c dent. |
| OpenSecond | OpenSecond[N] | Seconde d'ouverture du n-i me chandelier pr c dent. |
| OpenTime | OpenTime[N] | Temps (HHMMSS) d'ouverture du n-i me chandelier pr c dent. |
| OpenTimestamp | OpenTimestamp[N] | Timestamp UNIX d'ouverture du n-i me chandelier pr c dent. |
| OpenWeek | OpenWeek[N] | Semaine d'ouverture du n-i me chandelier pr c dent. |

| CODE | IMPLÉMENTATION | FONCTION |
|----------|----------------|---|
| OpenYear | OpenYear[N] | Année d'ouverture du n-ième chandelier précédent. |
| OR | a OR b | Opérateur logique OU. |

P – Q

| CODE | IMPLÉMENTATION | FONCTION |
|---------------------|---------------------------------------|---|
| PIPSIZE | PIPSIZE | Taille d'un pip (forex). |
| PIPVALUE | PIPVALUE | Valeur en €/\$ d'un pip (ou point) : PIPVALUE = POINTVALUE. |
| POINT | RETURN close STYLE(POINT, pointWidth) | Applique le style point sur la valeur rentrée. |
| POINTSIZE | POINTSIZE | Taille d'un pip (ou point) : PIPSIZE=POINTSIZE. |
| POINTVALUE | POINTVALUE | Valeur en €/\$ d'un pip (ou point) : PIPVALUE = POINTVALUE. |
| PositiveVolumeIndex | PositiveVolumeIndex(price) | Désigne l'indicateur Positive Volume Index. |
| POW | POW(N,P) | Retourne la valeur de N à la puissance P. |
| PriceOscillator | PriceOscillator[S,L](price) | Indicateur Percentage Price oscillator. |
| PRINT | PRINT x | Affiche la variable dans une fenêtre séparée, utile pour le débogage. |
| PRTBANDSUP | PRTBANDSUP | Donne la valeur de la bande supérieure de PRTBands. |
| PRTBANDSDOWN | PRTBANDSDOWN | Donne la valeur de la bande inférieure de PRTBands. |
| PRTBANDSHORTTERM | PRTBANDSSHORTTERM | Donne la valeur de la bande court terme de PRTBands. |
| PRTBANDMEDIUMTERM | PRTBANDSMEDIUMTERM | Donne la valeur de la bande long terme de PRTBands. |
| PVT | PVT(price) | Désigne l'indicateur « Price Volume Trend ». |

R

| CODE | IMPLÉMENTATION | FONCTION |
|-----------|---------------------------------------|--|
| R2 | R2[N](price) | Coefficient R Carré (taux d'erreur des prix à la régression linéaire). |
| RANDOM | RANDOM(Min, Max) | Génère un nombre entier aléatoire entre les bornes Min et Max incluses. |
| Range | Range[N] | Définition entre le prix le plus haut et le plus bas de la barre courante. |
| Repulse | Repulse[N](price) | Meilleure la poussée haussière et baissière de chaque bougie. |
| RepulseMM | RepulseMM[N,PeriodMM,factorMM](price) | Ligne MM de l'indicateur Repulse. |
| RETURN | RETURN Result | Instruction qui renvoie le résultat. |
| RIGHT | ANCHOR(RIGHT,INDEX,YSHIFT) | Ancre à droite du graphique. |
| ROC | ROC[N](price) | Désigne le « Price Rate of Change ». |
| RocnRoll | RocnRoll(price) | Désigne l'indicateur RocnRoll basé sur l'indicateur ROC. |
| ROUND | ROUND(a) | Fonction mathématique « Arrondi à l'unité ». |
| RSI | RSI[N](price) | Désigne l'oscillateur « Relative Strength Index ». |

S

| CODE | IMPLÉMENTATION | FONCTION |
|-------------|--|---|
| SansSerif | DRAWTEXT(« text »,barindex,close,SansSerif,Italic, 10) | Police SansSerif appliquée au texte. |
| SAR | SAR[At,St,Lim] | Désigne le Parabolique SAR. |
| SARatdmf | SARatdmf[At,St,Lim](price) | Désigne le Parabolique SAR ATDMF. |
| Second | Second[n] | Désigne la seconde du moment de la clôture de chaque barre de l'historique. |
| Seconds | TIMEFRAME(X Seconds) | Définit la période à « X Secondes » pour la suite des calculs du code. |
| SenkouSpanA | SenkouSpanA[T,K,S] | Retourne la valeur SenkouSpanA de l'indicateur Ichimoku. |
| SenkouSpanB | SenkouSpanB[T,K,S] | Retourne la valeur SenkouSpanB de l'indicateur Ichimoku. |

| CODE | IMPLÉMENTATION | FONCTION |
|--------------------|--|---|
| Serif | DRAWTEXT(« text »,barindex,close,Serif,Italic, 10) | Police Serif appliquée au texte. |
| SIN | SIN(a) | Fonction Mathématique « Sinus » (argument a en degrés). |
| SGN | SGN(a) | Fonction Mathématique « Signe de ». |
| SMI | SMI[N,SS,DS](price) | Désigne le Stochastic Momentum Index. |
| SmoothedRepulse | SmoothedRepulse[N](price) | Désigne un Repulse lissée. |
| SmoothedStochastic | SmoothedStochastic[N,K](price) | Désigne une Stochastique lissée. |
| SQUARE | SQUARE(a) | Fonction mathématique « Mise au carré ». |
| SQRT | SQRT(a) | Fonction Mathématique « Mise à la racine carrée ». |
| Standard | DRAWTEXT(« text »,barindex,close,Standard, 10) | Style standard appliquée au texte. |
| STD | STD[N](price) | Fonction Statistique « écart-type ». |
| STE | STE[N](price) | Fonction Statistique « écart-erreur ». |
| STYLE | STYLE(dottedline, width) | Applique le type de style dottedline avec une épaisseur width sur un objet. |
| Stochastic | Stochastic[N,K](price) | Ligne %K de la Stochastique. |
| StochasticD | StochasticD[N,K,D](price) | Ligne %D de la Stochastique. |
| Summation | Summation[N](price) | Somme d'un certain prix des N derniers chandeliers. |
| Supertrend | Supertrend[STF,N] | Désigne le Super Trend. |

T

| CODE | IMPLÉMENTATION | FONCTION |
|-------------------|-------------------------------|---|
| TAN | TAN(a) | Fonction mathématique « Tangente » (argument a en degrés). |
| TEMA | TEMA[N](price) | Moyenne Mobile Exponentielle Triple. |
| TenkanSen | TenkanSen[T,K,S] | Retourne la valeur TenkanSen de l'indicateur Ichimoku. |
| THEN | Voir IF/THEN/ELSE/ENDIF | Instruction suivant la première condition de l'instruction « IF ». |
| Ticks | TIMEFRAME(X Ticks) | Définit la période à « X Ticks » pour la suite des calculs du code. |
| Ticksize | Ticksize | Variation minimale du cours de l'instrument du graphique. |
| Time | Time[N] | Donne l'HeureMinuteSeconde (HHMMSS) et permet de faire appel à l'heure. |
| TimeSeriesAverage | TimeSeriesAverage[N](price) | Moyenne mobile des séries temporelles. |
| Timestamp | Timestamp[N] | Date UNIX de la clôture du chandelier N. |
| TO | Voir FOR/TO/NEXT | Instruction « jusqu'à » dans la boucle « Pour ». |
| Today | Today | Date de la journée actuelle (AAAAMMDD). |
| TOP | ANCHOR(TOP,INDEX,YSHIFT) | Ancre en haut du graphique. |
| TOPLEFT | ANCHOR(TOPLEFT,INDEX,YSHIFT) | Ancre en haut à gauche du graphique. |
| TOPRIGHT | ANCHOR(TOPRIGHT,INDEX,YSHIFT) | Ancre en haut à droite du graphique. |
| TotalPrice | TotalPrice[N] | (Clôture + Ouverture + Plus Haut + Plus Bas) / 4. |
| TR | TR(price) | Désigne le True Range. |
| TriangularAverage | TriangularAverage[N](price) | Moyenne Mobile Triangulaire. |
| TRIX | TRIX[N](price) | Triple Moyenne Mobile Exponentielle. |
| TypicalPrice | TypicalPrice[N] | Prix Typique (moyenne de plus haut, plus bas et clôture). |

U – V – W

| CODE | IMPLÉMENTATION | FONCTION |
|-----------|------------------|---|
| Undefined | a = Undefined | Pour laisser une variable indéfinie (Null). |
| Unset | unset(\$MyArray) | Réinitialise les données du tableau. |

| CODE | IMPLÉMENTATION | FONCTION |
|-----------------------|------------------------------------|---|
| VALUE | ANCHOR(TOP, INDEX, VALUE) | Définit la valeur du point de l'objet pour l'axe vertical comme étant un prix. |
| Variation | Variation(price) | Définit la variation entre la clôture de la veille et la clôture courante en %. |
| ViMinus | ViMinus[N] | Bandes inférieures de l'indicateur Vortex. |
| ViPlus | ViPlus[N] | Bandes supérieures de l'indicateur Vortex. |
| Volatility | Volatility[S, L] | Désigne la volatilité de Chaikin. |
| Volume | Volume[N] | Désigne le volume. |
| VolumeAdjustedAverage | VolumeAdjustedAverage[N](Price) | Désigne la moyenne mobile ajustée au volume de l'instrument. |
| VolumeOscillator | VolumeOscillator[S, L] | Désigne l'oscillateur de volume. |
| VolumeROC | VolumeROC[N] | Désigne le volume du Rate Of Change. |
| Weeks | TIMEFRAME(X Weeks) | Définit la période à « X Semaines » pour la suite des calculs du code. |
| WeightedAverage | WeightedAverage[N](price) | Désigne la Moyenne Mobile Pondérée. |
| WeightedClose | WeightedClose[N] | Moyenne pondérée entre le prix de clôture, le plus haut et le plus bas. |
| WEND | Voir WHILE/DO/WEND | Instruction à placer à la fin de la boucle Tant Que. |
| WHILE/DO/WEND | WHILE (condition) DO (action) WEND | Boucle "Tant Que". |
| WilderAverage | WilderAverage[N](price) | Donne la moyenne mobile de Wilder. |
| Williams | Williams[N](close) | Calcule le %R de Williams. |
| WilliamsAccumDistr | WilliamsAccumDistr(price) | Indicateur Accumulation/Distribution de Williams. |

X – Y – Z

| CODE | IMPLÉMENTATION | FONCTION |
|-------------|----------------------------|--|
| XOR | a XOR b | Opérateur logique OU exclusif. |
| XSHIFT | ANCHOR(TOP, XSHIFT, VALUE) | Définit la valeur du point de l'objet pour l'axe horizontal comme étant un décalage. |
| Year | Year[N] | Donne l'évolution des années. |
| Years | TIMEFRAME(X Years) | Définit la période à « X Année(s) » pour la suite des calculs du code. |
| Yesterday | Yesterday[N] | Donne l'évolution du jour d'avant. |
| YSHIFT | ANCHOR(TOP, INDEX, YSHIFT) | Définit la valeur du point de l'objet pour l'axe vertical comme étant un décalage. |
| ZigZag | ZigZag[Zr](price) | Zig-Zag de la théorie des vagues d'Elliott. |
| ZigZagPoint | ZigZagPoint[Zp](price) | Zig-Zag de la théorie des vagues d'Elliott calculé à Zp points. |

Autres

| CODE | FONCTION | CODE | FONCTION |
|------|---------------------------------|------|-----------------------------------|
| + | Opérateur d'addition. | <> | Opérateur de différence. |
| - | Opérateur de soustraction. | < | Opérateur d'infériorité stricte. |
| * | Opérateur de multiplication. | > | Opérateur de supériorité stricte. |
| / | Opérateur de division décimale. | <= | Opérateur d'infériorité. |
| = | Opérateur d'égalité. | >= | Opérateur de supériorité. |



ProRealTime