Coming soon

A documentação para esta página ainda está em desenvolvimento. Em breve, adicionaremos as informações completas. Agradecemos a sua compreensão.

Eventos

Nesta página, você encontrará informações sobre os eventos, para que servem e como eles podem ser utilizados em estratégias no ScriptBot.

> **Nota**
>
> Os eventos ainda não estão em sua versão final. Em futuras atualizações, será incluída uma manipulação mais avançada e flexível dos eventos.
>
> Algumas funcionalidades previstas incluem:
>
> - **Manipulação de fluxo:** Permitir cancelar a execução e continuação de uma ação específica;
> - **Manipulação de dados:** Permitir alteração dos parâmetros de um evento;
>
> Atualmente, os eventos são utilizados de forma simples para monitorar e reagir a acontecimentos do mercado, sem interferir no fluxo principal de execução do robô.

O que é um evento?

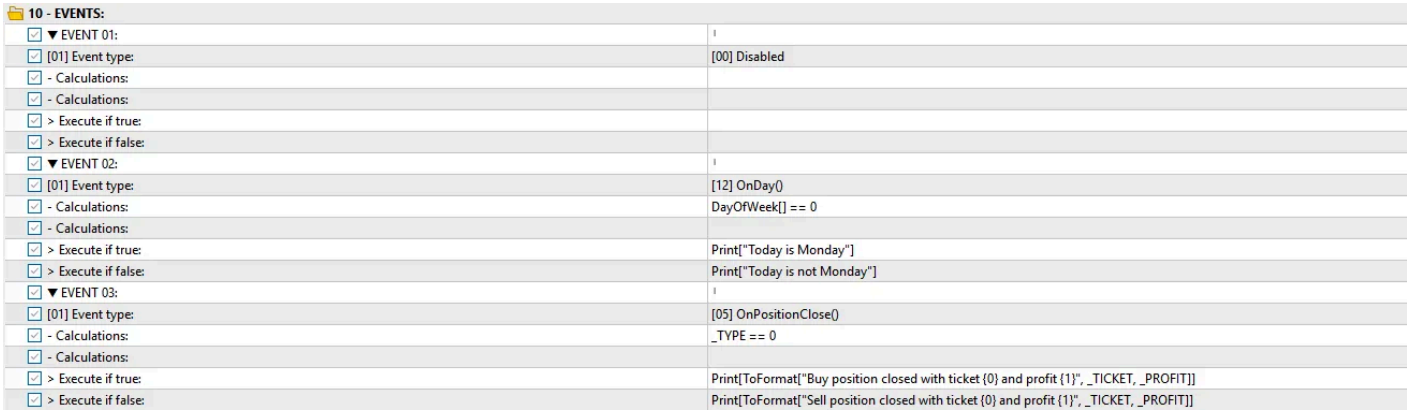
Eventos são ações ou ocorrências que acontecem durante a execução do robô, como a abertura de uma ordem, o encerramento de uma posição ou a validação de um gatilho. Sempre que uma tarefa relevante é realizada, o robô dispara um evento correspondente.

Esses eventos podem ser monitorados e utilizados para automatizar as ações do robô em diferentes situações de mercado. Dessa forma, o usuário pode configurar regras e estratégias personalizadas que respondem imediatamente após uma ação ocorrer, sem depender de temporizadores ou atrasos adicionais.

Na prática, os eventos tornam o ScriptBot mais inteligente, dinâmico e flexível, permitindo a execução de ações em tempo real a partir de cada ocorrência. Isso proporciona maior controle e aumenta a eficiência das estratégias.

Configurando os eventos

A configuração dos eventos é simples e intuitiva. Abaixo, você pode visualizar uma imagem que ilustra a estrutura dos eventos:



Cada evento pode ser configurado de forma personalizada, de acordo com as necessidades do usuário.

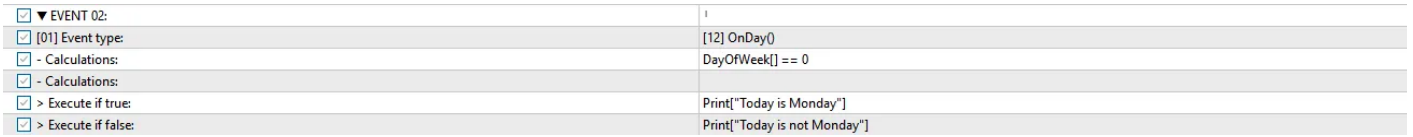
Evento

- **[01] - Tipo**: Define o tipo do evento que será ouvido. Acesse a Lista de Eventos
- **[02 & 03] - Cálculos de Validação**: São expressões condicionais que o sistema executa quando o evento for chamado.
- **[04] - Executar em caso de sucesso**: Ações que serão executadas quando o evento for validado.
- **[05] - Executar em caso de falha**: Ações que serão executadas quando o evento falhar.

Exemplo

Neste exemplo, vamos monitorar o evento de início de um novo dia e registrar uma mensagem nas logs indicando se é segunda-feira ou não.

Exemplo da configuração:



Sempre que um novo dia começar, o evento `OnDay()` será acionado. O sistema então realizará os cálculos necessários para determinar se o evento foi validado ou não.

O fluxo da execução segue a seguinte ordem:

```
flowchart TD
    A["OnDay()"] --> n1
    n1["DayOfWeek[] == 0"] -- TRUE --- n3["Print[Today is Monday]"]
    n1 -- FALSE --- n4["Print[Today is not Monday]"]
    n1@{ shape: braces}
    n3@{ shape: braces}
    n4@{ shape: braces}
    A@{ shape: event}
```

O fluxo é resumido nas seguintes etapas:

1. **EVENTO**
   - Evento: `OnDay()`.
   - Evento que será monitorado.

2. **CÁLCULOS**
   - Condição: `DayOfWeek[] == 0`.
   - Responsável por executar a tentativa da validação.
   - Se o resultado for **verdadeiro**, o fluxo segue para a etapa **(SUCESSO)**.
   - Se o resultado for **falso**, o fluxo segue para a etapa **(FALHA)**.

3. **SUCESSO**
   - Resposta: `Print["Today is Monday"]`.
   - Executa os cálculos definidos para o cenário de validação bem-sucedida.

4. **FALHA**
   - Resposta: `Print["Today is not Monday"]`.
   - Executa os cálculos definidos para o cenário de validação mal-sucedida.

Filtros

Nesta página, você encontrará informações sobre os filtros, para que servem e como eles podem ajudar a melhorar a confiabilidade das estratégias no ScriptBot.

O que é um filtro?

Os filtros são um sistema avançado de validação em camadas, baseado em regras lógicas personalizadas e atribuídas com pesos específicos. Cada filtro funciona de forma independente, mas contribui para a decisão final com seu valor de importância. A soma dos pesos define se a operação será confirmada ou rejeitada.

Diversos filtros podem ser criados para compras e vendas, cada um com configurações personalizáveis:

- Definir como obrigatório ou opcional;
- Atribuir um peso personalizado;
- Estabelecer condições lógicas baseadas em indicadores ou sinais de mercado;
- Escolher diferentes métodos de validação.

Os filtros são acionados automaticamente após o **Gatilho de Confirmação**, iniciando a análise para autorizar ou não a entrada no mercado.

Para que servem?

Os filtros garantem que entradas de compra ou venda só ocorram quando condições específicas forem atendidas, oferecendo ao usuário um controle mais rigoroso e personalizado sobre suas operações.

O sistema funciona com base na soma dos pesos dos filtros validados. Quando esse total atinge (ou supera) o peso mínimo definido, a ordem é autorizada. Caso contrário, a entrada é bloqueada, e a ordem não é enviada ao mercado.

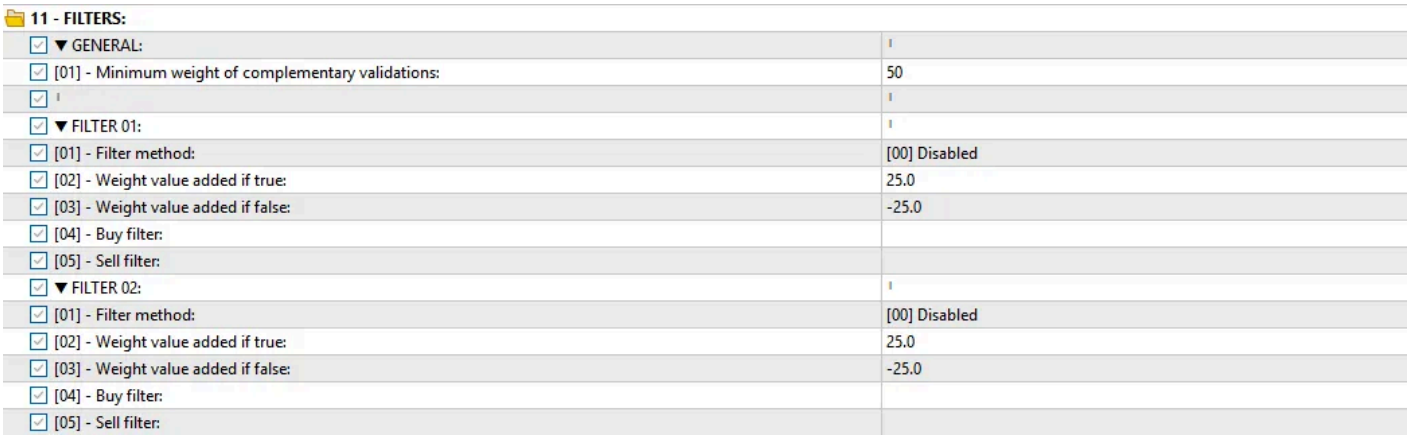
Em resumo, os filtros servem para:

- Evitar entradas impulsivas ou fora dos critérios definidos;
- Apoiar decisões com validações objetivas e consistentes;
- Adicionar camadas de verificação que aumentam a segurança da estratégia;

- Organizar e simplificar validações complexas, especialmente quando há múltiplas condições envolvidas.

Configurando os filtros

A configuração dos filtros é simples e intuitiva. Abaixo, você pode visualizar uma imagem que ilustra a estrutura dos filtros no sistema:



Cada filtro pode ser configurado de forma personalizada, de acordo com as necessidades do usuário. A estrutura de configuração se divide em duas partes: **Geral** e **Filtro**.

Geral

- **[01] - Peso mínimo de validação**: Define o valor mínimo que a soma dos pesos dos filtros deve atingir para que o sistema autorize a entrada da ordem.

Filtro

- **[01] - Tipo**: Define o comportamento do filtro durante a validação. As opções são:
  - **Desativado**: O filtro é ignorado no processo de validação.
  - **Obrigatório**:
    - O filtro é sempre executado e deve ser validado com sucesso para que a ordem seja autorizada.
    - Este filtro também contribui para a pontuação total quando validado com sucesso.
  - **Opcional**:
    - O filtro é executado apenas se a soma total dos pesos ainda não tiver atingido o valor necessário.
    - Seu resultado contribui para a pontuação, seja validado ou não.

- **[02] - Peso de Sucesso**: Valor somado à pontuação total caso o filtro seja validado com sucesso.
- **[03] - Peso de Falha**: Valor somado à pontuação caso o filtro não seja validado. Pode ser zero, positivo ou negativo, conforme a estratégia adotada.
- **[04 & 05] - Cálculos de Validação**: São expressões condicionais que o sistema executa quando o filtro for chamado.

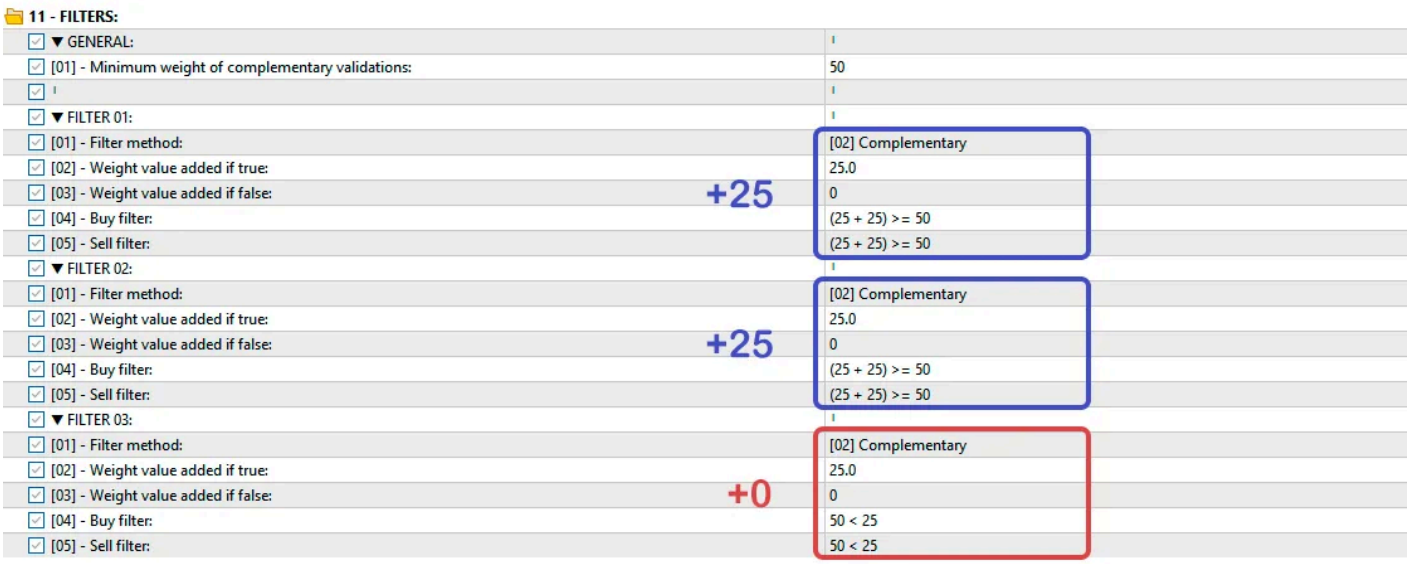
Entendendo os filtros

Os filtros são, essencialmente, validações em camadas. Cada validação tem um peso associado, que influencia a pontuação total da ordem. Se a soma dos pesos dos filtros atingir ou superar o valor do **peso mínimo**, a ordem é autorizada.

Primeiro exemplo:

Neste exemplo, os filtros são utilizados de forma complementar. O sistema considera uma entrada válida (compra ou venda) quando a soma dos pesos dos filtros validados atinge um valor mínimo.

Exemplo da configuração:



Se ao menos dois filtros forem validados com sucesso (2 × 25 = 50), o sistema considera que o critério mínimo foi atingido e confirma a entrada de compra ou venda.

Cálculos da Validação:

```
Filtro 01 (complementar): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 02 (complementar): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 03 (complementar): "50 < 25"         → falso      → +0

Soma dos pesos: 25 + 25 + 0 = 50
```

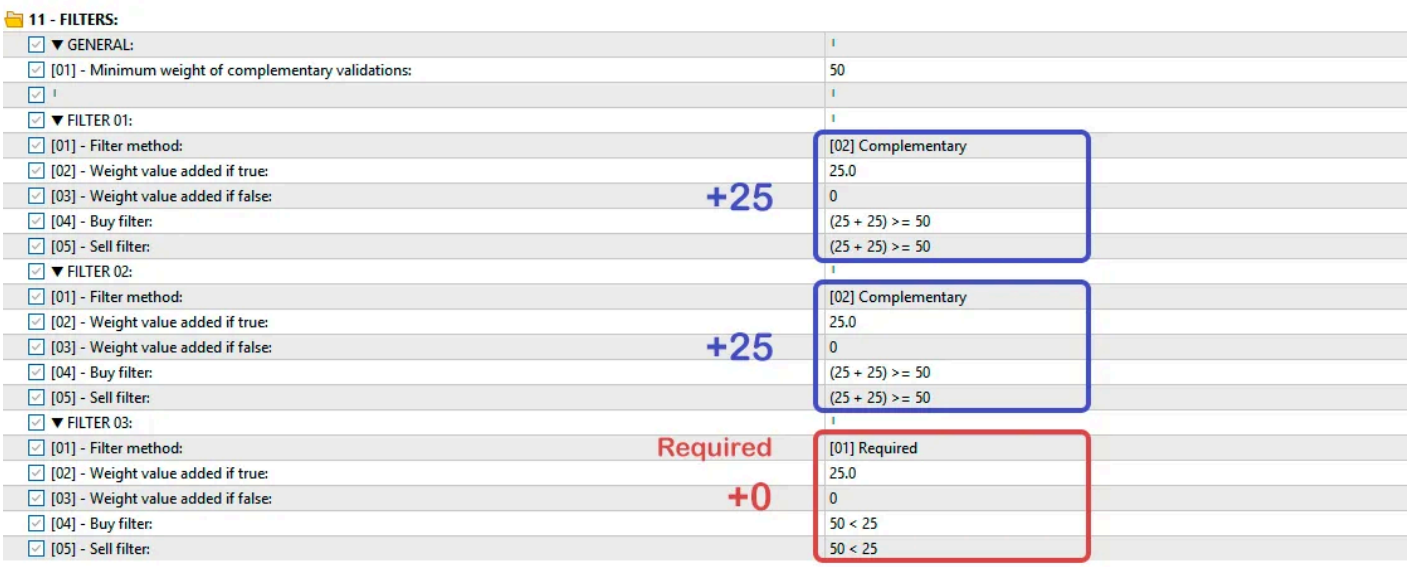
Resultado:
Como a soma dos pesos dos filtros validados atingiu o mínimo exigido (50), a entrada é confirmada.

Segundo exemplo:

Neste exemplo, são utilizados filtros complementares e obrigatórios. O sistema exige que:

1. A soma dos pesos dos filtros validados atinja o valor mínimo configurado.
2. Todos os filtros obrigatórios também sejam validados com sucesso.

Exemplo da configuração:



Mesmo que a soma dos filtros complementares atinja o peso mínimo, se algum filtro obrigatório falhar, a entrada não será confirmada.

Cálculos da Validação:

```
Filtro 01 (complementar): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 02 (complementar): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 03 (obrigatório):  "50 < 25"         → falso      → +0

Soma dos pesos: 25 + 25 + 0 = 50
```

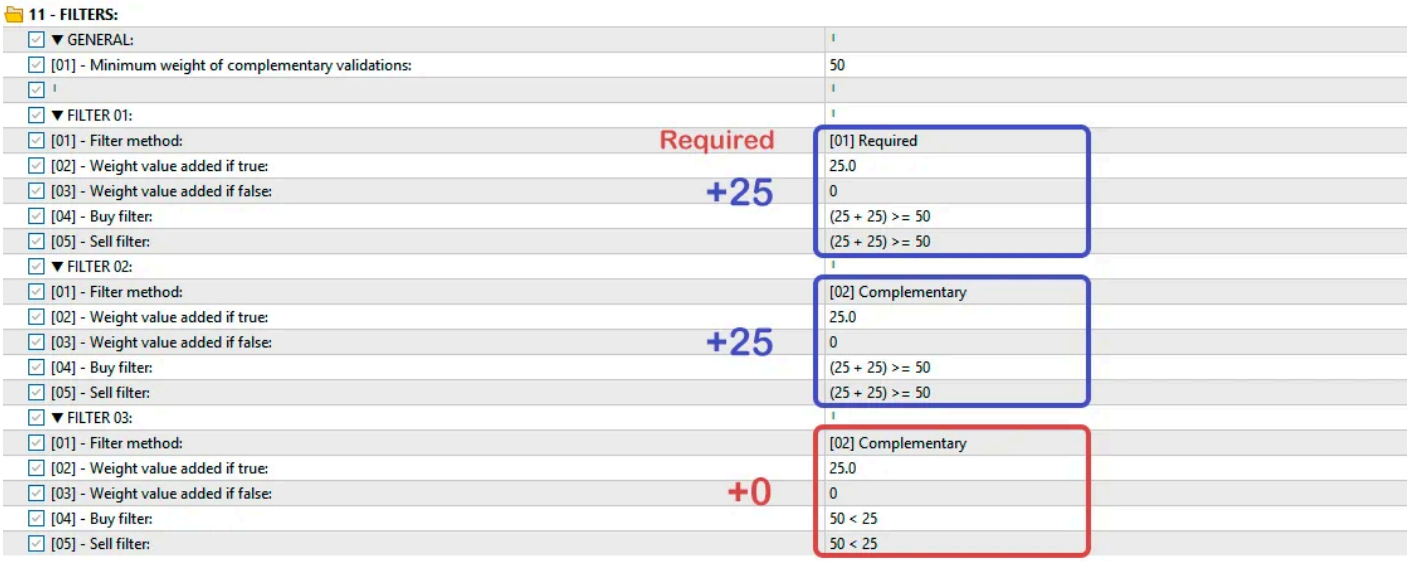
Resultado:
Embora a soma dos filtros validados tenha atingido o mínimo necessário (50), o filtro obrigatório não foi validado, portanto a entrada não é confirmada.

Terceiro exemplo:

Neste exemplo, são utilizados filtros complementares e obrigatórios. O sistema exige que:

1. A soma dos pesos dos filtros validados atinja o valor mínimo configurado.
2. Todos os filtros obrigatórios também sejam validados com sucesso.

Exemplo da configuração:



Mesmo que a soma dos filtros complementares atinja o peso mínimo, se algum filtro obrigatório falhar, a entrada não será confirmada.

Cálculos da Validação:

```
Filtro 01 (obrigatório):  "(25 + 25) >= 50" → verdadeiro → +25
Filtro 02 (complementar): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 03 (complementar): "50 < 25"         → falso      → +0

Soma dos pesos: 25 + 25 + 0 = 50
```

Resultado:
A soma dos filtros validados atingiu o mínimo necessário (50) e o filtro obrigatório foi validado. Portanto, a entrada é confirmada.

Quarto exemplo:

Neste exemplo, são utilizados apenas filtros obrigatórios. Nesse caso, o sistema não depende da soma mínima de pesos, mas sim da validação de todos os filtros obrigatórios.

Exemplo da configuração:

Mesmo que os filtros validados acumulem peso suficiente, a entrada não será confirmada se qualquer filtro obrigatório falhar.

Cálculos da Validação:

```
Filtro 01 (obrigatório): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 02 (obrigatório): "(25 + 25) >= 50" → verdadeiro → +25
Filtro 03 (obrigatório): "50 < 25"         → falso      → +0

Soma dos pesos: 25 + 25 + 0 = 50
```

Resultado:

Apesar da soma dos pesos atingir 50, o filtro obrigatório “Filtro 03” não foi validado. Como nem todos os filtros obrigatórios foram validados, a entrada não é confirmada.

Fragmentos

Nesta página, você vai conhecer os fragmentos disponíveis no ScriptBot: para que servem, como funcionam e de que forma podem facilitar suas automatizações. Entenda também como esses recursos aceleram o desenvolvimento de estratégias de forma prática e eficiente.

O que são fragmentos?

Os fragmentos (anteriormente chamados de “variáveis modeláveis”) são blocos de dados usados no ScriptBot para armazenar informações que podem ser aplicadas em qualquer parte da estratégia, como expressões lógicas, filtros, indicadores e eventos.

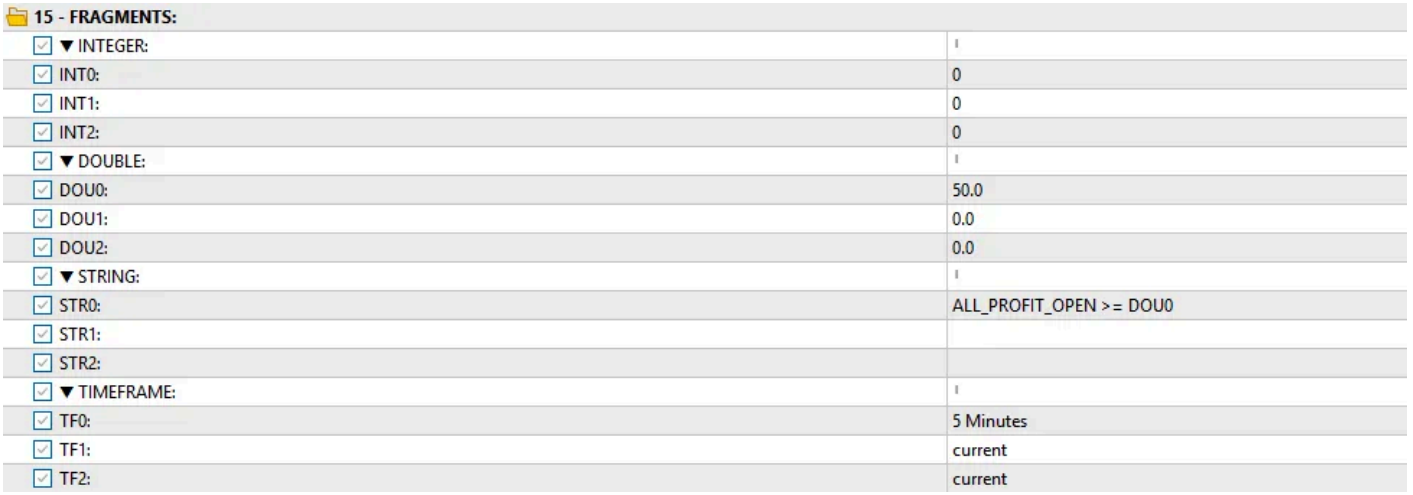
Eles ajudam a organizar melhor a construção das estratégias e facilitam o reaproveitamento de expressões. Além disso, os fragmentos resolvem uma limitação importante: a impossibilidade de realizar otimizações no Backtest do MetaTrader 5, causada pela forma como expressões e indicadores eram carregados. Essa restrição foi superada com a introdução dos fragmentos.

Durante a inicialização da estratégia, os fragmentos são automaticamente convertidos em valores fixos, já incorporados à lógica final. Isso melhora o desempenho e impede alterações durante a execução.

Tipos de fragmentos

Abaixo estão os principais tipos de fragmentos disponíveis:

- **INTEGER**: Números inteiros
- **DOUBLE**: Números decimais (ponto flutuante)
- **STRING**: Textos, expressões ou qualquer outro valor
- **TIMEFRAME**: Períodos gráficos (como M1, H1, etc)



15 - FRAGMENTS:	
▼ INTEGER:	
INT0:	0
INT1:	0
INT2:	0
▼ DOUBLE:	
DOU0:	50.0
DOU1:	0.0
DOU2:	0.0
▼ STRING:	
STR0:	ALL_PROFIT_OPEN >= DOU0
STR1:	
STR2:	
▼ TIMEFRAME:	
TF0:	5 Minutes
TF1:	current
TF2:	current

Usando fragmentos na expressão lógica

Para utilizar um fragmento é simples, basta utilizar o nome do fragmento na sua expressão lógica, como por exemplo:

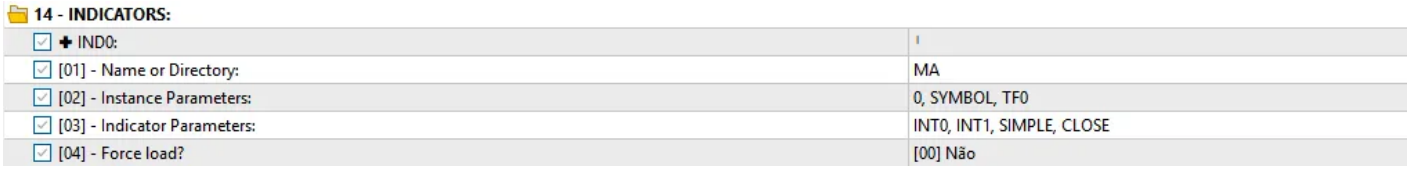
```
CLOSE[0] > CLOSE[1] && STR0
```

Se, por exemplo, o fragmento **STR0** tiver o valor `ALL_PROFIT_OPEN >= 50`, a expressão lógica será automaticamente convertida, durante a inicialização, para:

```
CLOSE[0] > CLOSE[1] && ALL_PROFIT_OPEN >= 50
```

Usando fragmentos nos indicadores

Os fragmentos podem ser utilizados em parâmetros de indicadores, como mostrado na imagem abaixo:



14 - INDICATORS:	
✚ IND0:	
[01] - Name or Directory:	MA
[02] - Instance Parameters:	0, SYMBOL, TF0
[03] - Indicator Parameters:	INT0, INT1, SIMPLE, CLOSE
[04] - Force load?	[00] Não

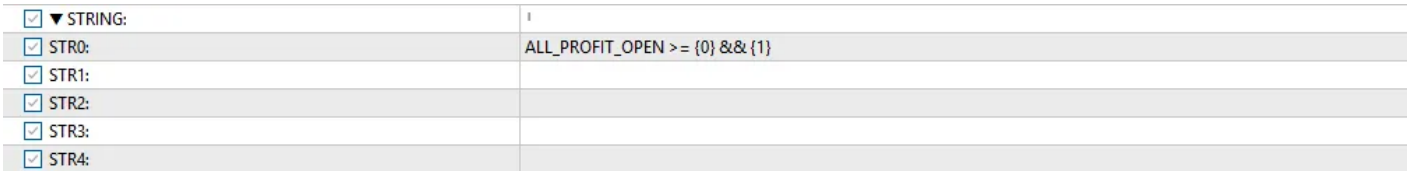
Podemos observar o uso dos fragmentos **TF0**, **INT0** e **INT1**. Esses fragmentos serão automaticamente convertidos durante a inicialização, o que permite realizar otimizações de estratégias usando o Backtest do MetaTrader 5 ou simplesmente facilitar a manutenção da estratégia.

Usando fragmentos com parâmetros

A possibilidade de utilizar parâmetros nos fragmentos do tipo STRING os torna ainda mais versáteis, permitindo que atuem como verdadeiras funções de conversão. Isso possibilita modificar o comportamento do fragmento de acordo com os parâmetros fornecidos, tornando as expressões mais flexíveis e dinâmicas.

Definição do fragmento

Vamos definir o fragmento **STR0** com o seguinte valor: `ALL_PROFIT_OPEN >= {0} && {1}`.
Os caracteres `{` e `}` delimitam os parâmetros. Dentro das chaves, você deve informar o índice do parâmetro, começando em zero. O índice `{0}` representa o primeiro parâmetro, `{1}` o segundo, e assim por diante.



Uso sem parâmetros

Abaixo está um exemplo de uso do fragmento sem parâmetros:

```
CLOSE[0] > CLOSE[1] && STR0
```

O fragmento **STR0** será automaticamente convertido para `CLOSE[0] > CLOSE[1] && ALL_PROFIT_OPEN >= {0} && {1}`, o que causará um erro na compilação da expressão lógica devido à ausência dos parâmetros necessários. Esse erro acontece porque, na definição do fragmento, os parâmetros `{0}` e `{1}` foram declarados como obrigatórios.

Uso com parâmetros

Abaixo está um exemplo de uso do fragmento com parâmetros:

```
CLOSE[0] > CLOSE[1] && STR0[50, Print["My age is: ", INT0]]
```

O fragmento **STR0** será convertido para `CLOSE[0] > CLOSE[1] && ALL_PROFIT_OPEN >= 50 && Print["My age is: ", INT0]`. Nenhum erro ocorrerá, pois todos os parâmetros foram fornecidos corretamente.

De modo geral, o uso de parâmetros permite criar expressões mais complexas e flexíveis. Além disso, possibilita modificar o comportamento da expressão de forma dinâmica, conforme os valores passados.

Indicadores

Nesta página, você irá aprender a como instalar um indicador no ScriptBot.

O que é um indicador?

Os indicadores são ferramentas analíticas essenciais para apoiar a tomada de decisão no mercado financeiro. Eles podem ser aplicados visualmente, diretamente nos gráficos, ou de forma automatizada, integrando seus dados a estratégias operacionais por meio de expressões lógicas no ScriptBot.

Esses indicadores são especialmente úteis na criação de filtros e na definição de pontos de entrada mais precisos. Além disso, possibilitam análises mais avançadas, que não seriam viáveis utilizando apenas expressões lógicas, trazendo maior flexibilidade e robustez às estratégias.

Indicador

Os indicadores são de extrema importância no desenvolvimento de estratégias no ScriptBot. Com eles, é possível analisar o comportamento do mercado, identificar tendências, detectar sinais de entrada e saída, além de otimizar a tomada de decisões automatizadas.

Parâmetros:

- **[01] Nome ou Diretório**:
  - Define o nome ou diretório do indicador:
- **[02] Parâmetros da Instância**:
  - Parâmetros da instancia do indicador, separados por vírgula.
  - **Valor padrão**: 0, SYMBOL, CURRENT
- **[03] Parâmetros do Indicador**:
  - Parâmetros do indicador, se houver e separados por vírgula.
- **[04] Carregamento forçado**:
  - O indicador é carregado imediatamente, mesmo que o indicador não seja utilizado.

Tipos de Indicadores

No ScriptBot, existem dois tipos de indicadores:

- **Indicadores internos**
  - São indicadores nativos do MetaTrader 5.
  - Sua instalação é simples, feita apenas pelo nome do indicador.
  - Você pode consultar a lista completa acessando: Lista de Indicadores
- **Indicadores externos**
  - Desenvolvidos por terceiros, não fazem parte do MetaTrader 5 por padrão.
  - A instalação requer o caminho completo do arquivo. Por exemplo: **"Market/Ultra ZigZag.ex5"**
  - Podem ser encontrados no MQL5 Market, na MQL5 CodeBase ou ainda ser criados pelo próprio usuário.

Instalando um Indicador

O processo de instalação varia conforme o tipo de indicador, mas em ambos os casos é simples, embora exija atenção.

Abaixo, você pode ver as duas formas de instalar um indicador:

> **Indicadores Internos**

> **Indicadores Externos**

Parâmetros da Instância

Os parâmetros da instância do indicador são obrigatórios e de extrema importância para o seu funcionamento adequado.

Eles servem como diretrizes que definem o comportamento do indicador, como: em qual janela ele será inserido, qual ativo será analisado e qual o período (tempo gráfico) será utilizado para obter os dados.

Esses parâmetros permitem que o indicador seja flexível, adaptando-se a diferentes contextos de análise. Por exemplo, é possível aplicar o indicador a diferentes ativos ou intervalos de tempo, sem a necessidade de alterar o esses dados diretamente no gráfico em operação, apenas modificando os parâmetros da instância.

Entendendo os Parâmetros:

```
0, SYMBOL, CURRENT
|    |        |-> Período de Análise: Lista de Períodos
|    |----------> Tipo de Ativo: Lista de Ativos
|---------------> Janela de Análise: Lista de Janelas
```

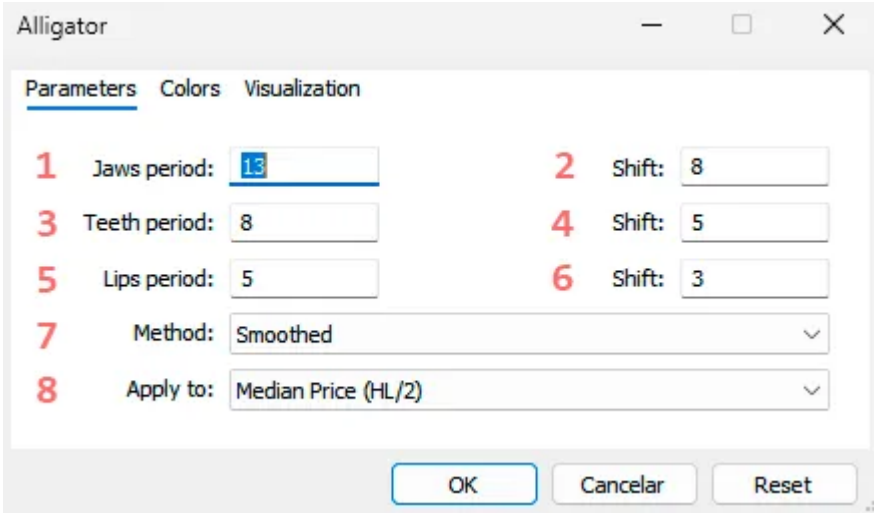
Parâmetros do Indicador

Os parâmetros dos indicadores não são obrigatórios. Caso nenhum parâmetro seja informado, o indicador será iniciado com os valores padrão.

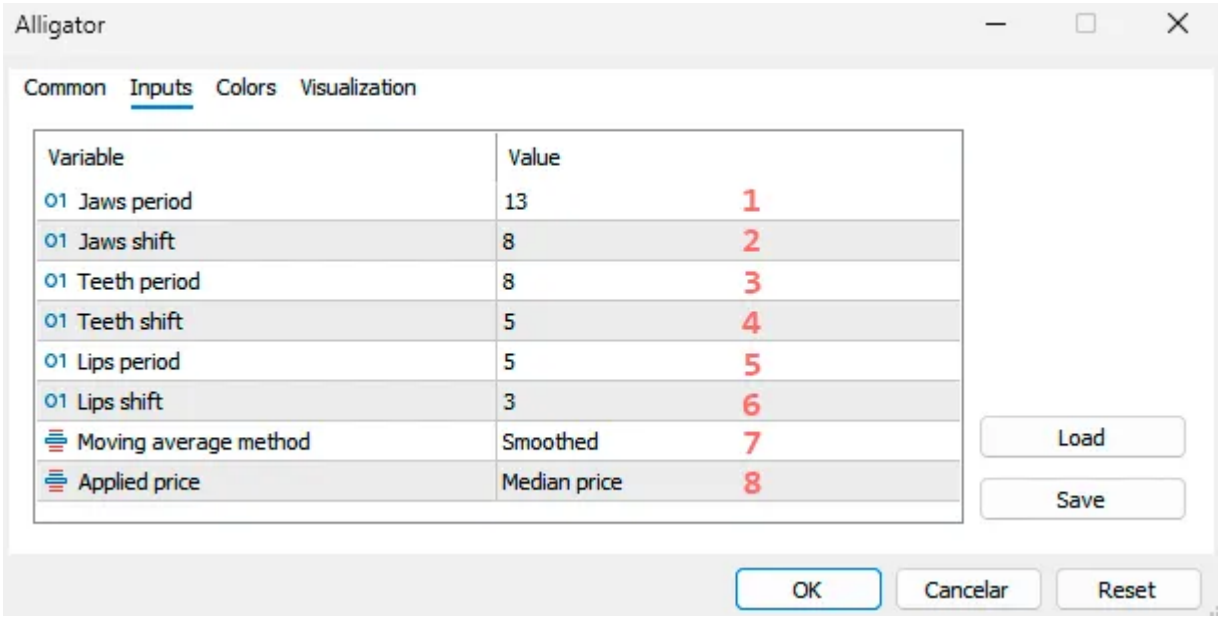
Se optar por configurar os parâmetros manualmente, é fundamental respeitar a ordem correta. A sequência dos parâmetros deve seguir exatamente a estrutura esperada pelo indicador, caso contrário, ele poderá funcionar de forma incorreta.

Para identificar os parâmetros corretamente, basta adicionar o indicador ao gráfico. O layout da janela de parâmetros pode variar, e o indicador pode apresentar dois formatos diferentes de entrada de dados, dependendo da sua implementação.

Primeiro formato:



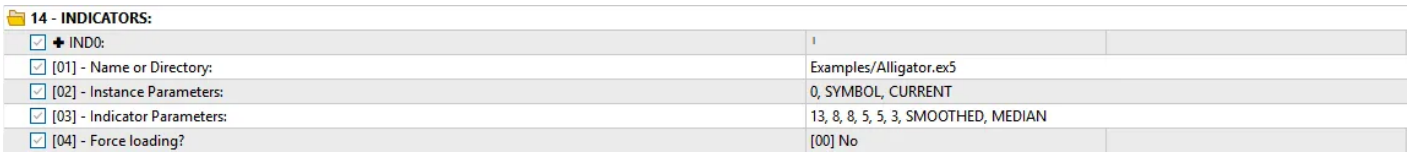
Segundo formato:



Ordens dos Parâmetros

Agora que sabemos a ordem correta em que os argumentos devem ser adicionados, é importante observar que o indicador Alligator possui dois parâmetros que utilizam enumerações: um para o método de média móvel (**Moving average method**) e outro para o tipo de preço a ser aplicado (**Applied price**).

Os parâmetros devem ser adicionados na ordem correta, conforme mostra a imagem abaixo:



Entendendo os Parâmetros:

```
13, 8, 8, 5, 5, 3, SMOOTHED, MEDIAN
|   |  |  |  |  |      |        |-> Applied price: Lista de Preços
|   |  |  |  |  |      |----------> Moving average method: Lista de Metodos
|   |  |  |  |  |------------------> Lips shift
|   |  |  |  |---------------------> Lips period
|   |  |  |-----------------------> Teeth shift
|   |  |--------------------------> Teeth period
|   |-----------------------------> Jaws shift
|---------------------------------> Jaws period
```

Parâmetros de Texto

Muitos indicadores podem incluir parâmetros do tipo texto, utilizados para armazenar expressões, nomes, descrições ou outras informações.

Para facilitar o uso desses parâmetros, foi introduzido o operador #. Esse operador é exclusivo para parâmetros de indicadores e atua como um delimitador de sequência, garantindo que todo o conteúdo entre os símbolos # seja tratado como texto literal, sem sofrer qualquer tipo de interpretação ou modificação.

Veja, por exemplo, a seguinte configuração fictícia:

Entendendo os Parâmetros:

[01] Nome ou Diretório:	Examples/ExpIndicator.ex5
[02] Parâmetros da Instância:	0, SYMBOL, CURRENT
[03] Parâmetros do Indicador:	22, 0, #CLOE[0] > OPEN[0, M5]#, SMOOTHED, MEDIAN
[04] Carregamento forçado:	false

Observe que o parâmetro `#CLOE[0] > OPEN[0, M5]#` utiliza o operador #. Nesse caso, todo o conteúdo entre os símbolos # é interpretado como texto literal, sem qualquer tipo de modificação ou interpretação adicional.

Set por Arquivo

O carregamento por arquivo .set pode não ser um processo simples, mas é fundamental para configurar corretamente os parâmetros de alguns indicadores. Embora para a maioria dos indicadores essa etapa não seja necessária, há casos em que eles possuem muitos parâmetros ou configurações complexas, tornando inviável a instalação manual tradicional. Nesses casos, utilizar arquivos .set é a maneira mais eficiente e prática de garantir a configuração adequada.

Antes de começar, é importante entender algumas limitações:

- Alguns indicadores do MetaTrader 5 não permitem que suas configurações sejam salvas.
- No MT5, existe um limite máximo de parâmetros que um robô pode carregar. Se o arquivo `.set` contiver mais de 62 parâmetros, o robô carregará apenas os primeiros 62. Caso esse limite seja ultrapassado, uma mensagem de erro será exibida. O indicador ainda será carregado, mas os parâmetros excedentes serão ignorados e pode ocorrer um comportamento diferente do esperado.

Salvando o Arquivo set

1. Primeiro, escolhemos um indicador, configuramos suas opções e salvamos o arquivo `.set` em um local de nossa preferência no computador.
   Salve o arquivo com um nome descritivo, por exemplo:

   ```
   my-set-alligator.set
   ```

2. Antes de mover o arquivo `.set` para a pasta correta, execute pelo menos uma vez o robô em um ambiente de teste (como o testador de estratégia do MetaTrader).
   Isso é necessário para que o sistema crie automaticamente as pastas internas do robô, incluindo a pasta onde os sets serão armazenados.

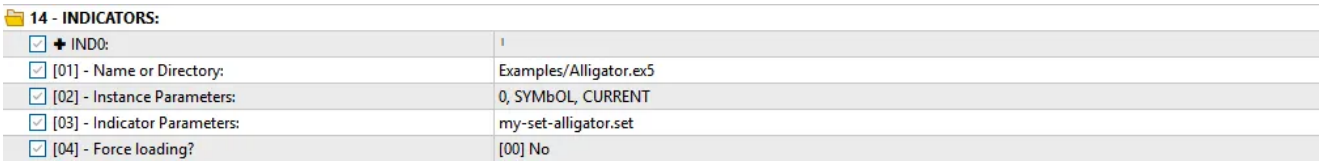
3. Agora que o arquivo está salvo e as pastas do robô foram criadas, vamos mover o arquivo `.set` para o local correto:
   Existe duas formas diferentes de fazer isso:

   > **Localizando a pasta padrão:**

   > **Acessando usando o executador: Windows + R**

   Independendo da forma escolhida, copie o arquivo `.set` para a pasta de sets que foi localizada.

4. Com o arquivo `.set` no lugar certo, agora é hora de carregá-lo dentro do robô.
   Em vez de preencher os parâmetros manualmente, digite o nome exato do arquivo, incluindo a extensão `.set` nos parâmetros do indicador.
   Isso fará com que o robô carregue automaticamente todos os parâmetros configurados no arquivo.
   Exemplo visual:



5 Pronto! O indicador já está com os parâmetros devidamente configurados.

Usando Fragmentos

Para utilizar os **Fragmentos** em um arquivo .set, é necessário abrir o arquivo .set. Nele, você pode alterar os valores e substituir pelos nomes dos fragmentos.
Os arquivos .set salvos pelo Backtest podem apresentar parâmetros com várias barras (||), o que pode parecer confuso.
No entanto, tudo que estiver depois das barras são totalmente descartáveis e podem ser removidas sem causar qualquer erro de execução.

O exemplo abaixo mostra como utilizar fragmentos em um arquivo .set:

```
; saved on 2025.08.06 01:18:32
; this file contains input parameters for testing Alligator custom indicator
; to use it in the strategy tester, click Load in the context menu of the Inputs tab
;
InpJawsPeriod=INT0
InpJawsShift=INT1
InpTeethPeriod=INT2
InpTeethShift=5||5||1||50||N
InpLipsPeriod=5||5||1||50||N
InpLipsShift=3||3||1||30||N
InpMAMethod=2||0||0||3||N
InpAppliedPrice=5||1||0||7||N
```

Gatilhos

Nesta página, você encontrará informações sobre os gatilhos de entrada, saída e atualização disponíveis no ScriptBot.

O que é um gatilho?

Um gatilho é um ponto de execução automatizado que verifica, em intervalos regulares, uma condição lógica definida pelo usuário. Essas condições são expressas por meio de expressões lógicas e avaliadas conforme a **Frequência Configurada**.

No ScriptBot, os gatilhos são o mecanismo central de controle sobre ordens e posições, permitindo que o robô atue de forma autônoma, seguindo as regras definidas pelo usuário. Os principais tipos de gatilhos são:

- **Gatilhos de entrada:** disparam ordens de compra ou venda quando a condição de entrada for satisfeita;
- **Gatilhos de saída:** executam a saída do mercado, encerrando uma posição ou cancelando uma ordem;
- **Gatilhos de atualização:** ajustam ordens ou posições existentes, com base em condições atualizadas do mercado.

Há diferentes tipos de gatilhos, cada um com um papel específico no fluxo da estratégia. Eles oferecem controle granular e flexibilidade para automatizar decisões com precisão e segurança.

Gatilhos de entrada

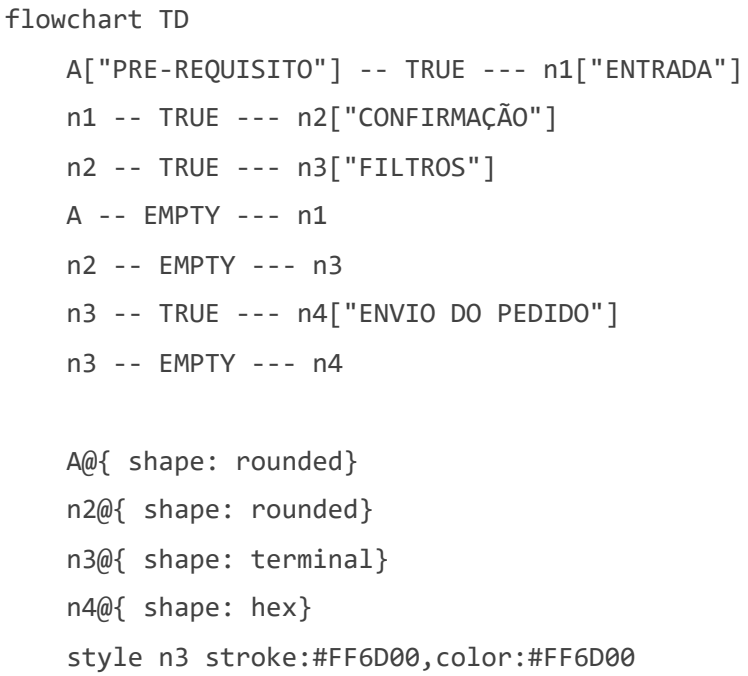
Os gatilhos de entrada são responsáveis por realizar cálculos e avaliar condições para determinar se uma ordem de compra ou venda deve ser executada. Esses gatilhos possuem um fluxo de execução, o que os torna mais flexíveis na criação das estratégias.

Abaixo está uma imagem contendo os gatilhos de entrada disponíveis no ScriptBot.

☑ ▼ PRE-REQUISITE:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑		
☑ ▼ ENTRY:		
☑ - Calculations:	IND0[0, 2] < IND1[0, 2] && IND0[0, 1] >= IND1[0, 1]	
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑		
☑ ▼ CONFIRMATION:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑		
☑ ▼ EXIT:		
☑ - Calculations:		
☑ - Calculations:		
☑ - Calculations:		
☑		
☑ ▼ INDIVIDUAL EXIT:		
☑ - Calculations:		
☑ - Calculations:		

Os gatilhos seguem uma ordem de execução baseada em uma lógica condicional que define o fluxo da estratégia. Alguns são obrigatórios, enquanto outros são opcionais, mas todos, quando combinados, compõem o funcionamento completo da estratégia.

O fluxo de operações do ScriptBot segue a seguinte ordem:



O fluxo de execução é composto por quatro etapas principais, organizadas sequencialmente:

1. **PRE-REQUISITO**
   - Primeira verificação da sequência.
   - Responsável por fazer verificações iniciais, usadas para cálculos simples e rápidos.
   - Se validado como **verdadeiro**, o fluxo prossegue para o próximo gatilho.
   - **Opcional**: caso esteja vazio, o processo segue diretamente para o **ENTRADA**.

2. **ENTRADA**
   - Gatilho de entrada da estratégia.
   - Responsável por realizar o cálculo da lógica principal da estratégia.
   - Se validado como **verdadeiro**, leva à próxima etapa (**CONFIRMAÇÃO**).

3. **CONFIRMAÇÃO**
   - Confirmação adicional antes da execução.
   - Responsável por fazer cálculos adicionais, se necessário.
   - **Opcional**: se não estiver configurado, o fluxo segue diretamente para **FILTROS**.

4. **FILTROS**
   - Os filtros aplicam múltiplas camadas de validações para aumentar a **confiabilidade** da estratégia.

- Cada filtro realiza uma **validação específica** e pode ter **pesos diferentes**, permitindo uma análise mais refinada.
- A documentação completa sobre os filtros podem ser encontradas em Filtros.
- **Opcional**: se não estiver configurado, o fluxo segue diretamente para **ENVIO DO PEDIDO**.

5 **ENVIO DO PEDIDO**

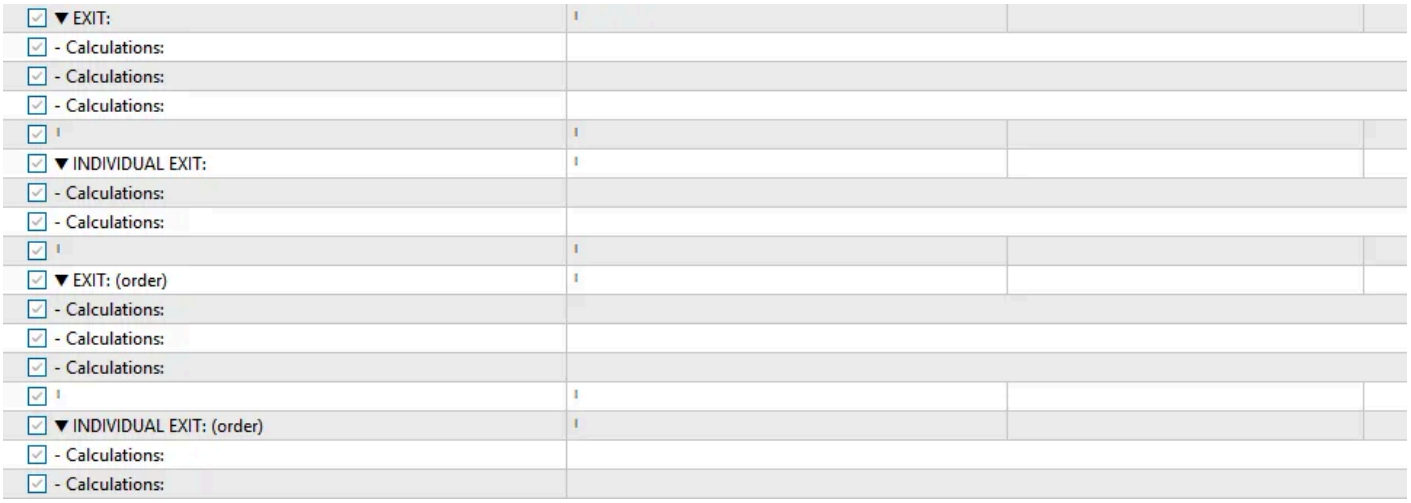
- Etapa final do fluxo.
- Responsável pelo envio da operação propriamente dita.

> **Dica**
>
> Confira a página de Configurações de Operações para complementar seu entendimento sobre os gatilhos e dar os primeiros passos na criação da sua estratégia.

Gatilhos de saída

Os gatilhos de saida são responsáveis por realizar cálculos e avaliar condições para determinar se uma ordem de compra ou venda deve ser removida.

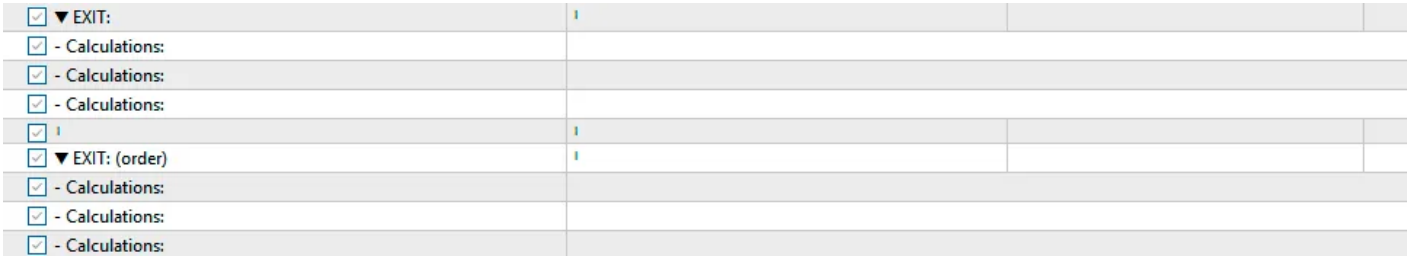
Abaixo está uma imagem contendo os gatilhos de saída disponíveis no ScriptBot.



No ScriptBot, existem quatro tipos diferentes de gatilhos de saída, cada um responsável por um tipo específico de finalização.

Gatilhos de saída totais

Os gatilhos de **saída total** são responsáveis por tentar encerrar todas as **posições** e **ordens** atualmente abertas, de acordo com as condições definidas pelo usuário.



O fluxo de saída segue a seguinte ordem:

```
flowchart TD
    A["SAIDA"] -- TRUE --- n3["FECHAMENTO TOTAL"]
    n3@{ shape: hex}
```

O fluxo é resumido em duas etapas:

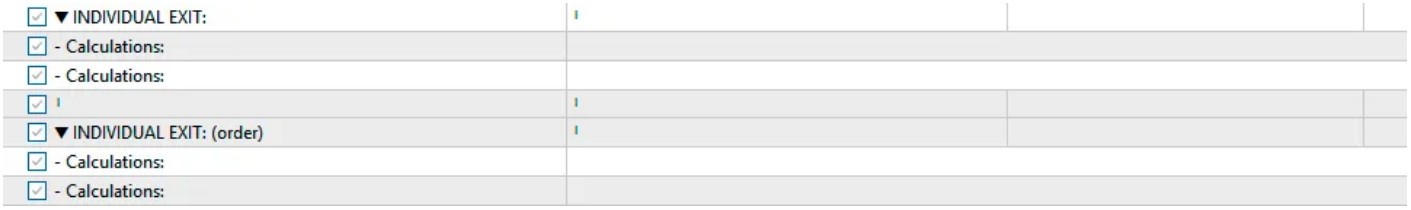
1. **SAÍDA**
   - Representa o cálculo principal para realizar o fechamento.
   - Tem como função verificar se é possível encerrar todas as posições ou ordens em aberto.
   - Se o resultado for **verdadeiro**, o processo avança para a etapa seguinte: **FECHAMENTO TOTAL**.
   - **Importante**: se este campo estiver vazio, nenhum cálculo de fechamento será executado.

2. **FECHAMENTO TOTAL**
   - Última etapa do fluxo.
   - Responsável por encerrar definitivamente todas as posições e ordens abertas.

Gatilhos de saída individuais

Os gatilhos de **saída individual** são responsáveis por tentar encerrar, de forma isolada, as **posições** e **ordens** atualmente abertas.

Esse tipo de cálculo realiza uma **iteração separada** para cada posição ou ordem, permitindo que o usuário defina **condições específicas** para o fechamento de cada uma delas.



O fluxo de saída segue a seguinte ordem:

```
flowchart TD
    n5["ITERADOR"] --- n4["Small Circle"]
    A["SAIDA INDIVIDUAL"] -- TRUE --- n3["FECHAMENTO"]
    n3 --- n4
    A -- FALSE --- n4
    n5 --- n6>"ORDEM OU POSIÇÃO"]
    n6 --- A
    n3@{ shape: hex}
    n5@{ shape: loop-limit}
    n4@{ shape: sm-circ}
```

O fluxo é resumido nas seguintes etapas:

1. **ITERADOR**
   - Inicia o ciclo do fluxo.
   - Controla a repetição sobre uma coleção de ordens ou posições.
   - Garante que cada item seja processado individualmente, mantendo o controle do avanço.

2. **SELEÇÃO DA ORDEM OU POSIÇÃO**
   - Etapa responsável por selecionar a próxima ordem ou posição a ser processada.
   - Responsável por alimentar as Variáveis Locais.
   - Essa seleção alimenta a etapa seguinte: **SAÍDA INDIVIDUAL**.

3. **SAÍDA INDIVIDUAL**

- Realiza a verificação principal para determinar se a ordem ou posição pode ser encerrada.
- Se o resultado for **verdadeiro**, o fluxo segue para a etapa de **FECHAMENTO**.
- Se **falso**, o item não será encerrado e o fluxo retorna diretamente ao **ITERADOR** para processar o próximo.

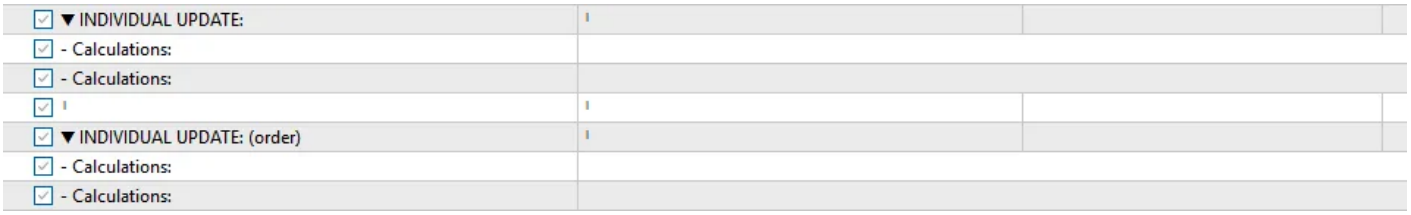
4 **FECHAMENTO**

- Etapa final do processamento individual.
- Encerra a ordem ou posição que foi validada na etapa anterior.
- Após o fechamento, o fluxo retorna ao **ITERADOR** para continuar o ciclo com o próximo item.

Gatilhos de atualização individuais

Os gatilhos de **atualização individual** são responsáveis por selecionar, de forma isolada, um **pedido**, bem como as **posições** e **ordens** que estão abertas no momento.

Esse tipo de cálculo realiza uma **iteração separada** para cada posição ou ordem, permitindo que o usuário manipule este pedido, modificando seus dados, como stop, take e preços.



O fluxo de atualização segue a seguinte ordem:

```
flowchart TD
    n5["ITERADOR"] --- n4["Small Circle"] & n6>"ORDEM OU POSIÇÃO"]
    A["ATUALIZADOR INDIVIDUAL"] --- n4
    n6 --- A
    n5@{ shape: loop-limit}
    n4@{ shape: sm-circ}
```

O fluxo é resumido nas seguintes etapas:

1. **ITERADOR**
   - Inicia o ciclo do fluxo.
   - Controla a repetição sobre uma coleção de ordens ou posições.
   - Garante que cada item seja processado individualmente, mantendo o controle do avanço.

2. **SELEÇÃO DA ORDEM OU POSIÇÃO**
   - Etapa responsável por selecionar a próxima ordem ou posição a ser processada.
   - Responsável por alimentar as Variáveis Locais.
   - Essa seleção alimenta a etapa seguinte: **ATUALIZADOR INDIVIDUAL**.

3. **ATUALIZADOR INDIVIDUAL**
   - Realiza os cálculos condicionais da ordem ou posição.
   - Após a manipulação, o fluxo retorna ao **ITERADOR** para continuar o ciclo com o próximo item.

Constantes

Nesta página, você encontrará as constantes disponíveis no ScriptBot. Uma constante é um tipo especial de variável cujo valor permanece o mesmo do início ao fim da execução. No ScriptBot, constantes também são utilizadas como variáveis globais, podendo ser acessadas em todo o projeto, e não apenas em expressões lógicas.

Janelas

> Constantes referente ao tipo da janela com qual o indicador vai ser anexado.

VARIÁVEIS	RETORNOS
`NEW`	Cria o indicador em uma nova janela.
`0`	Mesma janela do gráfico.
`1 … 2 … 3 …`	Cria o indicador em uma nova janela ou agrupa o indicador caso a janela já foi criada.
`-1`	Cria o indicador mas sem aparecer no gráfico.

Ativos

> Constantes referente ao tipo de ativo que o indicador vai ser anexado.

VARIÁVEIS	RETORNOS
`SYMBOL`	Ativo que o robô esta anexado.
`REAL`	Ativo que o robô enviará as ordens.
`”Nome do ativo”`	Ativo customizado.

Cores

> Constantes referente as cores de um objeto.

CORES	RESULTADOS
clrBlack	#000000
clrDarkGreen	#004000
clrDarkSlateGray	#2F4F4F
clrOlive	#808000
clrGreen	#018000
clrTeal	#008080
clrNavy	#000080
clrPurple	#800080
clrMaroon	#800000
clrIndigo	#4B0082
clrMidnightBlue	#191970
clrDarkBlu	#00008B

CORES	RESULTADOS
e	
clrDarkOliveGreen	#556B2F
clrSaddleBrown	#8B4513
clrForestGreen	#228B22
clrOliveDrab	#6B8E23
clrSeaGreen	#2E8B57
clrDarkGoldenrod	#B8860B
clrDarkSlateBlue	#483D8B
clrSienna	#A0522D

CORES	RESULTADOS
clrMediumBlue	#0000CD
clrBrown	#A52A2A
clrDarkTurquoise	#00CED1
clrDimGray	#696969
clrLightSeaGreen	#20B2AA
clrDarkViolet	#9400D3
clrFireBrick	#B22222
clrMediumVioletRed	#C71585
clrMediumSeaGreen	#3CB371

CORES	RESULTADOS
clrChocolate	#D2691E
clrCrimson	#DC143C
clrSteelBlue	#4682B4
clrGoldenrod	#DAA520
clrMediumSpringGreen	#00FA9A
clrLawnGreen	#7CFC00
clrCadetBlue	#5F9EA0
clrDarkOrchid	#9932CC
clrYellowGreen	#9ACD32
clrLimeGre	#32CD32

CORES	RESULTADOS
en	
clrOrangeRed	#FF4500
clrDarkOrange	#FF8C00
clrOrange	#FFA500
clrGold	#FFD700
clrYellow	#FFFF00
clrChartreuse	#7FFF00
clrLime	#00FF00
clrSpringGreen	#00FF7F
clrAqua	#00FFFF
clrDeepSkyBlue	#00BFFF
clrBlue	#0000FF

CORES	RESULTADOS
clrMagenta	#FF00FF
clrRed	#FF0000
clrGray	#808080
clrSlateGray	#708080
clrPeru	#CD853F
clrBlueViolet	#8A2BE2
clrLightSlateGray	#778899
clrDeepPink	#FF1493
clrMediumTurquoise	#48D1CC
clrDodgerBlue	#1E90FF

CORES	RESULTADOS
clrTurquoise	#40E0D0
clrRoyalBlue	#4169E1
clrSlateBlue	#6A5ACD
clrDarkKhaki	#BDB76B
clrIndianRed	#CD5C5C
clrMediumOrchid	#BA55D3
clrGreenYellow	#ADFF2F
clrMediumAquamarine	#66CDAA
clrDarkSeaGreen	#8FBC8F

CORES	RESULTADOS
clrTomato	#FF6347
clrRosyBrown	#BC8F8F
clrOrchid	#DA70D6
clrMediumPurple	#9370DB
clrPaleVioletRed	#DB7093
clrCoral	#FF7F50
clrCornflowerBlue	#6495ED
clrDarkGray	#A9A9A9
clrSandyBrown	#F4A460
clrMediumS	#7B68EE

CORES	RESULTADOS
lateBlue	
clrTan	#D2B48C
clrDarkSalmon	#E9967A
clrBurlyWood	#DEB887
clrHotPink	#FF69B4
clrSalmon	#FA8072
clrViolet	#EE82EE
clrLightCoral	#F08080
clrSkyBlue	#87CEEB
clrLightSalmon	#FFA07A
clrPlum	#DDA0DD
clrKhaki	#F0E68C

CORES	RESULTADOS
clrLightGreen	#90EE90
clrAquamarine	#7FFFD4
clrSilver	#C0C0C0
clrLightSkyBlue	#87CEFA
clrLightSteelBlue	#B0C4DE
clrLightBlue	#ADD8E6
clrPaleGreen	#98FB98
clrThistle	#D8BFD8
clrPowderBlue	#B0E0E6
clrPaleGol	#EEE8AA

CORES	RESULTADOS
denrod	
clrPaleTurquoise	#AFEEEE
clrLightGray	#F0F8FF
clrWheat	#F5DEB3
clrNavajoWhite	#FFDEAD
clrMoccasin	#FFE4B5
clrLightPink	#FFB6C1
clrGainsboro	#DCDCDC
clrPeachPuff	#FFDAB9
clrPink	#FFC0CB

CORES	RESULTADOS
clrBisque	#FFE4C4
clrLightGoldenrod	#EEDC82
clrBlanchedAlmond	#FFEBCD
clrLemonChiffon	#FFFACD
clrBeige	#F5F5DC
clrAntiqueWhite	#FAEBD7
clrPapayaWhip	#FFEFD5
clrCornsilk	#FFF8DC
clrLightYellow	#FFFFE0
clrLightCy	#E0FFFF

CORES	RESULTADOS
an	
clrLinen	#FAF0E6
clrLavender	#E6E6FA
clrMistyRose	#FFE4E1
clrOldLace	#FDF5E6
clrWhiteSmoke	#F5F5F5
clrSeashell	#FFF5EE
clrIvory	#FFFFF0
clrHoneydew	#F0FFF0
clrAliceBlue	#F0F8FF
clrLavenderBlush	#FFF0F5

CORES	RESULTADOS
clrMintCream	#F5FFFA
clrSnow	#FFFAFA
clrWhite	#FFFFFF

Enumeradores

Nesta página, você encontrará os enumeradores utilizados no ScriptBot para representar diferentes categorias de valores. Esses enumeradores facilitam a definição de parâmetros em indicadores e funções.

Preços

> **ENUM_PRICES**
> Enumerador referente ao método de preço de um indicador.

VARIÁVEIS	SIGLAS
`CLOSE`	1
`OPEN`	2
`HIGH`	3
`LOW`	4
`MEDIAN`	5
`TYPICAL`	6
`WEIGHTED`	7

Tempo gráfico

> **ENUM_TIMEFRAMES**
> Enumerador referente aos tempos gráficos.

VARIÁVEIS	TEMPOS
`CURRENT`	Tempo gráfico do robô.
`M1`	1 minuto.
`M2`	2 minutos.

VARIÁVEIS	TEMPOS
`M3`	3 minutos.
`M4`	4 minutos.
`M5`	5 minutos.
`M6`	6 minutos.
`M10`	10 minutos.
`M12`	12 minutos.
`M15`	15 minutos.
`M20`	20 minutos.
`M30`	30 minutos.
`H1`	1 hora.
`H2`	2 horas.
`H3`	3 horas.
`H4`	4 horas.
`H6`	6 horas.
`H8`	8 horas.
`H12`	12 horas.
`D1`	1 dia.
`W1`	1 semana.
`MN1`	1 mês.

Métodos

> **ENUM_METHODS**
> Enumerador referente ao método do cálculo de um indicador.

VARIÁVEIS	SIGLAS
SIMPLE	0
EXPONENTIAL	1
SMOOTHED	2
LINEAR	3

Volumes

> **ENUM_VOLUMES**
> Enumerador referente ao método do cálculo de um indicador de volume.

VARIÁVEIS	SIGLAS
TICK	0
REAL	1

Stochastic

> **ENUM_STOCHASTIC**
> Enumerador referente ao método de preço de um indicador de stochatic.

VARIÁVEIS	SIGLAS
LOWHIGH	0
CLOSECLOSE	1

Tipo de posição

> **ENUM_TRADE**
> Enumerador referente ao tipo de posição/ordem.

VARIÁVEIS	RETORNOS
TYPE_ALL	Todas as posições.
TYPE_BUY	Apenas posições de compra.
TYPE_SELL	Apenas posições de venda.

Ação de fechamento

> **ENUM_MARKET_ACTION**
> Enumerador referente ao tipo de ação de fechamento.

VARIÁVEIS	RETORNOS
TYPE_ALL	Todos os tipos de fechamento.
TYPE_IN	Apenas posições de abertura (in). Utiliza a ordem original de abertura da posição, em vez da ordem que foi usada para o fechamento.
TYPE_OUT	Apenas posições de fechamento (out).

Tipo de retorno

> **ENUM_MARKET_GET**
> Enumerador referente ao tipo de retorno.

VARIÁVEIS	RETORNOS
TYPE_ALL	Todos os tipos de retorno.
TYPE_POSITIVE	Apenas retornos positivo.
TYPE_NEGATIVE	Apenas retornos negativo.

Histórico

> **ENUM_TIME_HISTORIC**
> Enumerador referente ao tipo de tempo histórico.

VARIÁVEIS	RETORNOS
`TYPE_DAY`	Apenas o histórico do dia.
`TYPE_WEEK`	Apenas o histórico da semana.
`TYPE_MONTH`	Apenas o histórico do mês.
`TYPE_YEAR`	Apenas o histórico do ano.
`TYPE_ALL`	Histórico completo.

Tempo

> **ENUM_TIME**
> Enumerador referente ao tipo de tempo.

VARIÁVEIS	RETORNOS
`TYPE_SEC`	Apenas os segundos.
`TYPE_MIN`	Apenas os minutos.
`TYPE_HOUR`	Apenas as horas.
`TYPE_DAY`	Apenas o dia.
`TYPE_DAY_WEEK`	Apenas o dia da semana.
`TYPE_DAY_YEAR`	Apenas o dia do ano.
`TYPE_MONTH`	Apenas o mês.

VARIÁVEIS	RETORNOS
TYPE_YEAR	Apenas o ano.

Tipo de modificação

> **ENUM_TRADE_MODIFY**
> Enumerador referente a forma que a negociação será modificada.

VARIÁVEIS	RETORNOS
TYPE_PRICE	Será usado o preço.
TYPE_POINT	Será usado o ponto.

Propriedade da posição (double)

> **ENUM_POS_DOUBLE**
> Enumerador referente a propriedade da posição.

VARIÁVEIS	RETORNOS
TYPE_VOLUME	Volume de uma posição.
TYPE_OPEN	Preço de abertura.
TYPE_SL	Preço do stop.
TYPE_TP	Preço do take.
TYPE_CURRENT	Preço atual.
TYPE_SWAP	Swap acumulativo.

VARIÁVEIS	RETORNOS
TYPE_PROFIT	Lucro corrente.

Propriedade da posição (int)

> **ENUM_POS_INT**
> Enumerador referente a propriedade da posição.

VARIÁVEIS	RETORNOS
TYPE_TICKET	Bilhete da posição.
TYPE_TIME	Hora de abertura de uma posição.
TYPE_TIME_MSC	Posição de tempo de abertura em milissegundos desde 01.01.1970.
TYPE_TIME_UPDATE	Posição de tempo de alteração.
TYPE_TIME_UPDATE_MSC	Posição de tempo de alteração em milissegundo desde 01.01.1970.
TYPE_TYPE	Tipo de posição. (0: Compra, 1: Venda)
TYPE_MAGIC	Número mágico de uma posição.
TYPE_IDENTIFIER	Identificador de uma posição.
TYPE_REASON	Razão para a abertura da posição. (0: Terminal desktop, 1: Aplicativo móvel, 2: Plataforma web, 3: Expert Advisor, script ou outro código MQL5)

Propriedade da posição (string)

> **ENUM_POS_STRING**
> Enumerador referente a propriedade da posição.

VARIÁVEIS	RETORNOS
TYPE_SYMBOL	Simbolo da posição.
TYPE_COMMENT	Comentario da posição.
TYPE_EXTERNAL_ID	ID de posição no sistema externo de negociação (na bolsa de valores).

Propriedade da ordem (double)

> **ENUM_ORDER_DOUBLE**
> Enumerador referente a propriedade da ordem.

VARIÁVEIS	RETORNOS
TYPE_VOLUME_INITIAL	Volume inicial da ordem.
TYPE_VOLUME	Volume atual da ordem.
TYPE_OPEN	Preço de abertura da ordem.
TYPE_SL	Preço do stop da ordem.
TYPE_TP	Preço do take da ordem.
TYPE_CURRENT	Preço atual da ordem.
TYPE_STOP_LIMIT	Preço do stop limit da ordem.

Propriedade da ordem (int)

> **ENUM_ORDER_INT**
> Enumerador referente a propriedade da ordem.

VARIÁVEIS	RETORNOS
TYPE_TICKET	Bilhete da ordem.
TYPE_TIME_SETUP	Hora da configuração da ordem.
TYPE_TYPE	Tipo da ordem. (0: Compra mercado, 1: Venda mercado, 2: Buy Limit, 3: Sell Limit, 4: Buy Stop, 5: Sell Stop, 6: Buy Stop Limit, 7: Sell Stop Limit, 8: Fechar por oposta)
TYPE_STATE	Estado da ordem. (0: Verificando, 1: Aceita, 2: Cancelada, 3: Parcial, 4: Executada, 5: Rejeitada, 6: Expirada, 7: Registrando, 8: Modificando, 9: Cancelando)
TYPE_TIME_EXPIRATION	Tempo de expiração da ordem.
TYPE_TIME_DONE	Tempo de conclusão da ordem.
TYPE_TIME_SETUP_MSC	Tempo de configuração da ordem em milissegundos desde 01.01.1970.
TYPE_TIME_DONE_MSC	Tempo de conclusão da ordem em milissegundos desde 01.01.1970.
TYPE_FILLING	Tipo de preenchimento da ordem. (0: FOK (tudo ou nada), 1: IOC (tudo/parcial), 2: BOC (só livro), 3: Return (parcial continua))
TYPE_TIME	Hora da ordem.
TYPE_MAGIC	Número mágico da ordem.

VARIÁVEIS	RETORNOS
`TYPE_REASON`	Razão da ordem. (0: Desktop, 1: Mobile, 2: Web, 3: Expert, 4: Stop Loss, 5: Take Profit, 6: Stop Out)
`TYPE_POSITION_ID`	ID da posição da ordem.
`TYPE_POSITION_BY_ID`	Posição da ordem por ID.

Propriedade da ordem (string)

> **ENUM_ORDER_STRING**
> Enumerador referente a propriedade da ordem.

VARIÁVEIS	RETORNOS
`TYPE_SYMBOL`	Simbolo da ordem.
`TYPE_COMMENT`	Comentario da ordem.

Estilo de linha

> **ENUM_LINE_STYLE**
> Enumerador referente ao estilo de linha de um objeto

VARIÁVEIS	RETORNOS
`TYPE_SOLID`	Linha solida.
`TYPE_DASH`	Linha tracejada.
`TYPE_DOT`	Linha pontilhada.

VARIÁVEIS	RETORNOS
TYPE_DASHDOT	Linha tracejada pontilhada.
TYPE_DASHDOTDOT	Linha tracejada pontilhada pontilhada.

Eventos

Nesta página, você encontrará os eventos disponíveis no ScriptBot. Eventos são ocorrências que podem ser monitoradas e personalizadas conforme o comportamento desejado.

> **Nota**
>
> Antes de prosseguir, é importante entender o que é um evento e como utilizar.
>
> Para entender, leia **O que é um evento**.

Eventos normais

> Eventos normais são ocorrências comuns que acontecem durante a execução do robô, como o início, parada ou passagem do tempo.
> Estes eventos não possuem parametros.

EVENTOS	DESCRIÇÃO
`OnStart()`	Ocorre quando o robô inicia.
`OnRestart()`	Ocorre quando o robô reinicia.
`OnStop()`	Ocorre quando o robô para.
`OnTick()`	Ocorre em cada tick.
`OnLateTick()`	Ocorre em cada final de tick.
`OnOptimizedTick()`	Ocorre em cada tick otimizado.
`OnTickSecond()`	Ocorre em cada segundo de tick.

EVENTOS	DESCRIÇÃO
`OnCandle()`	Ocorre em cada candle.
`OnTimer()`	Ocorre em cada milisegundo.
`OnLateTimer()`	Ocorre em cada final do milisegundo.
`OnSecond()`	Ocorre em cada segundo.
`OnDay()`	Ocorre em cada dia.
`OnWeek()`	Ocorre em cada semana.
`OnMonth()`	Ocorre em cada mês.
`OnYear()`	Ocorre em cada ano.
`OnOnline()`	Ocorre quando o robô volta online.
`OnOffline()`	Ocorre quando o robô volta offline.

Eventos de gatilhos

> Eventos de gatilho são ocorrências que acontecem quando uma condição ou expressão lógica é avaliada nos cálculos de REQUISITO, ENTRADA, CONFIRMAÇÃO, SAÍDA...
> Variáveis de manipulação: Gatilhos

EVENTOS	DESCRIÇÃO
`OnRequisite()`	Ocorre quando uma condição de requisito é atendida com sucesso.

EVENTOS	DESCRIÇÃO
`OnRequisiteFailed()`	Ocorre quando uma condição de requisito é atendida sem sucesso.
`OnEntry()`	Ocorre quando uma condição de entrada é atendida com sucesso.
`OnEntryFailed()`	Ocorre quando uma condição de entrada é atendida sem sucesso.
`OnConfirm()`	Ocorre quando uma condição de confirmação é atendida com sucesso.
`OnConfirmFailed()`	Ocorre quando uma condição de confirmação é atendida sem sucesso.
`OnPositionExit()`	Ocorre quando uma condição de saída de posições é atendida com sucesso.
`OnPositionExitFailed()`	Ocorre quando uma condição de saída de posições é atendida sem sucesso.
`OnPositionExitIndividual()`	Ocorre quando uma condição de saída de posições individual é atendida com sucesso.
`OnPositionExitIndividualFailed()`	Ocorre quando uma condição de saída de posições individual é atendida sem sucesso.
`OnOrderExit()`	Ocorre quando uma condição de saída de ordens é atendida com sucesso.
`OnOrderExitFaile`	Ocorre quando uma condição de saída de ordens é atendida sem sucesso.

EVENTOS	DESCRIÇÃO
`d()`	
`OnOrderExitIndividual()`	Ocorre quando uma condição de saída de ordens individual é atendida com sucesso.
`OnOrderExitIndividualFailed()`	Ocorre quando uma condição de saída de ordens individual é atendida sem sucesso.
`OnFilter()`	Ocorre quando uma condição de filtro é atendida com sucesso.
`OnFilterFailed()`	Ocorre quando uma condição de filtro é atendida sem sucesso.

Eventos de transações

> Eventos de ordens são ocorrências de transações de operações realizadas pelo robô, estes eventos são chamados sempre que uma ordem ou posição for criada, removida ou modificada.
> Variáveis de manipulação: Posições e Ordens

EVENTOS	DESCRIÇÃO
`OnInactive()`	Ocorre quando um pedido fica inativo por um certo tempo.
`OnValidate()`	Ocorre quando um pedido foi validado e criado no mercado.
`OnPositionAdd()`	Ocorre quando uma posição foi criada.
`OnPositionNotAdd()`	Ocorre quando uma posição não foi criada.

EVENTOS	DESCRIÇÃO
`OnPositionClose()`	Ocorre quando uma posição foi fechada.
`OnPositionModify()`	Ocorre quando uma posição foi modificada.
`OnOrderAdd()`	Ocorre quando uma ordem foi criada.
`OnOrderNotAdd()`	Ocorre quando uma ordem não foi criada.
`OnOrderCancel()`	Ocorre quando uma ordem foi cancelada.
`OnOrderTrigger()`	Ocorre quando uma ordem foi acionada.
`OnOrderModify()`	Ocorre quando uma ordem foi modificada.

Lista de indicadores

Nesta página, você encontrará a lista de indicadores disponíveis no ScriptBot. Ela inclui indicadores internos fornecidos pela própria plataforma MetaTrader 5.

Accelerator Oscillator

Nome:	`AC`
Instância:	`NEW, SYMBOL, CURRENT`

> **Imagens**

Accumulation/Distribution

Nome:	`AD`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`TICK`

> **Imagens**

Average Directional Index

Nome:	`ADX`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14`

> **Imagens**

ADX by Welles Wilder

Nome:	`ADXW`

Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14`

> **Imagens**

Alligator

Nome:	`ALLIGATOR`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`13, 8, 8, 5, 5, 3, SMOOTHED, MEDIAN`

> **Imagens**

Adaptive Moving Average

Nome:	`AMA`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`10, 2, 30, 0, CLOSE`

> **Imagens**

Awesome Oscillator

Nome:	`AO`
Instância:	`NEW, SYMBOL, CURRENT`

> **Imagens**

Average True Range

Nome:	`ATR`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14`

> **Imagens**

Bollinger Bands

Nome:	`BANDS`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`20, 0, 2.0, CLOSE`

> **Imagens**

Bears Power

Nome:	`BEARS`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`13`

> **Imagens**

Bulls Power

Nome:	`BULLS`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`13`

> **Imagens**

Market Facilitation Index

Nome:	`BWMFI`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`TICK`

> **Imagens**

Commodity Channel Index

Nome:	`CCI`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14, TYPICAL`

> **Imagens**

Chaikin Oscillator

Nome:	`CHAIKIN`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`3, 10, EXPONENTIAL, TICK`

> **Imagens**

Double Exponential Moving Average

Nome:	`DEMA`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`14, 0, CLOSE`

> **Imagens**

DeMarker

Nome:	`DEMARKER`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14`

> **Imagens**

Envelopes

Nome:	`ENVELOPES`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`14, 0, EXPONENTIAL, CLOSE, 0.1`

> **Imagens**

Force Index

Nome:	`FORCE`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`13, SIMPLE, TICK`

> **Imagens**

Fractals

Nome:	`FRACTALS`
Instância:	`0, SYMBOL, CURRENT`

> **Imagens**

Fractal Adaptive Moving Average

Nome:	`FRAMA`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`30, 0, CLOSE`

> **Imagens**

Gator Oscillator

Nome:	`GATOR`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`13, 8, 8, 5, 5, 3, SMOOTHED, MEDIAN`

> **Imagens**

Ichimoku Kinko Hyo

Nome:	`ICHIMOKU`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`9, 26, 52`

> **Imagens**

Moving Average

Nome:	`MA`
Instância:	`0, SYMBOL, CURRENT`

Parâmetros:	`10, 0, SIMPLE, CLOSE`

> **Imagens**

MACD

Nome:	`MACD`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`12, 26, 9, CLOSE`

> **Imagens**

Money Flow Index

Nome:	`MFI`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14, TICK`

> **Imagens**

Momentum

Nome:	`MOMENTUM`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14, CLOSE`

> **Imagens**

On Balance Volume

Nome:	`OBV`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`TICK`

> **Imagens**

OsMA

Nome:	`OSMA`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`12, 26, 9, CLOSE`

> **Imagens**

Relative Strength Index

Nome:	`RSI`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14, CLOSE`

> **Imagens**

Relative Vigor Index

Nome:	`RVI`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`10`

> **Imagens**

Parabolic SAR

Nome:	`SAR`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`0.02, 0.2`

> **Imagens**

Standard Deviation

Nome:	`STDDEV`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`20, 0, SIMPLE, CLOSE`

> **Imagens**

Stochastic Oscillator

Nome:	`STOCHASTIC`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`5, 3, 3, SIMPLE, LOWHIGH`

> **Imagens**

Triple Exponential Moving Average

Nome:	`TEMA`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`14, 0, CLOSE`

> **Imagens**

Triple Exponential Moving Averages Oscillator

Nome:	`TRIX`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`14, CLOSE`

> **Imagens**

Variable Index Dynamic Average

Nome:	`VIDYA`
Instância:	`0, SYMBOL, CURRENT`
Parâmetros:	`15, 12, 0, CLOSE`

> **Imagens**

Volumes

Nome:	`VOLUMES`
Instância:	`NEW, SYMBOL, CURRENT`
Parâmetros:	`TICK`

> **Imagens**

Williams’ Percent Range

Nome:	`WPR`
Instância:	`NEW, SYMBOL, CURRENT`

Parâmetros:	1

> **Imagens**

Operadores

Nesta página, você encontrará os operadores disponíveis no ScriptBot. Operadores são símbolos utilizados para realizar operações matemáticas, lógicas ou estruturais dentro de uma expressão, sendo fundamentais na criação de condições, cálculos e controle de fluxo em scripts.

Tipos de operadores

> Tabela de operadores.

OPERADORES	DESCRIÇÕES
+	Adicionar.
-	Subtrair.
*	Multiplicar.
/	Dividir.
!	Negação.
%	Resto da divisão.
>	Maior que.
<	Menor que.
>=	Maior ou igual a.
<=	Menor ou igual a.
==	Igual a.
!=	Diferente de.
&&	E.
|	Ou.
( )	Priorizar.

OPERADORES	DESCRIÇÕES
`? :`	Condição IF ‘?’ e ELSE ‘:’.
`->`	Executador lógico.
`//`	Comentário de linha.
`/* */`	Comentário de bloco.

Lista de Operadores

Operator +

> Operador de adição.
> Retorna a adição entre dois números.

```
CLOSE[0] + CLOSE[1]
```

Operator -

> Operador de subtração.
> Retorna a subtração entre dois números.

```
CLOSE[0] - CLOSE[1]
```

Operator *

> Operador de multiplicação.
> Retorna a multiplicação entre dois números.

```
CLOSE[0] * 2
```

Operator /

> Operador de divisão.
> Retorna a divisão entre dois números.

```
CLOSE[0] / 2
```

Operator !

> Operador de negação.
> Usando o operador de negação, podemos inverter o resultado de uma expressão.
> Exemplo:
> `!(CLOSE[0] > CLOSE[1])` quando `CLOSE[0]` for maior que `CLOSE[1]`, o resultado será `true`, mas como estamos negando essa expressão, o resultado será `false`.

```
!(CLOSE[0] > CLOSE[1])
```

Operator %

> Operador de resto da divisão.
> Retorna o resto da divisão entre dois números.

```
11 % 2 // Retorna 1, pois é o resto da divisão de 11 por 2 (11 / 2 = 5 com resto 1)
```

Operator >

> Operador de maior que.
> Retorna verdadeiro se o valor à esquerda for maior ao valor à direita.

```
CLOSE[0] > CLOSE[1]
```

Operator <

> Operador de menor que.
> Retorna verdadeiro se o valor à esquerda for menor ao valor à direita.

```
CLOSE[0] < CLOSE[1]
```

Operator >=

> Operador de maior ou igual a.
> Retorna verdadeiro se o valor à esquerda for maior ou igual ao valor à direita.

```
CLOSE[0] >= CLOSE[1]
```

Operator <=

> Operador de menor ou igual a.
> Retorna verdadeiro se o valor à esquerda for menor ou igual ao valor à direita.

```
CLOSE[0] <= CLOSE[1]
```

Operator ==

> Operador de igualdade.
> Retorna verdadeiro se os valores comparados são iguais.

```
CLOSE[0] == CLOSE[1]
```

Operator !=

> Operador de desigualdade.
> Retorna verdadeiro se os valores comparados não forem iguais.

```
CLOSE[0] != CLOSE[1]
```

Operator &&

> Operador de e.
> Retorna verdadeiro apenas se ambas as condições forem verdadeiras.

```
CLOSE[0] && CLOSE[1]
```

Operator |

> Operador de ou.
> Retorna verdadeiro se pelo menos uma das condições for verdadeira.

```
CLOSE[0] >= CLOSE[1] || OPEN[0] >= OPEN[1]
```

Operator ( )

> Operador de priorização.
> Utilizado para definir a ordem de avaliação das expressões, garantindo que os cálculos entre parênteses sejam executados antes dos demais operadores.

```
(CLOSE[0] + CLOSE[1]) / 2
```

Operator ?:

> Operador de condição.
> Avalia uma condição e executa uma das duas instruções com base no resultado: a primeira se for verdadeira, a segunda se for falsa.

```
CLOSE[0] > CLOSE[1] ? Print["Biggest"] : Print["Smallest"]
```

Operator ->

> Operador de execução lógica.
> Utilizado para executar comandos apenas quando a condição anterior for verdadeira.
>
> **Atenção:**
> É obrigatório encerrar a instrução com `;`, pois esse caractere delimita o fim da expressão.

```
CLOSE[0] > CLOSE[1] -> Print["Este comando só será executado se a condição for verdadeira"];
```

Operator //

> Operador de comentário de linha.
> Usando o operador de comentário de linha, podemos evitar que uma expressão a partir do comentario seja executada.
> No exemplo abaixo, `CLOSE[0] > CLOSE[1]` é executado, mas `OPEN[0] < OPEN[1]` nao, pois ele foi comentado.

```
CLOSE[0] > CLOSE[1] // && OPEN[0] < OPEN[1]
```

Operator /* */

> Operador de comentário de bloco.
> Usando o operador de comentário de bloco, podemos evitar que uma expressão entre o comentario seja executada.
> No exemplo abaixo, `CLOSE[0] > CLOSE[1]` nao é executado, pois ele foi comentado.

```
/*CLOSE[0] > CLOSE[1]*/ OPEN[0] < OPEN[1]
```

Tipos de dados

Nesta página, você encontrará os tipos de dados primitivos disponíveis no ScriptBot. Esses tipos são utilizados como parâmetros e valores de retorno em funções, permitindo o controle preciso sobre o tipo de informação manipulada no projeto. Cada tipo possui uma finalidade específica, abrangendo desde números inteiros e caracteres até valores booleanos e horários.

Tipos de dados

> Esses tipos primitivos são usadados como retornos e parâmetros de funções.

TIPOS	RETORNOS
`void`	Representa um valor vazio, sem retorno.
`any`	Representa qualquer valor. Seja um int, double, string, etc.
`int`	Representa um inteiro com sinal (-2147483648 até 2147483647).
`uint`	Representa um inteiro positivo (0 até 4294967295).
`datetime`	Representa um horário em segundos.
`long`	Representa um inteiro com sinal de 64 bits (-9223372036854775808 até 9223372036854775807).
`ulong`	Representa um inteiro sem sinal de 64 bits (0 até 18446744073709551615).
`char`	Representa um caractere Unicode.
`uchar`	Representa um caractere sem sinal.
`short`	Representa um inteiro com sinal de 16 bits (-32768 até 32767).
`ushort`	Representa um inteiro sem sinal de 16 bits (0 até 65535).
`float`	Representa um número de ponto flutuante de precisão simples (7 dígitos).

TIPOS	RETORNOS
`double`	Representa um número de ponto flutuante de precisão dupla (15-16 dígitos).
`string`	Representa uma sequência de caracteres Unicode.
`bool`	Representa um valor lógico (true ou false).
`color`	Representa um valor de cor clrRed, clrGreen, clrBlue...

Funções

Funções são identificadores que armazenam instruções dinâmicas utilizadas em operações e expressões lógicas. Diferente das variáveis, as funções aceitam parâmetros, o que permite modificar seu comportamento conforme os valores fornecidos. Elas oferecem maior flexibilidade na criação de regras e estratégias personalizadas.

Depuradores

> Essas funções permitem executar comandos de depuração.

FUNÇÕES	RETORNOS
`Log`	Imprime um texto nas Logs do ScriptBot.
`Print`	Imprime um texto nas logs do Metatrader.

Indicadores

> Essas funções permitem acessar os valores dos buffers de um indicador, como preços, sinais, cores.

FUNÇÕES	ALIASES	RETORNOS
`IND Indicator`	`IND`	Retorna o valor do buffer de um indicador.
`IND'X' Indicator'X'`	`IND'X'`	Retorna o valor do buffer de um indicador.

Candles

> Essas funções permitem acessar os valores das velas, como preços de abertura, fechamento etc.

FUNÇÕES	ALIASES	RETORNOS
`O` `Open`	`O`	Retorna o preço de abertura de uma vela.
`C` `Close`	`C`	Retorna o preço de fechamento de uma vela.
`H` `High`	`H`	Retorna o preço da máxima de uma vela.
`L` `Low`	`L`	Retorna o preço da mínima de uma vela.
`T` `Time`	`T`	Retorna o horário de abertura de uma vela.
`DIR` `Direction`	`DIR`	Retorna a direção de uma vela (-1 = Negativa, 0 = Neutra, 1 = Positiva).
`SPD` `Spread`	`SPD`	Retorna a quantidade de spread de uma vela.
`Candle`		Retorna o index da vela.
`MaxH` `MaxHigh`	`MaxH`	Retorna o preço máximo entre duas posições de velas.
`MinL` `MinLow`	`MinL`	Retorna o preço mínimo entre duas posições de velas.

FUNÇÕES	ALIASES	RETORNOS
`TICKV` `TickVolume`	`TICKV`	Retorna o volume de uma vela. 
`REALV` `RealVolume`	`REALV`	Retorna o volume de uma vela usando os dados reais. 

Mercado

> Essas funções fornecem dados de mercado semelhantes aos preços das velas, mas com escopo mais abrangente para representar diferentes aspectos do mercado.

FUNÇÕES	ALIASES	RETORNOS
`Last`		Retorna o último preço de negociação. 
`LastH` `LastHigh`	`LastH`	Retorna o maior preço do último negócio. 
`LastL` `LastLow`	`LastL`	Retorna o menor preço do último negócio. 
`Ask`		Retorna o preço de venda (ask). 
`AskH` `AskHigh`	`AskH`	Retorna o maior preço de venda (ask). 
`AskL` `AskLow`	`AskL`	Retorna o menor preço de venda (ask). 

FUNÇÕES	ALIASES	RETORNOS
`Bid`		Retorna o preço de compra (bid).
`BidH` `BidHigh`	`BidH`	Retorna o maior preço de compra (bid).
`BidL` `BidLow`	`BidL`	Retorna o menor preço de compra (bid).
`LastT` `LastTime`	`LastT`	Retorna o horário do último negócio.
`DayST` `DayStartTime`	`DayST`	Retorna o horário de início do pregão no dia especificado.
`DayET` `DayEndTime`	`DayET`	Retorna o horário de encerramento do pregão no dia especificado.
`DaySC` `DayStartCandle`	`DaySC`	Retorna a vela (candle) de início do dia.
`DayEC` `DayEndCandle`	`DayEC`	Retorna a vela (candle) de fim do dia.
`DayH` `DayHigh`	`DayH`	Retorna o maior preço do dia.
`DayL` `DayLow`	`DayL`	Retorna o menor preço do dia.

Matemática

> Estas são funções matemáticas utilizadas para realizar operações numéricas, como comparação de valores, cálculo de casas decimais e geração de resultados baseados em probabilidade.

FUNÇÕES	ALIASES	RETORNOS
`Med` `Median`	`Med`	Retorna a media entre dois valores. 
`TickTP` `TickToPoint`	`TickTP`	Retorna o valor de tick convertido em pontos. 
`MoneyTP` `MoneyToPoint`	`MoneyTP`	Retorna o valor na moeda convertida em pontos. 
`PipTP` `PipToPoint`	`PipTP`	Retorna o valor do pip convertido em pontos. 
`Count` `CountStep`	`Count`	Retorna a quantidade de passos que ocorreram. 
`CountGrad` `CountStepGradual`	`CountGrad`	Retorna a quantidade de passos que ocorreram de forma gradual. 
`Divi` `Division`	`Divi`	Retorna a divisão entre dois valores, permitindo que o denominador seja zero. 
`RDivi` `RestDivision`	`RDivi`	Retorna o resto da divisão entre dois valores, permitindo que o denominador seja zero. 
`Percent`		Retorna a variação percentual de value até target, com value sendo o valor base.

FUNÇÕES	ALIASES	RETORNOS
		📖
`PercentValue` `PercentOfValue`	`PercentValue`	Retorna a quantidade percentual de um valor. 📖
`HitPercent`		Retorna a quantidade percentual de acerto. 📖
`FixPrice` `CorrectPrice`	`FixPrice`	Retorna o valor correto de um preço. 📖
`Random`		Retorna um valor inteiro aleatório entre dois valores. 📖
`Round`		Retorna o valor arredondado. 📖
`RoundUp`		Retorna o valor arredondado para cima. 📖
`RoundDown`		Retorna o valor arredondado para baixo. 📖
`Max`		Retorna o maior valor entre dois números. 📖
`Min`		Retorna o menor valor entre dois números. 📖
`Chance` `IsChance`	`Chance`	Retorna verdadeiro com base em uma chance percentual. 📖
`Decimal`	`Decimal`	Retorna o número de casas decimais de um valor.

FUNÇÕES	ALIASES	RETORNOS
`Decimals`		📖

Conta

> Estas funções estão relacionadas aos dados da conta e permitem tanto recuperar quanto atualizar essas informações.

FUNÇÕES	ALIASES	RETORNOS
`GetVol` `GetVolume`	`GetVol`	Retorna o volume que a ordem será executada. 📖
`SetVol` `SetVolume`	`SetVol`	Define o novo volume da ordem. 📖

Tempo

> Estas funções auxiliam em operações relacionadas a datas e horários, como conversões, cálculos, formatações e retornos.

FUNÇÕES	ALIASES	RETORNOS
`IsTime`		Verifica se o tempo corresponde aos critérios informados. 📖
`STime` `SymbolTime`	`STime`	Retorna o horário atual do símbolo. 📖
`TLeft` `TimeLeft`	`TLeft`	Retorna o tempo restante da vela atual. 📖
`DayW`	`DayW`	Retorna o dia da semana.

FUNÇÕES	ALIASES	RETORNOS
`DayOfWeek`		🕮
`DayY` `DayOfYear`	`DayY`	Retorna o dia do ano. 🕮
`Month`		Retorna o mês atual. 🕮
`Year`		Retorna o ano atual. 🕮
`StartT` `StartTime`	`StartT`	Retorna o horário inicial de referência. 🕮
`ValueT` `ValueOfTime`	`ValueT`	Converte o horário em valor numérico baseado no método. 🕮

Posições e Ordens abertas

> Estas funções permitem obter informações sobre posições e ordens abertas, como quantidades, preços, volumes, etc.

FUNÇÕES	ALIASES	RETORNOS
`AVolO` `AllVolumeOpen`	`AVolO`	Retorna o volume total de todas as posições e ordens abertas. 🕮
`BVolO` `BuyVolumeOpen`	`BVolO`	Retorna o volume total das posições e ordens de compra abertas. 🕮
`SVolO`	`SVolO`	Retorna o volume total das posições e ordens de venda abertas.

FUNÇÕES	ALIASES	RETORNOS
`SellVolumeOpen`		📖
`APVolO` `AllPosVolumeOpen`	`APVolO`	Retorna o volume total de todas as posições abertas. 📖
`BPVolO` `BuyPosVolumeOpen`	`BPVolO`	Retorna o volume total das posições de compra abertas. 📖
`SPVolO` `SellPosVolumeOpen`	`SPVolO`	Retorna o volume total das posições de venda abertas. 📖
`AOVolO` `AllOrderVolumeOpen`	`AOVolO`	Retorna o volume total de todas as ordens pendentes. 📖
`BOVolO` `BuyOrderVolumeOpen`	`BOVolO`	Retorna o volume total das ordens de compra pendentes. 📖
`SOVolO` `SellOrderVolumeOpen`	`SOVolO`	Retorna o volume total das ordens de venda pendentes. 📖
`AOpen` `AllOpen`	`AOpen`	Retorna a quantidade total de posições e ordens abertas. 📖
`BOpen` `BuyOpen`	`BOpen`	Retorna a quantidade total de posições e ordens de compra abertas. 📖

FUNÇÕES	ALIASES	RETORNOS
`SOpen` `SellOpen`	`SOpen`	Retorna a quantidade total de posições e ordens de venda abertas.
`APosO` `AllPosOpen`	`APosO`	Retorna a quantidade total de posições abertas.
`BPosO` `BuyPosOpen`	`BPosO`	Retorna a quantidade total de posições de compra abertas.
`SPosO` `SellPosOpen`	`SPosO`	Retorna a quantidade total de posições de venda abertas.
`AOrderO` `AllOrderOpen`	`AOrderO`	Retorna a quantidade total de ordens pendentes.
`BOrderO` `BuyOrderOpen`	`BOrderO`	Retorna a quantidade total de ordens de compra pendentes.
`SOrderO` `SellOrderOpen`	`SOrderO`	Retorna a quantidade total de ordens de venda pendentes.
`AProfitO` `AllProfitOpen`	`AProfitO`	Retorna o lucro/prejuízo total de todas as posições abertas.
`BProfitO` `BuyProfitOpen`	`BProfitO`	Retorna o lucro/prejuízo total das posições de compra abertas.
`SProfitO` `SellProfitOpen`	`SProfitO`	Retorna o lucro/prejuízo total das posições de venda abertas.

FUNÇÕES	ALIASES	RETORNOS
`PAvgP` `AvgPrice`	`PAvgP`	Retorna o preço médio das posições abertas. 
`PAvgDir` `AvgDir`	`PAvgDir`	Retorna a direção média das posições abertas. 

`AAvgPric`
`e`
`AllAvgPr`
`ice`
	`AAvgPri`
	`ce`
		Retorna o preço médio de todas as posições abertas.
		 

`BAvgPric`
`e`
`BuyAvgPr`
`ice`
	`BAvgPri`
	`ce`
		Retorna o preço médio das posições de compra abertas.
		 

`SAvgPric`
`e`
`SellAvgP`
`rice`
	`SAvgPri`
	`ce`
		Retorna o preço médio das posições de venda abertas.
		 

`AAvgDir`
`AllAvgDi`
`r`
	`AAvgDir`	Retorna a direção média de todas as posições abertas.
		 

`BAvgDir`
`BuyAvgDi`
`r`
	`BAvgDir`	Retorna a direção média das posições de compra abertas.
		 

`SAvgDir`
`SellAvgD`
`ir`
	`SAvgDir`	Retorna a direção média das posições de venda abertas.
		 

`PTicket`
`PosTicke`
`t`
	`PTicket`	Retorna o ticket de uma posição.
		 

FUNÇÕES	ALIASES	RETORNOS
`PosDouble`		Retorna o valor de uma propriedade numérica (double) de uma posição.
`PosInt`		Retorna o valor de uma propriedade inteira de uma posição.
`PosString`		Retorna o valor de uma propriedade textual de uma posição.
`PCurrent` `PosCurrent`	`PCurrent`	Retorna o preço atual de uma posição aberta.
`PO` `PosOpen`	`PO`	Retorna o preço de abertura de uma posição.
`PProfit` `PosProfit`	`PProfit`	Retorna o lucro atual de uma posição.
`PSL` `PosStop`	`PSL`	Retorna o preço do Stop Loss de uma posição.
`PSWAP` `PosSwap`	`PSWAP`	Retorna o valor do swap acumulado de uma posição.
`PTP` `PosTake`	`PTP`	Retorna o preço do Take Profit de uma posição.
`PVol` `PosVolume`	`PVol`	Retorna o volume de uma posição.
`PIdent` `PosIdentifier`	`PIdent`	Retorna o identificador único de uma posição.

FUNÇÕES	ALIASES	RETORNOS
`PMagic` `PosMagic`	`PMagic`	Retorna o número mágico de uma posição. 
`PReason` `PosReason`	`PReason`	Retorna o motivo de abertura de uma posição. 
`PT` `PosTime`	`PT`	Retorna o horário de abertura de uma posição. 
`PTMsc` `PosTimeMsc`	`PTMsc`	Retorna o horário de abertura de uma posição em milissegundos. 
`PTUpdate` `PosTimeUpdate`	`PTUpdate`	Retorna o horário da última atualização de uma posição. 
`PTUpdateMsc` `PosTimeUpdateMsc`	`PTUpdateMsc`	Retorna o horário da última atualização de uma posição em milissegundos. 
`PType` `PosType`	`PType`	Retorna o tipo de uma posição (compra/venda). 
`PSymbol` `PosSymbol`	`PSymbol`	Retorna o símbolo do ativo de uma posição. 
`PComment` `PosComment`	`PComment`	Retorna o comentário associado a uma posição. 
`PExternal`	`PExternal`	Retorna o ID externo de uma posição. 

FUNÇÕES	ALIASES	RETORNOS
`PosExternal`		
`OTicket` `OrderTicket`	`OTicket`	Retorna o ticket de uma ordem.
`OrderDouble`		Retorna o valor de uma propriedade numérica (double) de uma ordem.
`OrderInt`		Retorna o valor de uma propriedade inteira de uma ordem.
`OrderString`		Retorna o valor de uma propriedade textual de uma ordem.
`OVolInit` `OrderVolumeInit`	`OVolInit`	Retorna o volume inicial de uma ordem pendente.
`OVol` `OrderVolume`	`OVol`	Retorna o volume atual de uma ordem.
`OO` `OrderOpen`	`OO`	Retorna o preço de abertura de uma ordem.
`OSL` `OrderStop`	`OSL`	Retorna o preço do Stop Loss de uma ordem.
`OTP` `OrderTake`	`OTP`	Retorna o preço do Take Profit de uma ordem.

FUNÇÕES	ALIASES	RETORNOS
`OCurrentOrderCurrent`	`OCurrent`	Retorna o preço atual de uma ordem pendente.
`OSLLimitOrderStopLimit`	`OSLLimit`	Retorna o preço de ativação para ordens Stop Limit.
`OSetupOrderSetup`	`OSetup`	Retorna a data/hora de criação da ordem.
`OTypeOrderType`	`OType`	Retorna o tipo de uma ordem.
`OStateOrderState`	`OState`	Retorna o estado atual de uma ordem.
`OTExpiraOrderTimeExpiration`	`OTExpira`	Retorna a data/hora de expiração de uma ordem.
`ODoneOrderDone`	`ODone`	Retorna a data/hora de execução ou cancelamento de uma ordem.
`OSetupMscOrderSetupMsc`	`OSetupMsc`	Retorna a data/hora de criação da ordem em milissegundos.
`ODoneMscOrderDoneMsc`	`ODoneMsc`	Retorna a data/hora de execução ou cancelamento em milissegundos.

FUNÇÕES	ALIASES	RETORNOS
`OFilling` `OrderFilling`	`OFilling`	Retorna a política de preenchimento de uma ordem. 📖
`OT` `OrderTime`	`OT`	Retorna a data/hora da última modificação da ordem. 📖
`OMagic` `OrderMagic`	`OMagic`	Retorna o número mágico de uma ordem. 📖
`OReason` `OrderReason`	`OReason`	Retorna o motivo de criação da ordem. 📖
`OId` `OrderId`	`OId`	Retorna o ID da ordem no sistema. 📖
`OById` `OrderById`	`OById`	Retorna o ticket da ordem pelo ID do sistema. 📖
`OSymbol` `OrderSymbol`	`OSymbol`	Retorna o símbolo do ativo de uma ordem. 📖
`OComment` `OrderComment`	`OComment`	Retorna o comentário associado a uma ordem. 📖

Posições fechadas

> Estas funções permitem obter informações sobre posições fechadas, como quantidades, preços, volumes, etc.

FUNÇÕES	ALIASES	RETORNOS
`DTicket` `DealTicket`	`DTicket`	Retorna o ticket de uma posição fechada. 
`DProfit` `DealProfit`	`DProfit`	Retorna o lucro/prejuízo de um negócio. 
`DO` `DealOpen`	`DO`	Retorna o preço de abertura de um negócio. 
`DComm` `DealCommission`	`DComm`	Retorna o valor da comissão de um negócio. 
`DFee` `DealFee`	`DFee`	Retorna o valor da taxa de um negócio. 
`DSL` `DealStop`	`DSL`	Retorna o preço do stop loss de um negócio. 
`DTP` `DealTake`	`DTP`	Retorna o preço do take profit de um negócio. 
`DSwap` `DealSwap`	`DSwap`	Retorna o valor do swap de um negócio. 
`DVol` `DealVolume`	`DVol`	Retorna o volume de um negócio. 
`DMagic` `DealMagic`	`DMagic`	Retorna o número mágico de um negócio. 
`DReason`	`DReason`	Retorna o motivo da execução de um negócio.

FUNÇÕES	ALIASES	RETORNOS
`DealReason`		📖
`DOrder` `DealOrder`	`DOrder`	Retorna o ticket da ordem associada a um negócio. 📖
`DId` `DealId`	`DId`	Retorna o ID de um negócio. 📖
`DEntry` `DealEntry`	`DEntry`	Retorna o tipo de entrada de um negócio. 📖
`DOut` `DealOut`	`DOut`	Retorna se o negócio é uma saída. 📖
`DIn` `DealIn`	`DIn`	Retorna se o negócio é uma posição de entrada. 📖
`DT` `DealTime`	`DT`	Retorna a data/hora de execução de um negócio. 📖
`DTMsc` `DealTimeMsc`	`DTMsc`	Retorna o tempo de execução de negociações em milissegundos desde 01.01.1970 📖
`DType` `DealType`	`DType`	Retorna o tipo de um negócio. 📖
`DComment` `DealComment`	`DComment`	Retorna o comentário associado a um negócio. 📖
`DExternal`	`DExternal`	Retorna o ID externo de um negócio. 📖

FUNÇÕES	ALIASES	RETORNOS
`DealExternal`		
`DSymbol` `DealSymbol`	`DSymbol`	Retorna o símbolo de um negócio.

Conversores

> Estas funções de conversão permitem transformar valores entre diversos formatos, como converter números positivos em negativos e transformar strings em datas, entre outras operações.

FUNÇÕES	ALIASES	RETORNOS
`Pos` `ToPositive`	`Pos`	Retorna o valor absoluto de um número.
`Neg` `ToNegative`	`Neg`	Retorna o valor negativo de um número.
`ToTime`		Retorna o horário em forma de texto simples para data em segundos.
`TMod` `ToTimeModify`	`TMod`	Retorna o horário modificado.
`TFormat` `ToTimeFormat`	`TFormat`	Retorna o horário em forma de texto.
`Format` `ToFormat`	`Format`	Retorna uma nova string formatada.

FUNÇÕES	ALIASES	RETORNOS
`ToInt`		Retorna o valor convertido para inteiro.

Variáveis do usuário

> Essas variáveis permitem que o usuário crie suas próprias variáveis personalizadas.

FUNÇÕES	ALIASES	RETORNOS
`ClrDou` `ClearDouble`	`ClrDou`	Remove todas as variáveis criadas pelo usuário.
`ClrInt` `ClearInt`	`ClrInt`	Remove todas as variáveis criadas pelo usuário.
`ClrStr` `ClearString`	`ClrStr`	Remove todas as variáveis criadas pelo usuário.
`CreDou` `CreateDouble`	`CreDou`	Cria uma variável caso ela não exista.
`CreInt` `CreateInt`	`CreInt`	Cria uma variável caso ela não exista.
`CreStr` `CreateString`	`CreStr`	Cria uma variável caso ela não exista.
`GetDou` `GetDouble`	`GetDou`	Retorna o valor de uma variável criada pelo usuário.

FUNÇÕES	ALIASES	RETORNOS
`GetInt`		Retorna o valor de uma variável criada pelo usuário.
`GetStr` `GetString`	`GetStr`	Retorna o valor de uma variável criada pelo usuário.
`RemDou` `RemoveDouble`	`RemDou`	Remove uma variável caso ela exista.
`RemInt` `RemoveInt`	`RemInt`	Remove uma variável caso ela exista.
`RemStr` `RemoveString`	`RemStr`	Remove uma variável caso ela exista.
`RepDou` `ReplaceDouble`	`RepDou`	Substitui o valor de uma variável caso ela exista.
`RepInt` `ReplaceInt`	`RepInt`	Substitui o valor de uma variável caso ela exista.
`RepStr` `ReplaceString`	`RepStr`	Substitui o valor de uma variável caso ela exista.
`SetDou` `SetDouble`	`SetDou`	Define o valor de uma variável ou cria uma nova.
`SetInt`		Define o valor de uma variável ou cria uma nova.

FUNÇÕES	ALIASES	RETORNOS
`SetStr` `SetString`	`SetStr`	Define o valor de uma variável ou cria uma nova. 
`AddDou` `AddDouble`	`AddDou`	Adiciona um valor double a uma variável existente. 
`AddInt`		Adiciona um valor inteiro a uma variável existente. 

Variáveis globais

> Essas variáveis armazenam valores globais que são compartilhados entre todos os terminais do MetaTrader 5.
> Lembre-se: por serem globais, qualquer programa pode alterar seus valores, o que pode gerar conflitos.
> As variáveis permanecem disponíveis por até quatro semanas; caso não sejam utilizadas nesse período, o sistema as removerá automaticamente.

FUNÇÕES	ALIASES	RETORNOS
`GetG` `GetGlobal`	`GetG`	Obtém o valor de uma variável global do tipo double. 
`CreG` `CreateGlobal`	`CreG`	Cria uma nova variável global do tipo double. 
`RepG` `ReplaceGlobal`	`RepG`	Substitui o valor de uma variável global existente. 
`GetGT`	`GetGT`	Obtém o timestamp do último acesso de uma variável global. 

FUNÇÕES	ALIASES	RETORNOS
`GetGlobalTime`		
`SetG` `SetGlobal`	`SetG`	Define o valor de uma variável global (cria se não existir).
`AddG` `AddGlobal`	`AddG`	Adiciona um valor a uma variável global existente.
`RemG` `RemoveGlobal`	`RemG`	Remove uma variável global do tipo double.
`ClrG` `ClearGlobal`	`ClrG`	Limpa variáveis globais com um prefixo específico.

Trades

> Estas funções permitem abrir, gerenciar e fechar posições de trade, incluindo ordens.

FUNÇÕES	RETORNOS
`BuyAuto`	Abre uma posição de compra no mercado.
`SellAuto`	Abre uma posição de venda no mercado.
`Buy`	Abre uma posição de compra no mercado atual.
`Sell`	Abre uma posição de venda no mercado atual.

FUNÇÕES	RETORNOS
`BuyOrder`	Coloca uma ordem pendente de compra. 
`SellOrder`	Coloca uma ordem pendente de venda. 
`PosClose`	Fecha uma posição aberta completamente. 
`PosClosePartial`	Fecha parcialmente uma posição aberta. 
`OrderClose`	Cancela uma ordem pendente. 
`PosCloseAll`	Fecha todas as posições abertas. 
`PosBuyCloseAll`	Fecha todas as posições de compra abertas. 
`PosSellCloseAll`	Fecha todas as posições de venda abertas. 
`OrderCloseAll`	Fecha todas as ordens abertas. 
`OrderBuyCloseAll`	Fecha todas as ordens de compra abertas. 
`OrderSellCloseAll`	Fecha todas as ordens de venda abertas. 

FUNÇÕES	RETORNOS
`PosModify`	Modifica os níveis de take profit e stop loss de uma posição aberta.
`PosModifyTake`	Modifica apenas o take profit de uma posição aberta.
`PosModifyStop`	Modifica apenas o stop loss de uma posição aberta.
`OrderModify`	Modifica os níveis de take profit e stop loss de uma ordem aberta.
`OrderModifyTake`	Modifica apenas o take profit de uma ordem aberta.
`OrderModifyStop`	Modifica apenas o stop loss de uma ordem aberta.
`OrderModifyPrice`	Modifica apenas o preço de uma ordem aberta.
`Modify`	Modifica os níveis de take profit e stop loss de uma posição ou ordem aberta.
`ModifyTake`	Modifica apenas o take profit de uma posição ou ordem aberta.
`ModifyStop`	Modifica apenas o stop loss de uma posição ou ordem aberta.

Objetos básicos

> Estas funções permitem criar e manipular objetos básicos do MetaTrader 5.

FUNÇÕES	ALIASES	RETORNOS
`ObjHL` `ObjHLine`	`ObjHL`	Cria uma linha horizontal no gráfico. 
`ObjVL` `ObjVLine`	`ObjVL`	Cria uma linha vertical no gráfico. 
`ObjTL` `ObjTrendLine`	`ObjTL`	Cria uma linha de tendência no gráfico. 
`ObjRem` `ObjRemove`	`ObjRem`	Remove um objeto do gráfico. 
`ObjClr` `ObjClear`	`ObjClr`	Remove múltiplos objetos do gráfico. 
`ObjEx` `ObjExist`	`ObjEx`	Verifica se um objeto existe no gráfico. 
`ObjGT` `ObjGetTime`	`ObjGT`	Obtém o tempo associado a um objeto. 
`ObjGP` `ObjGetPrice`	`ObjGP`	Obtém o preço associado a um objeto. 
`ObjST` `ObjSetTime`	`ObjST`	Define o tempo de um ponto de um objeto. 
`ObjSP` `ObjSetPrice`	`ObjSP`	Define o preço de um ponto de um objeto. 

Variáveis

Variáveis são identificadores que armazenam dados dinâmicos utilizados em operações e expressões lógicas. Elas representam valores que podem mudar ao longo do tempo, mas não possuem parâmetros próprios que alterem seu comportamento.

Posições e Ordens:

> Essas variáveis são definidas previamente e utilizadas como parâmetros fixos durante a execução de gatilhos ou eventos de posição.

VARIÁVEIS	RETORNOS
`_TICKET`	Retorna o ticket do pedido.
`_REQUEST`	Retorna o ticket de solicitação do pedido.
`_PARENT`	Retorna o ticket da ordem-mãe, se houver.
`_MAGIC`	Retorna o magic number do pedido.
`_SYMBOL`	Retorna o ativo no qual o pedido foi criado.
`_TIME`	Retorna o horário de abertura do pedido.
`_PRICE`	Retorna o preço de abertura do pedido.
`_SL`	Retorna o preço do stop loss do pedido.
`_TP`	Retorna o preço do take profit do pedido.
`_PROFIT`	Retorna o lucro atual da posição.
`_VOLUME`	Retorna o volume atual do pedido.
`_UPTIME`	Retorna o tempo, em segundos, que o pedido está aberto.
`_TYPE`	Retorna o tipo do pedido (0 = Compra, 1 = Venda).
`_COMMENT`	Retorna o comentário associado ao pedido.

VARIÁVEIS	RETORNOS
`_FLAG_CHILD`	Retorna se o pedido é filho de outra posição.
`_FLAG_CLIENT`	Retorna se o pedido foi adicionado usando expressões lógicas (como Buy(), Sell(), etc.).
`_FLAG_SA`	Retorna se o pedido foi adicionado pelo subsistema de Preço Médio Simples.

> **Dica**
>
> Nos Eventos, como no `OnPositionModify()`, as variáveis locais possuem duas versões: uma com o sufixo **_OLD** e outra sem sufixo.
>
> As variáveis com o sufixo **_OLD** contêm o valor **antes** da modificação, enquanto as sem sufixo refletem o valor **após** a modificação.
>
> Exemplos:
> **_SL_OLD**: Retorna o valor antes da modificação.
> **_SL**: Retorna o valor após a modificação.

Gatilhos:

> Essas variáveis funcionam como parâmetros locais e imutáveis, configurados previamente à ativação de um gatilho.

VARIÁVEIS	RETORNOS
`_TRIG_TYPE`	Retorna o tipo do gatilho. (0 = Compra, 1 = Venda)
`_TRIG_IS_OPERATION`	Retorna se o gatilho é uma operação boleana.
`_TRIG_OPERATION`	Retorna o resultado da operação. (0 = Falso, 1 = Verdadeiro)

VARIÁVEIS	RETORNOS
`_TRIG_PRICE`	Retorna o preço do gatilho, caso ele seja um gatilho de preço.
`_TRIG_WEIGHT_MIN`	Retorna o peso mínimo do filtro.
`_TRIG_WEIGHT_VALIDATE`	Retorna o peso total validado do filtro.
`_TRIG_VALIDS`	Retorna a quantidade de filtros validados.
`_TRIG_VALIDS_COM`	Retorna a quantidade de filtros complementares validados.
`_TRIG_VALIDS_REQ`	Retorna a quantidade de filtros obrigatórios validados.

Conta:

> Essas variáveis contêm dados da conta, incluindo saldo disponível, margens operacionais, limites de operação e informações da corretora.

VARIÁVEIS	ALIASES	RETORNOS
`BAL` `BALANCE`	`BAL`	Retorna o saldo da conta.
`CRED` `CREDIT`	`CRED`	Retorna o crédito da conta.
`EQT` `EQUITY`	`EQT`	Retorna o patrimônio líquido da conta.
`MARGIN`		Retorna a margem da conta.
`MARGF`	`MARGF`	Retorna a margem livre da conta.

VARIÁVEIS	ALIASES	RETORNOS
`MARGIN_FREE`		
`MARGL` `MARGIN_LEVEL`	`MARGL`	Retorna o nível de margem da conta.
`MARGI` `MARGIN_INITIAL`	`MARGI`	Retorna a margem inicial da conta.
`MARGM` `MARGIN_MAINTENANCE`	`MARGM`	Retorna a margem de manutenção da conta.
`LOGIN`		Retorna o número da conta.
`NAME`		Retorna o nome do titular da conta.
`SRV` `SERVER`	`SRV`	Retorna o servidor da conta.
`CURR` `CURRENCY`	`CURR`	Retorna a moeda da conta.
`COMP` `COMPANY`	`COMP`	Retorna a empresa/corretora da conta.
`LIMITO` `LIMIT_ORDER`	`LIMITO`	Retorna o limite máximo de ordens da conta.

Mercado

> Essas variáveis contêm dados do mercado, incluindo preços de entrada, ativos envolvidos e horários de negociação.

VARIÁVEIS	ALIASES	RETORNOS
TCANDLE TOTAL_CANDLE	TCANDLE	Retorna o total de candles.
TLEFT TIME_LEFT	TLEFT	Retorna o tempo restante.
TCURRENT TIME_CURRENT	TCURRENT	Retorna o tempo atual.
TLOCAL TIME_LOCAL	TLOCAL	Retorna o tempo local.
TSERVER TIME_SERVER	TSERVER	Retorna o tempo do servidor.
DAYW DAY_OF_WEEK	DAYW	Retorna o dia da semana.
DAYY DAY_OF_YEAR	DAYY	Retorna o dia do ano.
MONTH		Retorna o mês atual.
YEAR		Retorna o ano atual.
LAST		Retorna o último preço comercializado.
LASTH LAST_HIGH	LASTH	Retorna a máxima do dia.
LASTL LAST_LOW	LASTL	Retorna a mínima do dia.

VARIÁVEIS	ALIASES	RETORNOS
`LASTT` `LAST_TIME`	`LASTT`	Retorna o último tempo comercializado.
`ASK`		Retorna o preço de compra.
`ASKH` `ASK_HIGH`	`ASKH`	Retorna o preço máximo de compra.
`ASKL` `ASK_LOW`	`ASKL`	Retorna o preço mínimo de compra.
`BID`		Retorna o preço de venda.
`BIDH` `BID_HIGH`	`BIDH`	Retorna o preço máximo de venda.
`BIDL` `BID_LOW`	`BIDL`	Retorna o preço mínimo de venda.
`SYMBOL`		Retorna o ativo que o robô está operando.
`MAGIC`		Retorna o id (magic number) que o robô está operando.
`REAL`		Retorna o ativo real, o ativo que as ordens serão executadas.

Posições e Ordens abertas

> Essas variáveis contém os dados de ordens e posições que ainda não foram encerradas, permitindo o acompanhamento em tempo real.

VARIÁVEIS	ALIASES	RETORNOS

`AVOLO`
`ALL_VOLUM`
`E_OPEN`
	`AVOLO`	Retorna a soma do volume total das posições.

VARIÁVEIS	ALIASES	RETORNOS
BVOLO BUY_VOLUME_OPEN	BVOLO	Retorna a soma do volume total das posições de compra.
SVOLO SELL_VOLUME_OPEN	SVOLO	Retorna a soma do volume total das posições de venda.
APOSO ALL_POS_OPEN	APOSO	Retorna a quantidade de posições abertas.
BPOSO BUY_POS_OPEN	BPOSO	Retorna a quantidade de posições abertas de compra.
SPOSO SELL_POS_OPEN	SPOSO	Retorna a quantidade de posições abertas de venda.
AOPEN ALL_OPEN	AOPEN	Retorna a quantidade de pedidos abertos, tanto de posições quanto de ordens.
BOPEN BUY_OPEN	BOPEN	Retorna a quantidade de pedidos abertos de compra.
SOPEN SELL_OPEN	SOPEN	Retorna a quantidade de pedidos abertos de venda.
APROFITO ALL_PROFIT_OPEN	APROFITO	Retorna a soma do lucro das posições abertas.
BPROFITO BUY_PROFIT_OPEN	BPROFITO	Retorna a soma do lucro das posições de compra abertas.

VARIÁVEIS	ALIASES	RETORNOS
SPROFITO SELL_PROFIT_OPEN	SPROFITO	Retorna a soma do lucro das posições de venda abertas.
AORDERO ALL_ORDER_OPEN	AORDERO	Retorna a quantidade de ordens abertas.
BORDERO BUY_ORDER_OPEN	BORDERO	Retorna a quantidade de ordens de compra abertas.
SORDERO SELL_ORDER_OPEN	SORDERO	Retorna a quantidade de ordens de venda abertas.
AAVG ALL_AVG_PRICE	AAVG	Retorna o preço médio de todas as posições abertas.
AAVGDIR ALL_AVG_DIR	AAVGDIR	Retorna a direção média das posições abertas. (-1 = Venda, 0 = Equilíbrio e 1 = Compra).
BAVG BUY_AVG_PRICE	BAVG	Retorna o preço médio das posições de compra abertas.
BAVGDIR BUY_AVG_DIR	BAVGDIR	Retorna a direção média das posições de compra abertas.
SAVG SELL_AVG_PRICE	SAVG	Retorna o preço médio das posições de venda abertas.
SAVGDIR	SAVGDIR	Retorna a direção média das posições de venda abertas.

VARIÁVEIS	ALIASES	RETORNOS
SELL_AVG_DIR		

Posições e Ordens fechadas

> Essas variáveis contém os dados de ordens e posições que foram encerradas.

VARIÁVEIS	ALIASES	RETORNOS
AVOL ALL_VOLUME	AVOL	Retorna a soma total do volume.
BVOL BUY_VOLUME	BVOL	Retorna a soma do volume de compra.
SVOL SELL_VOLUME	SVOL	Retorna a soma do volume de venda.
APROFIT ALL_PROFIT	APROFIT	Retorna o total do lucro.
BPROFIT BUY_PROFIT	BPROFIT	Retorna o total do lucro de compra.
SPROFIT SELL_PROFIT	SPROFIT	Retorna o total do lucro de venda.
AGAINP ALL_GAIN_PROFIT	AGAINP	Retorna o total de profit ganho ignorando as perdas.

VARIÁVEIS	ALIASES	RETORNOS
BGAINP BUY_GAIN_ PROFIT	BGAINP	Retorna o total de profit ganho em compras ignorando as perdas.
SGAINP SELL_GAIN _PROFIT	SGAINP	Retorna o total de profit ganho em vendas ignorando as perdas.
ALOSSP ALL_LOSS_ PROFIT	ALOSSP	Retorna o total de profit perdido ignorando os ganhos.
BLOSSP BUY_LOSS_ PROFIT	BLOSSP	Retorna o total de profit perdido em compras ignorando os ganhos.
SLOSSP SELL_LOSS _PROFIT	SLOSSP	Retorna o total de profit perdido em vendas ignorando os ganhos.
APOS ALL_POS	APOS	Retorna o total de posições.
BPOS BUY_POS	BPOS	Retorna o total de posições de compra.
SPOS SELL_POS	SPOS	Retorna o total de posições de venda.
AGAIN ALL_GAIN	AGAIN	Retorna o total de vitorias.
BGAIN BUY_GAIN	BGAIN	Retorna o total de vitorias de compra.
SGAIN SELL_GAIN	SGAIN	Retorna o total de vitorias de venda.
ALOSS	ALOSS	Retorna o total de derrotas.

VARIÁVEIS	ALIASES	RETORNOS
ALL_LOSS		
BLOSS BUY_LOSS	BLOSS	Retorna o total de derrotas de compra.
SLOSS SELL_LOSS	SLOSS	Retorna o total de derrotas de venda.
ACOMM ALL_COMMISSION	ACOMM	Retorna o total de comissões.
BCOMM BUY_COMMISSION	BCOMM	Retorna o total de comissões de compra.
SCOMM SELL_COMMISSION	SCOMM	Retorna o total de comissões de venda.
ASWAP ALL_SWAP	ASWAP	Retorna o total de trocas.
BSWAP BUY_SWAP	BSWAP	Retorna o total de trocas de compra.
SSWAP SELL_SWAP	SSWAP	Retorna o total de trocas de venda.
TICKET		Retorna o ticket da ultima posição.
TICKETB TICKET_BUY	TICKETB	Retorna o ticket da ultima compra.
TICKETS TICKET_SELL	TICKETS	Retorna o ticket da ultima venda.
GTICKET	GTICKET	Retorna o ticket da ultima posição com ganho.

VARIÁVEIS	ALIASES	RETORNOS
GAIN_TICKET		
GTICKETBGAIN_TICKET_BUY	GTICKETB	Retorna o ticket da ultima compra com ganho.
GTICKETSGAIN_TICKET_SELL	GTICKETS	Retorna o ticket da ultima venda com ganho.
LTICKETLOSS_TICKET	LTICKET	Retorna o ticket da ultima posição com perda.
LTICKETBLOSS_TICKET_BUY	LTICKETB	Retorna o ticket da ultima compra com perda.
LTICKETSLOSS_TICKET_SELL	LTICKETS	Retorna o ticket da ultima venda com perda.

> **Dica**
>
> As variáveis de Ordens e Posições fechadas possuem quatro versões, que indicam diferentes períodos de tempo:
>
> - **_WEEK**: soma total da **semana**
> - **_MONTH**: soma total do **mês**
> - **_YEAR**: soma total do **ano**
> - (sem sufixo): soma total do **dia**
>
> Para utilizá-las, basta adicionar o sufixo correspondente ao final da variável.
> Exemplos: `ALL_VOLUME_WEEK`, `ALL_GAIN_YEAR`.
>
> No caso dos **aliases**, os sufixos são aplicados em formato reduzido:
> Exemplos: `AVOLW` (ALL_VOLUME_WEEK), `AGAINY` (ALL_GAIN_YEAR)
> Sufixos disponíveis: `W` (semana), `M` (mês), `Y` (ano).

Sinais

> Essas variáveis guardam as informações dos sinais, que são definidos com base nos gatilhos de entrada e confirmação.

VARIÁVEIS	ALIASES	RETORNOS
`SIGTT` `SIG_TOTAL`	`SIGTT`	Retorna a soma de todos os sinais do dia.
`SIGB` `SIG_BUY`	`SIGB`	Retorna a soma de sinais de compra.
`SIGS` `SIG_SELL`	`SIGS`	Retorna a soma de sinais de venda.
`SIGL` `SIG_LAST`	`SIGL`	Retorna o tipo do último sinal. (-1 = Nenhum, 0 = Compra, 1 = Venda)
`SIGT` `SIG_TIME`	`SIGT`	Retorna a hora do último sinal.

VARIÁVEIS	ALIASES	RETORNOS
SIGTB SIG_TIME_ BUY	SIGTB	Retorna a hora do último sinal de compra.
SIGTS SIG_TIME_ SELL	SIGTS	Retorna a hora do último sinal de venda.
SIGP SIG_PRICE	SIGP	Retorna o preço do último sinal.
SIGPB SIG_PRICE _BUY	SIGPB	Retorna o preço do último sinal de compra.
SIGPS SIG_PRICE _SELL	SIGPS	Retorna o preço do último sinal de venda.
SIGTC SIG_TOTAL _CONFIRM	SIGTC	Retorna a soma de sinais confirmados.
SIGLC SIG_LAST_ CONFIRM	SIGLC	Retorna o tipo do último sinal confirmado. (-1 = Nenhum, 0 = Compra, 1 = Venda)
SIGBC SIG_BUY_C ONFIRM	SIGBC	Retorna a soma de sinais de compra confirmados.
SIGSC SIG_SELL_ CONFIRM	SIGSC	Retorna a soma de sinais de venda confirmados.
SIGTLC SIG_TIME_ LAST_CONF	SIGTLC	Retorna a hora do último sinal confirmado.

VARIÁVEIS	ALIASES	RETORNOS
IRM		
SIGTBC SIG_TIME_ BUY_CONFI RM	SIGTBC	Retorna a hora do último sinal de compra confirmado.
SIGTSC SIG_TIME_ SELL_CONF IRM	SIGTSC	Retorna a hora do último sinal de venda confirmado.
SIGPLC SIG_PRICE _LAST_CON FIRM	SIGPLC	Retorna o preço do último sinal confirmado.
SIGPBC SIG_PRICE _BUY_CONF IRM	SIGPBC	Retorna o preço do último sinal de compra confirmado.
SIGPSC SIG_PRICE _SELL_CON FIRM	SIGPSC	Retorna o preço do último sinal de venda confirmado.

Downloads

Nesta página, você encontrará os downloads do ScriptBot.

Documentação em PDF

Baixe a documentação do ScriptBot em PDF.

Baixar PDF

Dev-Build versions

> **Perigo**
>
> Estas versões são de dev-build (versão de desenvolvimento) e podem conter erros, instabilidades ou funcionalidades incompletos. Seu uso em ambiente de produção é de inteira responsabilidade do usuário. Recomenda-se utilizar apenas para testes, desenvolvimento e avaliação, não sendo indicada para uso final em sistemas críticos.

Os downloads e changelogs podem ser encontrados em nosso Grupo de Discord.

FAQ

Nesta página, você encontrará respostas para perguntas frequentes sobre o ScriptBot.

Perguntas Frequentes

Preciso de conhecimentos avançados para usar o ScriptBot?

Em quais corretoras posso usar o ScriptBot?

Em quais mercados o ScriptBot é compatível?

O ScriptBot é compatível com contas Hedge?

O ScriptBot é compatível com contas Netting?

O ScriptBot oferece estratégias prontas para uso?

Como instalar um indicador no ScriptBot?

O ScriptBot funciona em conta real e demo?

Quais versões do ScriptBot estão disponíveis?

Como adquirir uma licença para o ScriptBot?

Como faço para receber suporte caso tenha dúvidas?

Existe garantia ou política de reembolso?

O ScriptBot garante lucro?

Posso personalizar as configurações do ScriptBot?

Log

Esta função é usada para imprimir um texto nas Logs do ScriptBot, ela é útil quando queremos separar textos de debug das Logs do Metatrader.

Parâmetros

string	text1;		// Texto a ser impresso.
string	text2	= "";	// Segundo parametro de texto que será concatenado ao primeiro.
string	text...63	= "";	// Outro parametro de texto que será concatenado, máximo de 63 parametros.

Exemplos

```
1  //Default example:
2  Log[text1, text2 = "", text...63 = ""]
3
4  //Other examples
5  Log["Hello World!"]
6  Log["Symbol: ", SYMBOL, " CurrentTime: ", TIME_CURRENT]
```

Retornos

Em caso de sucesso:

Retorna um valor boolean (true) em caso de sucesso de execução.

Retorno:
bool

Em caso de erro:

Retorna um valor boolean (false) em caso de erro de execução.

Retorno:
bool

Print

Esta função é usada para imprimir um texto nas logs do Metatrader, muito utilizada para depurar o código.

Parâmetros

string	text1;		// Texto a ser impresso.
string	text2	= "";	// Segundo parametro de texto que será concatenado ao primeiro.
string	text...63	= "";	// Outro parametro de texto que será concatenado, máximo de 63 parametros.

Exemplos

```
1  //Default example:
2  Print[text1, text2 = "", text...63 = ""]
3
4  //Other examples
5  Print["Hello World!"]
6  Print["Symbol: ", SYMBOL, " CurrentTime: ", TIME_CURRENT]
```

Retornos

Em caso de sucesso:

Retorna um valor boolean (true) em caso de sucesso de execução.

Retorno:
bool

Em caso de erro:

Retorna um valor boolean (false) em caso de erro de execução.

Retorno:
bool

Indicator

Esta função acessa diretamente o buffer de um indicador e retorna os valores armazenados, permitindo a leitura dos dados calculados pelo indicador em cada barra do gráfico.

Parâmetros

uint	id	= 0;	// Categoria que o indicador esta instalado.
uint	buffer	= 0;	// Buffer do indicador que armazena o valor.
uint	candle	= 0;	// A vela de referância para obter o preço do indicador.

Exemplos

```
1  //Default example:
2  Indicator[id = 0, buffer = 0, candle = 0]
3
4  //Using aliases:
5  IND[id = 0, buffer = 0, candle = 0]
6
7  //Other examples
8  CLOSE[1] <= IND[7, 0, 1] && CLOSE[0] >= IND[7, 1, 1]
```

Retornos

Em caso de sucesso:

Retorna o valor do buffer do indicador do tipo double.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

Indicator'X'

Esta função acessa diretamente o buffer de um indicador e retorna os valores armazenados, permitindo a leitura dos dados calculados pelo indicador em cada barra do gráfico. Esta função é prefixada, sendo necessário substituir o 'x' por um valor entre 0 e 20, onde 'x' representa a categoria em que o indicador está configurado nas definições do robô.

Parâmetros

uint	buffer	= 0;	// Buffer do indicador que armazena o valor.
uint	candle	= 0;	// A vela de referância para obter o preço do indicador.

Exemplos

```
1  //Default example:
2  Indicator'X'[buffer = 0, candle = 0]
3
4  //Using aliases:
5  IND'X'[buffer = 0, candle = 0]
6
7  //Other examples
8  IND1[buffer, candle]
9  CLOSE[1] <= IND7[0, 1] && CLOSE[0] >= IND7[1, 1]
```

Retornos

Em caso de sucesso:

Retorna o valor do buffer do indicador do tipo double.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

ClearDouble

Esta função permite remover todas as variáveis criadas pelo usuário.

Parâmetros

Esta função não requer parâmetros.

Exemplos

```
//Default example:
ClearDouble[]

//Using aliases:
ClrDou[]

//Other examples
GetDouble["Count"] >= 10 -> ClearDouble[];
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se as variáveis foram removidas.

Retorno:

bool

Em caso de erro:

Retorna um boolean (false) se as variáveis não foram removidas ou não existem.

Retorno:

bool

ClearInt

Esta função permite remover todas as variáveis criadas pelo usuário.

Parâmetros

Esta função não requer parâmetros.

Exemplos

```
//Default example:
ClearInt[]

//Using aliases:
ClrInt[]

//Other examples
GetInt["Count"] >= 10 -> ClearInt[];
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se as variáveis foram removidas.

Retorno:

bool

Em caso de erro:

Retorna um boolean (false) se as variáveis não foram removidas ou não existem.

Retorno:

bool

ClearString

Esta função permite remover todas as variáveis criadas pelo usuário.

Parâmetros

Esta função não requer parâmetros.

Exemplos

```
1  //Default example:
2  ClearString[]
3
4  //Using aliases:
5  ClrStr[]
6
7  //Other examples
8  GetInt["Count"] >= 10 -> ClearString[];
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se as variáveis foram removidas.

Retorno:

bool

Em caso de erro:

Retorna um boolean (false) se as variáveis não foram removidas ou não existem.

Retorno:

bool

CreateDouble

Esta função permite criar uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
double	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  CreateDouble[variable, value]
3
4  //Using aliases:
5  CreDou[variable, value]
6
7  //Other examples
8  CreateDouble["Name", 777]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

CreateInt

Esta função permite criar uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
long	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  CreateInt[variable, value]
3
4  //Using aliases:
5  CreInt[variable, value]
6
7  //Other examples
8  CreateInt["Name", 777]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

CreateString

Esta função permite criar uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
string	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  CreateString[variable, value]
3
4  //Using aliases:
5  CreStr[variable, value]
6
7  //Other examples
8  CreateString["Name", "my text"]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

GetDouble

Esta função retorna o valor de uma variável criada pelo usuário.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  GetDouble[variable]
3
4  //Using aliases:
5  GetDou[variable]
6
7  //Other examples
8  GetDouble["Name"
```

Retornos

Em caso de sucesso:

Retorna o valor armazenado na variável.

Retorno:

double

Em caso de erro:

Retorna '0.0' caso a variável não exista.

Retorno:

double

GetInt

Esta função retorna o valor de uma variável criada pelo usuário.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  GetInt[variable]
3
4  //Other examples
5  GetInt["Name"
```

Retornos

Em caso de sucesso:

Retorna o valor armazenado na variável.

Retorno:

long

Em caso de erro:

Retorna '0' caso a variável não exista.

Retorno:

long

GetString

Esta função retorna o valor de uma variável criada pelo usuário.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  GetString[variable]
3
4  //Using aliases:
5  GetStr[variable]
6
7  //Other examples
8  GetString["Name"
```

Retornos

Em caso de sucesso:

Retorna o valor armazenado na variável.

Retorno:

string

Em caso de erro:

Retorna valor vazio caso a variável não exista.

Retorno:

string

RemoveDouble

Esta função permite remover uma variável caso ela exista.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  RemoveDouble[variable]
3
4  //Using aliases:
5  RemDou[variable]
6
7  //Other examples
8  RemoveDouble["Name"
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi removida..

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável nao foi removida ou se não existia uma variável com esse nome.

Retorno:
bool

RemoveInt

Esta função permite remover uma variável caso ela exista.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  RemoveInt[variable]
3
4  //Using aliases:
5  RemInt[variable]
6
7  //Other examples
8  RemoveInt["Name"
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi removida..

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável nao foi removida ou se não existia uma variável com esse nome.

Retorno:
bool

RemoveString

Esta função permite remover uma variável caso ela exista.

Parâmetros

string	variable;		// Nome da variável.

Exemplos

```
1  //Default example:
2  RemoveString[variable]
3
4  //Using aliases:
5  RemStr[variable]
6
7  //Other examples
8  RemoveString["Name"]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi removida..

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável nao foi removida ou se não existia uma variável com esse nome.

Retorno:
bool

ReplaceDouble

Esta função permite substituir o valor de uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
double	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  ReplaceDouble[variable, value]
3
4  //Using aliases:
5  RepDou[variable, value]
6
7  //Other examples
8  ReplaceDouble["Name", 888]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

ReplaceInt

Esta função permite substituir o valor de uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
long	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  ReplaceInt[variable, value]
3
4  //Using aliases:
5  RepInt[variable, value]
6
7  //Other examples
8  ReplaceInt["Name", 888]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

ReplaceString

Esta função permite substituir o valor de uma variável caso não exista.

Parâmetros

string	variable;		// Nome da variável.
string	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  ReplaceString[variable, value]
3
4  //Using aliases:
5  RepStr[variable, value]
6
7  //Other examples
8  ReplaceString["Name", "my other text"]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou variavel já existente.

Retorno:
bool

SetDouble

Esta função define o valor de uma variável. Se a variável não existir, ela será criada, caso já exista, seu valor será substituído.

Parâmetros

string	variable;		// Nome da variável.
double	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  SetDouble[variable, value]
3
4  //Using aliases:
5  SetDou[variable, value]
6
7  //Other examples
8  SetDouble["Name", 777]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada ou atualizada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou o valor não foi atualizado.

Retorno:
bool

SetInt

Esta função define o valor de uma variável. Se a variável não existir, ela será criada, caso já exista, seu valor será substituído.

Parâmetros

string	variable;		// Nome da variável.
long	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  SetInt[variable, value]
3
4  //Other examples
5  SetInt["Name", 777]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada ou atualizada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou o valor não foi atualizado.

Retorno:
bool

SetString

Esta função define o valor de uma variável. Se a variável não existir, ela será criada, caso já exista, seu valor será substituído.

Parâmetros

string	variable;		// Nome da variável.
string	value;		// Valor da variável.

Exemplos

```
1  //Default example:
2  SetString[variable, value]
3
4  //Using aliases:
5  SetStr[variable, value]
6
7  //Other examples
8  SetString["Name", "my other text"]
```

Retornos

Em caso de sucesso:

Retorna um boolean (true) se a variável foi criada ou atualizada.

Retorno:
bool

Em caso de erro:

Retorna um boolean (false) se a variável não foi criada por conta de um erro ou o valor não foi atualizado.

Retorno:
bool

AddDouble

Esta função adiciona um valor double ao valor atual de uma variável. Se a variável não existir, ela será criada com o valor especificado.

Parâmetros

string	name;		// Nome da variável.
double	value;		// Valor double a ser adicionado.

Exemplos

```
1  //Default example:
2  AddDouble[name, value]
3
4  //Using aliases:
5  AddDou[name, value]
6
7  //Other examples
8  AddDouble("saldo", 15.75)
```

Retornos

Em caso de sucesso:

Retorna true se o valor foi adicionado/criado com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se ocorrer um erro na operação.

Retorno:
bool

AddInt

Esta função adiciona um valor inteiro ao valor atual de uma variável. Se a variável não existir, ela será criada com o valor especificado.

Parâmetros

string	name;		// Nome da variável.
int	value;		// Valor inteiro a ser adicionado.

Exemplos

```
1  //Default example:
2  AddInt[name, value]
3
4  //Other examples
5  AddInt("contador", 5)
```

Retornos

Em caso de sucesso:

Retorna true se o valor foi adicionado/criado com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se ocorrer um erro na operação.

Retorno:
bool

GetGlobal

Esta função retorna o valor armazenado em uma variável global do tipo double. Se a variável não existir, retorna 0.0.

Parâmetros

string	name;		// Nome da variável global.

Exemplos

```
1  //Default example:
2  GetGlobal[name]
3
4  //Using aliases:
5  GetG[name]
6
7  //Other examples
8  GetGlobal("minha_variavel")
```

Retornos

Em caso de sucesso:

Retorna o valor double da variável global.

Retorno:
double

Em caso de erro:

Retorna 0.0 se a variável não existir.

Retorno:
double

CreateGlobal

Esta função cria uma nova variável global do tipo double com o valor especificado. Retorna false se a variável já existir.

Parâmetros

string	name;		// Nome da variável global.
double	value;		// Valor inicial da variável.

Exemplos

```
1  //Default example:
2  CreateGlobal[name, value]
3
4  //Using aliases:
5  CreG[name, value]
6
7  //Other examples
8  CreateGlobal("nova_variavel", 123.45)
```

Retornos

Em caso de sucesso:

Retorna true se a variável foi criada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a variável já existir ou ocorrer um erro.

Retorno:
bool

ReplaceGlobal

Esta função substitui o valor de uma variável global do tipo double existente. Retorna false se a variável não existir.

Parâmetros

string	name;		// Nome da variável global.
double	value;		// Novo valor da variável.

Exemplos

```
1  //Default example:
2  ReplaceGlobal[name, value]
3
4  //Using aliases:
5  RepG[name, value]
6
7  //Other examples
8  ReplaceGlobal("minha_variavel", 678.90)
```

Retornos

Em caso de sucesso:

Retorna true se o valor foi substituído com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a variável não existir ou ocorrer um erro.

Retorno:
bool

GetGlobalTime

Esta função retorna o timestamp do último acesso (leitura ou escrita) de uma variável global do tipo double.

Parâmetros

string	name;		// Nome da variável global.

Exemplos

```
1  //Default example:
2  GetGlobalTime[name]
3
4  //Using aliases:
5  GetGT[name]
6
7  //Other examples
8  GetGlobalTime("minha_variavel")
```

Retornos

Em caso de sucesso:

Retorna o timestamp do último acesso da variável.

Retorno:
datetime

Em caso de erro:

Retorna 0 se a variável não existir.

Retorno:
datetime

SetGlobal

Esta função define o valor de uma variável global do tipo double. Se a variável não existir, ela será criada.

Parâmetros

string	name;		// Nome da variável global.
double	value;		// Valor a ser definido.

Exemplos

```
1  //Default example:
2  SetGlobal[name, value]
3
4  //Using aliases:
5  SetG[name, value]
6
7  //Other examples
8  SetGlobal("minha_variavel", 555.55)
```

Retornos

Em caso de sucesso:

Retorna true se o valor foi definido com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se ocorrer um erro.

Retorno:
bool

AddGlobal

Esta função adiciona um valor ao valor atual de uma variável global do tipo double. A variável deve existir previamente.

Parâmetros

string	name;		// Nome da variável global.
double	value;		// Valor a ser adicionado.

Exemplos

```
1  //Default example:
2  AddGlobal[name, value]
3
4  //Using aliases:
5  AddG[name, value]
6
7  //Other examples
8  AddGlobal("contador", 1.0)
```

Retornos

Em caso de sucesso:

Retorna true se o valor foi adicionado com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a variável não existir ou ocorrer um erro.

Retorno:
bool

RemoveGlobal

Esta função remove uma variável global do tipo double do sistema. Retorna false se a variável não existir.

Parâmetros

string	name;		// Nome da variável global a ser removida.

Exemplos

```
1  //Default example:
2  RemoveGlobal[name]
3
4  //Using aliases:
5  RemG[name]
6
7  //Other examples
8  RemoveGlobal("variavel_antiga")
```

Retornos

Em caso de sucesso:

Retorna true se a variável foi removida com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a variável não existir ou ocorrer um erro.

Retorno:
bool

ClearGlobal

Esta função remove todas as variáveis globais do tipo double que começam com o prefixo especificado.

Parâmetros

string	prefix;		// Prefixo das variáveis a serem removidas.

Exemplos

```
1  //Default example:
2  ClearGlobal[prefix]
3
4  //Using aliases:
5  ClrG[prefix]
6
7  //Other examples
8  ClearGlobal("temp_")
```

Retornos

Em caso de sucesso:

Retorna true se as variáveis foram removidas com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se ocorrer um erro.

Retorno:
bool

Open

Esta função retorna o preço de abertura de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Open[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  O[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Open[1]
9  Open[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

Close

Esta função retorna o preço de fechamento de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Close[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  C[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Close[1]
9  Close[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

High

Esta função retorna o preço da máxima de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  High[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  H[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  High[1]
9  High[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

Low

Esta função retorna o preço da mínima de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Low[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  L[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Low[1]
9  Low[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

Time

Esta função retorna o horário de abertura de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Time[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  T[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Time[1]
9  Time[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o horário da vela.

Retorno:
uint

Em caso de erro:

Retorna o valor padrão zero para o tipo uint (0).

Retorno:
uint

Direction

Esta função retorna a direção simplificada de uma vela, ou seja, se a vela é positiva ou negativa. (-1 = Negativa, 0 = Neutra, 1 = Positiva)

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Direction[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  DIR[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Direction[1]
9  Direction[1, M5, "EURUSD"]
10 DIR[1] == 1 // Check if the candle is positive/buying
```

Retornos

Em caso de sucesso:

Retorna a direção da vela.

Retorno:

int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:

int

Spread

Esta função retorna a quantidade de spread de uma vela.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Spread[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  SPD[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  Spread[1]
9  Spread[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o spread da vela.

Retorno:

int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:

int

Candle

Esta função retorna o identificador da vela, usado para obter o index utilizando o horário especificado.

Parâmetros

uint	timeSeconds;		// O horário em segundos.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  Candle[timeSeconds, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Other examples
5  Candle[1]
6  Candle[1, M5, "EURUSD"]
7  Candle[ToTime["13:00"]] // How to get the candle at 13:00 of the day
8  Candle[TIME_CURRENT-300] // How to get the last 5 minutes candles index
```

Retornos

Em caso de sucesso:

Retorna o index da vela.

Retorno:
int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:
int

MaxHigh

Esta função retorna o maior preço entre duas posições. Ela realiza uma varredura do índice inicial até o índice final, identificando e retornando o preço da maior vela encontrada nesse intervalo.

Parâmetros

uint	startIndex;		// O index da primeira vela de referência.
uint	endIndex;		// O index da ultima vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  MaxHigh[startIndex, endIndex, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  MaxH[startIndex, endIndex, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  MaxHigh[0, 20]
9  MaxHigh[0, 20, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da maior vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

MinLow

Esta função retorna o menor preço entre duas posições. Ela realiza uma varredura do índice inicial até o índice final, identificando e retornando o preço da menor vela encontrada nesse intervalo.

Parâmetros

uint	startIndex;		// O index da primeira vela de referência.
uint	endIndex;		// O index da ultima vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  MinLow[startIndex, endIndex, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  MinL[startIndex, endIndex, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  MinLow[0, 20]
9  MinLow[0, 20, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o preço da menor vela.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

TickVolume

Esta função retorna o volume de uma vela com base nos dados de tick.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  TickVolume[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  TICKV[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  TickVolume[1]
9  TickVolume[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o volume da vela.

Retorno:

long

Em caso de erro:

Retorna o valor padrão zero para o tipo long (0).

Retorno:

long

RealVolume

Esta função retorna o volume de uma vela com base em dados reais. Em alguns mercados, no entanto, o volume pode ser constantemente zero devido à ausência de dados de volume no feed de dados, seja por limitações da corretora, do tipo de ativo (como forex, que muitas vezes não fornece volume real), ou pela forma como o volume é calculado e disponibilizado.

Parâmetros

uint	candle;		// A vela de referência.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico de referência.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  RealVolume[candle, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  REALV[candle, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  TickVolume[1]
9  TickVolume[1, M5, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o volume da vela.

Retorno:
long

Em caso de erro:

Retorna o valor padrão zero para o tipo long (0).

Retorno:

long

Median

Esta função retorna a media entre dois valores.

Parâmetros

double	first;		// O primeiro valor.
double	second;		// O segundo valor.

Exemplos

```
1  //Default example:
2  Median[first, second]
3
4  //Using aliases:
5  Med[first, second]
6
7  //Other examples
8  Median[749, 750]
```

Retornos

Em caso de sucesso:

Retorna o preço médio entre os dois valores.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

TickToPoint

Esta função retorna o valor de tick convertido em pontos.

Parâmetros

double	value;		// O valor que será calculado.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  TickToPoint[value, symbol = SYMBOL]
3
4  //Using aliases:
5  TickTP[value, symbol = SYMBOL]
6
7  //Other examples
8  TickToPoint[50]
9  TickToPoint[50, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o valor de tick convertido.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

MoneyToPoint

Esta função retorna o valor na moeda convertida em pontos.

Parâmetros

double	value;		// O valor que será calculado.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  MoneyToPoint[value, symbol = SYMBOL]
3
4  //Using aliases:
5  MoneyTP[value, symbol = SYMBOL]
6
7  //Other examples
8  MoneyToPoint[50]
9  MoneyToPoint[50, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o valor de tick convertido.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

PipToPoint

Esta função retorna o valor do pip convertido em pontos.

Parâmetros

double	value;		// O valor que será calculado.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  PipToPoint[value, symbol = SYMBOL]
3
4  //Using aliases:
5  PipTP[value, symbol = SYMBOL]
6
7  //Other examples
8  PipToPoint[50]
9  PipToPoint[50, "EURUSD"]
```

Retornos

Em caso de sucesso:

Retorna o valor de tick convertido.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

CountStep

Esta função calcula a quantidade de passos realizados e arredonda o resultado para o inteiro mais próximo de zero.

Parâmetros

double	value;		// O valor base.
double	step;		// O valor do passo.

Exemplos

```
1  //Default example:
2  CountStep[value, step]
3
4  //Using aliases:
5  Count[value, step]
6
7  //Other examples
8  CountStep[50, 2]
```

Retornos

Em caso de sucesso:

Retorna a quantidade de passos.

Retorno:
uint

Em caso de erro:

Retorna o valor padrão zero para o tipo uint (0).

Retorno:
uint

CountStepGradual

Esta função calcula a quantidade de passos realizados de forma gradual e arredonda o resultado para o inteiro mais próximo de zero.

Parâmetros

double	value;		// O valor base.
double	step;		// O valor do passo.
double	multiplier	= 1.0;	// O valor do multiplicador.

Exemplos

```
1  //Default example:
2  CountStepGradual[value, step, multiplier = 1.0]
3
4  //Using aliases:
5  CountGrad[value, step, multiplier = 1.0]
6
7  //Other examples
8  CountStepGradual[50, 2]
```

Retornos

Em caso de sucesso:

Retorna a quantidade de passos.

Retorno:
uint

Em caso de erro:

Retorna o valor padrão zero para o tipo uint (0).

Retorno:
uint

Division

Esta função retorna a divisão entre dois valores, permitindo que o denominador seja zero e evitando divisões por zero.

Parâmetros

double	value;		// O valor base.
double	divisor;		// O valor do denominador.

Exemplos

```
1  //Default example:
2  Division[value, divisor]
3
4  //Using aliases:
5  Divi[value, divisor]
6
7  //Other examples
8  Division[50, 2]
```

Retornos

Em caso de sucesso:

Retorna o resultado da divisão.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

RestDivision

Esta função retorna o resto da divisão entre dois valores, permitindo que o denominador seja zero e evitando divisões por zero.

Parâmetros

double	value;		// O valor base.
double	divisor;		// O valor do denominador.

Exemplos

```
1  //Default example:
2  RestDivision[value, divisor]
3
4  //Using aliases:
5  RDivi[value, divisor]
6
7  //Other examples
8  RestDivision[50, 2]
```

Retornos

Em caso de sucesso:

Retorna o resto da divisão.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

Percent

Esta função retorna a variação percentual de value até target, com value sendo o valor base.

Parâmetros

double	value;		// O valor base.
double	target;		// O valor do alvo.

Exemplos

```
1  //Default example:
2  Percent[value, target]
3
4  //Other examples
5  Percent[100, 120] // 20%
6  Percent[100, 50] // -50%
```

Retornos

Em caso de sucesso:

Retorna a variação percentual.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

PercentOfValue

Esta função retorna a quantidade percentual de um valor.

Parâmetros

double	value;		// O valor base.
double	percent;		// O valor percentual que será retornado.

Exemplos

```
1  //Default example:
2  PercentOfValue[value, percent]
3
4  //Using aliases:
5  PercentValue[value, percent]
6
7  //Other examples
8  PercentOfValue[100, 20] // Return 20.0
9  PercentOfValue[200, 50] // Return 100.0
```

Retornos

Em caso de sucesso:

Retorna a quantidade percentual.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

HitPercent

Esta função retorna a quantidade percentual de acerto entre gain e loss. A variação sempre será de 0% a 100%.

Parâmetros

double	gain;		// O valor de quantidade de ganho.
double	loss;		// O valor de quantidade de perda.

Exemplos

```
1  //Default example:
2  HitPercent[gain, loss]
3
4  //Other examples
5  HitPercent[30, 70] // Is 30% of hit percent
6  HitPercent[30, 100] // Is 23,07% of hit percent
```

Retornos

Em caso de sucesso:

Retorna a quantidade percentual de acerto.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

CorrectPrice

Esta função corrige preços inválidos, convertendo-os para valores válidos conforme o passo do mercado.

Parâmetros

double	price;		// O valor do preço.
string	symbol	= SYMBOL;	// O simbolo de referência.

Exemplos

```
1  //Default example:
2  CorrectPrice[price, symbol = SYMBOL]
3
4  //Using aliases:
5  FixPrice[price, symbol = SYMBOL]
```

Retornos

Em caso de sucesso:

Retorna o preço correto do ativo alvo.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

Random

Esta função retorna um valor inteiro aleatório entre dois valores.

Parâmetros

int	min;		// O valor minimo.
int	max;		// O valor maximo.

Exemplos

```
1  //Default example:
2  Random[min, max]
3
4  //Other examples
5  Random[0, 1000]
```

Retornos

Em caso de sucesso:

Retorna um valor inteiro aleatório entre dois valores.

Retorno:

int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:

int

Round

Esta função retorna o valor arredondado.

Parâmetros

double	value;		// O valor para ser arredondado.

Exemplos

```
1  //Default example:
2  Round[value]
3
4  //Other examples
5  Round[10.7] // 11
6  Round[10.3] // 10
7  Round[Close[0]/2]
```

Retornos

Em caso de sucesso:

Retorna o valor arredondado.

Retorno:

double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:

double

RoundUp

Esta função retorna o valor arredondado para cima, garantindo que o resultado seja sempre o próximo inteiro maior ou igual.

Parâmetros

double	value;		// O valor para ser arredondado.

Exemplos

```
1  //Default example:
2  RoundUp[value]
3
4  //Other examples
5  RoundUp[10.7] // 11
6  RoundUp[10.3] // 11
7  RoundUp[Close[0]/2]
```

Retornos

Em caso de sucesso:

Retorna o valor arredondado.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

RoundDown

Esta função retorna o valor arredondado para baixo, garantindo que o resultado seja sempre o próximo inteiro menor ou igual.

Parâmetros

double	value;		// O valor para ser arredondado.

Exemplos

```
1  //Default example:
2  RoundDown[value]
3
4  //Other examples
5  RoundDown[10.7] // 10
6  RoundDown[10.3] // 10
7  RoundDown[Close[0]/2]
```

Retornos

Em caso de sucesso:

Retorna o valor arredondado.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

Max

Esta função retorna o maior valor entre dois números fornecidos.

Parâmetros

double	first;		// o primeiro valor.
double	second;		// O segundo valor.

Exemplos

```
1  //Default example:
2  Max[first, second]
3
4  //Other examples
5  Max[10, 20] // 20
6  Max[60, 40] // 60
```

Retornos

Em caso de sucesso:

Retorna o maior valor.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

Min

Esta função retorna o menor valor entre dois números fornecidos.

Parâmetros

double	first;		// o primeiro valor.
double	second;		// O segundo valor.

Exemplos

```
1  //Default example:
2  Min[first, second]
3
4  //Other examples
5  Min[10, 20] // 10
6  Min[60, 40] // 40
```

Retornos

Em caso de sucesso:

Retorna o menor valor.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

IsChance

Esta função retorna verdadeiro com base em uma chance percentual entre 0 e 100. Por exemplo, ao passar 30, há aproximadamente 30% de chance de retornar verdadeiro.

Parâmetros

double	chance;		// o valor da chance.

Exemplos

```
1  //Default example:
2  IsChance[chance]
3
4  //Using aliases:
5  Chance[chance]
6
7  //Other examples
8  IsChance[10] // It contains 10% chance of returning true.
9  IsChance[60]
```

Retornos

Em caso de sucesso:

Retorna verdadeiro caso a chance seja atingida.

Retorno:
bool

Em caso de erro:

Retorna falso caso a chance não seja atingida.

Retorno:
bool

Decimals

Esta função retorna a quantidade de casas decimais presentes em um número, indicando seu nível de precisão.

Parâmetros

double	value;		// o valor a ser verificado.

Exemplos

```
1  //Default example:
2  Decimals[value]
3
4  //Using aliases:
5  Decimal[value]
6
7  //Other examples
8  Decimals[10.00002] // Is 5
```

Retornos

Em caso de sucesso:

Retorna a quantidade de casas decimais.

Retorno:
int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:
int

GetVolume

Esta função retorna o volume que a ordem será colocada no mercado.

Parâmetros

Esta função não requer parâmetros.

Exemplos

```
1  //Default example:
2  GetVolume[]
3
4  //Using aliases:
5  GetVol[]
```

Retornos

Em caso de sucesso:

Retorna o volume.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0.0).

Retorno:
double

SetVolume

Esta função define o novo volume utilizado no sistema de envio de ordens. Ela altera diretaemente o valor no gerenciador de volume, portanto, é importante considerar o método de cálculo do mesmo.

Parâmetros

double	value;		// o valor do volume.
double	defaultValue	= VOLUME_MIN;	// o valor padrão do volume.

Exemplos

```
1  //Default example:
2  SetVolume[value, defaultValue = VOLUME_MIN]
3
4  //Using aliases:
5  SetVol[value, defaultValue = VOLUME_MIN]
6
7  //Other examples
8  SetVol[10]
```

Retornos

Em caso de sucesso:

Retorna true se o volume for alterado com sucesso, ou false caso contrário.

Retorno:
bool

Em caso de erro:

Retorna false em caso de erro ou falha.

Retorno:
bool

ToPositive

Esta função retorna o valor absoluto de um número, ou seja, o seu valor sem sinal (sempre positivo).

Parâmetros

double	value;		// o valor a ser convertido.

Exemplos

```
1  //Default example:
2  ToPositive[value]
3
4  //Using aliases:
5  Pos[value]
6
7  //Other examples
8  ToPositive[-10] // Is 10
9  ToPositive[Close[0]-Open[0]] // Result of the calculation "close [0] -Open [0]" will be pos
```

Retornos

Em caso de sucesso:

Retorna o valor absoluto.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

ToNegative

Esta função retorna o valor negativo de um número, ou seja, o seu valor com sinal (sempre negativo).

Parâmetros

double	value;		// o valor a ser convertido.

Exemplos

```
1  //Default example:
2  ToNegative[value]
3
4  //Using aliases:
5  Neg[value]
6
7  //Other examples
8  ToNegative[10] // Is -10
9  ToNegative[Close[0]-Open[0]] // Result of the calculation "close [0] -Open [0]" will be neg
```

Retornos

Em caso de sucesso:

Retorna o valor negativo.

Retorno:
double

Em caso de erro:

Retorna o valor padrão zero para o tipo double (0).

Retorno:
double

ToTime

Esta função retorna o horário em forma de texto para número em segundos. Com esta função você pode acessar a vela do dia usando um horário em forma de texto, por exemplo '13:00'.

Parâmetros

string	time;		// o valor do horário.
string	symbol	= SYMBOL;	// O ativo de referência.

Exemplos

```
1  //Default example:
2  ToTime[time, symbol = SYMBOL]
3
4  //Other examples
5  ToTime["19:00"]
```

Retornos

Em caso de sucesso:

Retorna o horário em segundos.

Retorno:
uint

Em caso de erro:

Retorna o valor padrão zero para o tipo uint (0).

Retorno:
uint

ToTimeModify

Esta função permite modificar um horário passado de referencia.

Parâmetros

int	time;		// o valor do horário em segundos.
int	seconds	= -1;	// O valor de segundos a ser modificado ou -1 para ignorar.
int	minutes	= -1;	// O valor de minutos a ser modificado ou -1 para ignorar.
int	hours	= -1;	// O valor de horas a ser modificado ou -1 para ignorar.
int	day	= -1;	// O valor do dia a ser modificado ou -1 para ignorar.
int	month	= -1;	// O valor do mês a ser modificado ou -1 para ignorar.
int	year	= -1;	// O valor do ano a ser modificado ou -1 para ignorar.

Exemplos

```
1  //Default example:
2  ToTimeModify[time, seconds = -1, minutes = -1, hours = -1, day = -1, month = -1, year = -
3
4  //Using aliases:
5  TMod[time, seconds = -1, minutes = -1, hours = -1, day = -1, month = -1, year = -1]
6
7  //Other examples
8  ToTimeModify[TIME_CURRENT, 50, 10, 12] // I would modify the seconds, minutes and hours.
9  ToTimeModify[TIME_CURRENT, -1, -1, -1, 10] // I would only change the day, the other data
```

Retornos

Em caso de sucesso:

Retorna o novo horário em segundos.

Retorno:

uint

Em caso de erro:

Retorna o valor padrão zero para o tipo uint (0).

Retorno:

uint

ToTimeFormat

Esta função transforma um valor em segundos em uma representação textual de horário.

Parâmetros

uint	time;		// o valor do horário.

Exemplos

```
1  //Default example:
2  ToTimeFormat[time]
3
4  //Using aliases:
5  TFormat[time]
6
7  //Other examples
8  ToTimeFormat[TIME_CURRENT] // It would be similar to: "1970.01.01 19:00:00"
```

Retornos

Em caso de sucesso:

Retorna o horário em forma de texto.

Retorno:

string

Em caso de erro:

Retorna a string vazia.

Retorno:

string

ToFormat

Esta função permite modificar um texto, formatando-o de acordo com os parâmetros definidos internamente.

Parâmetros

string	text;		// o valor do texto.
any	param1	= "";	// Primeiro parametro que será formatado.
any	param...63	= "";	// Os demais parametros que serão formatados. Não pode ser mais de 63.

Exemplos

```
1  //Default example:
2  ToFormat[text, param1 = "", param...63 = ""]
3
4  //Using aliases:
5  Format[text, param1 = "", param...63 = ""]
6
7  //Other examples
8  ToFormat["Hello, {0}!", "World"]] // New text: "Hello, World!"
9  ToFormat["My text is {0} and my number is {1}!", "Metatrader is God", 10]] // New text: "My
10 ToFormat["My list: {0}, {1}, again {0}!", "Jujubas", 777]] // New text: "My list: Jujubas,
```

Retornos

Em caso de sucesso:

Retorna o novo texto formatado.

Retorno:
string

Em caso de erro:

Retorna a string vazia.

Retorno:
string

ToInt

Esta função retorna o valor convertido para inteiro. Se o valor for uma string, ele será convertido para zero. Caso seja um número decimal, será arredondado para o inteiro mais próximo.

Parâmetros

any	value;		// o valor a ser convertido.

Exemplos

```
1  //Default example:
2  ToInt[value]
3
4  //Other examples
5  ToInt[10.5] // Is 10
6  ToInt["9999"] // Is 9999
7  ToInt["My age 30"] // To convert to zero
```

Retornos

Em caso de sucesso:

Retorna o valor convertido para inteiro.

Retorno:
int

Em caso de erro:

Retorna o valor padrão zero para o tipo int (0).

Retorno:
int

Last

Esta função retorna o valor do último preço de negociação para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  Last[symbol = SYMBOL]
3
4  //Other examples
5  Last["PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o último preço.

Retorno:
double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:
double

LastHigh

Esta função retorna o maior preço registrado no último negócio do símbolo informado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  LastHigh[symbol = SYMBOL]
3
4  //Using aliases:
5  LastH[symbol = SYMBOL]
6
7  //Other examples
8  LastHigh["VALE3"]
```

Retornos

Em caso de sucesso:

Retorna o maior preço do último negócio.

Retorno:

double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

double

LastLow

Esta função retorna o menor preço registrado no último negócio do símbolo informado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  LastLow[symbol = SYMBOL]
3
4  //Using aliases:
5  LastL[symbol = SYMBOL]
6
7  //Other examples
8  LastLow["VALE3"]
```

Retornos

Em caso de sucesso:

Retorna o menor preço do último negócio.

Retorno:
double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:
double

Ask

Esta função retorna o preço de venda (ask) atual do símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  Ask[symbol = SYMBOL]
3
4  //Other examples
5  Ask["PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o preço de venda.

Retorno:
double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:
double

AskHigh

Esta função retorna o maior valor de ask registrado no dia para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  AskHigh[symbol = SYMBOL]
3
4  //Using aliases:
5  AskH[symbol = SYMBOL]
6
7  //Other examples
8  AskHigh["ITUB4"]
```

Retornos

Em caso de sucesso:

Retorna o maior preço de venda.

Retorno:

double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

double

AskLow

Esta função retorna o menor valor de ask registrado no dia para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  AskLow[symbol = SYMBOL]
3
4  //Using aliases:
5  AskL[symbol = SYMBOL]
6
7  //Other examples
8  AskLow["ITUB4"]
```

Retornos

Em caso de sucesso:

Retorna o menor preço de venda.

Retorno:

double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

double

Bid

Esta função retorna o preço de compra (bid) atual do símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  Bid[symbol = SYMBOL]
3
4  //Other examples
5  Bid["BBAS3"]
```

Retornos

Em caso de sucesso:

Retorna o preço de compra.

Retorno:
double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:
double

BidHigh

Esta função retorna o maior valor de bid registrado no dia para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  BidHigh[symbol = SYMBOL]
3
4  //Using aliases:
5  BidH[symbol = SYMBOL]
6
7  //Other examples
8  BidHigh["BBAS3"]
```

Retornos

Em caso de sucesso:

Retorna o maior preço de compra.

Retorno:

double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

double

BidLow

Esta função retorna o menor valor de bid registrado no dia para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  BidLow[symbol = SYMBOL]
3
4  //Using aliases:
5  BidL[symbol = SYMBOL]
6
7  //Other examples
8  BidLow["BBAS3"]
```

Retornos

Em caso de sucesso:

Retorna o menor preço de compra.

Retorno:

double

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

double

LastTime

Esta função retorna o horário da última negociação para o símbolo especificado.

Parâmetros

string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  LastTime[symbol = SYMBOL]
3
4  //Using aliases:
5  LastT[symbol = SYMBOL]
6
7  //Other examples
8  LastTime["PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o horário da última negociação.

Retorno:

datetime

Em caso de erro:

Retorna zero se o símbolo for inválido.

Retorno:

datetime

DayStartTime

Esta função retorna o horário de abertura do pregão para o ativo no dia informado. Pode variar conforme o ativo e o mercado.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayStartTime[day, symbol = SYMBOL]
3
4  //Using aliases:
5  DayST[day, symbol = SYMBOL]
6
7  //Other examples
8  DayStartTime[0, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o horário de abertura do pregão.

Retorno:
datetime

Em caso de erro:

Retorna null se os dados estiverem indisponíveis.

Retorno:
datetime

DayEndTime

Esta função retorna o horário de fechamento do pregão para o ativo no dia informado.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayEndTime[day, symbol = SYMBOL]
3
4  //Using aliases:
5  DayET[day, symbol = SYMBOL]
6
7  //Other examples
8  DayEndTime[0, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o horário de fechamento do pregão.

Retorno:
datetime

Em caso de erro:

Retorna null se os dados estiverem indisponíveis.

Retorno:
datetime

DayStartCandle

Esta função retorna o identificador da primeira vela (candle) do dia para o ativo especificado.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayStartCandle[day, symbol = SYMBOL]
3
4  //Using aliases:
5  DaySC[day, symbol = SYMBOL]
6
7  //Other examples
8  DayStartCandle[20250719, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna a vela inicial do dia.

Retorno:
uint

Em caso de erro:

Retorna 0 se não houver vela no dia.

Retorno:
uint

DayEndCandle

Esta função retorna o identificador da última vela (candle) do dia para o ativo especificado.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayEndCandle[day, symbol = SYMBOL]
3
4  //Using aliases:
5  DayEC[day, symbol = SYMBOL]
6
7  //Other examples
8  DayEndCandle[0, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna a última vela do dia.

Retorno:
uint

Em caso de erro:

Retorna 0 se não houver vela no dia.

Retorno:
uint

DayHigh

Esta função retorna o maior valor de preço negociado no dia, para o timeframe e símbolo informados.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayHigh[day, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  DayH[day, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  DayHigh[0, M5, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o maior preço do dia.

Retorno:
double

Em caso de erro:

Retorna zero se não houver dados.

Retorno:
double

DayLow

Esta função retorna o menor valor de preço negociado no dia, para o timeframe e símbolo informados.

Parâmetros

int	day;		// O valor do dia, sendo 0 o dia atual, 1 o dia anterior, etc.
ENUM_TIMEFRAME	timeframe	= CURRENT;	// O tempo gráfico.
string	symbol	= SYMBOL;	// O símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayLow[day, timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  DayL[day, timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  DayLow[0, M5, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o menor preço do dia.

Retorno:
double

Em caso de erro:

Retorna zero se não houver dados.

Retorno:
double

IsTime

Esta função verifica se o horário de referência corresponde aos valores fornecidos para hora, minuto e segundo. Caso algum dos valores seja -1, ele será ignorado no critério.

Parâmetros

datetime	timeRef;		// Horário de referência.
int	hour	= -1;	// Hora (0 a 23) ou -1 para ignorar.
int	minute	= -1;	// Minuto (0 a 59) ou -1 para ignorar.
int	second	= -1;	// Segundo (0 a 59) ou -1 para ignorar.

Exemplos

```
1  //Default example:
2  IsTime[timeRef, hour = -1, minute = -1, second = -1]
3
4  //Other examples
5  IsTime[TIME_CURRENT, 10] // Returns true if the hour is 10
6  IsTime[TIME_CURRENT, -1, 30] // Returns true if the minute is 30
```

Retornos

Em caso de sucesso:

Retorna true se o horário corresponder aos critérios.

Retorno:
bool

Em caso de erro:

Retorna false se não corresponder ou houver erro nos parâmetros.

Retorno:
bool

SymbolTime

Esta função retorna o horário atual do ativo (símbolo) no mercado correspondente.

Parâmetros

string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  SymbolTime[symbol = SYMBOL]
3
4  //Using aliases:
5  STime[symbol = SYMBOL]
6
7  //Other examples
8  SymbolTime["PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o horário atual do ativo.

Retorno:

datetime

Em caso de erro:

Retorna null se o símbolo for inválido.

Retorno:

datetime

TimeLeft

Esta função retorna quanto tempo falta para o término da vela atual no timeframe e símbolo especificados.

Parâmetros

ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.
string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  TimeLeft[timeframe = CURRENT, symbol = SYMBOL]
3
4  //Using aliases:
5  TLeft[timeframe = CURRENT, symbol = SYMBOL]
6
7  //Other examples
8  TimeLeft[M5, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o tempo restante (em segundos) da vela atual.

Retorno:
int

Em caso de erro:

Retorna -1 se os parâmetros forem inválidos.

Retorno:
int

DayOfWeek

Esta função retorna o dia da semana (0 = domingo, 1 = segunda, ..., 6 = sábado) do símbolo informado.

Parâmetros

string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayOfWeek[symbol = SYMBOL]
3
4  //Using aliases:
5  DayW[symbol = SYMBOL]
6
7  //Other examples
8  DayOfWeek["PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o número correspondente ao dia da semana.

Retorno:
int

Em caso de erro:

Retorna -1 se não for possível determinar o dia.

Retorno:
int

DayOfYear

Esta função retorna o número do dia no ano (1 a 366) com base no horário atual do símbolo.

Parâmetros

string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  DayOfYear[symbol = SYMBOL]
3
4  //Using aliases:
5  DayY[symbol = SYMBOL]
6
7  //Other examples
8  DayOfYear["VALE3"]
```

Retornos

Em caso de sucesso:

Retorna o dia do ano.

Retorno:

int

Em caso de erro:

Retorna -1 se o símbolo for inválido.

Retorno:

int

Month

Esta função retorna o número do mês atual (1 a 12) com base no horário atual do símbolo.

Parâmetros

string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  Month[symbol = SYMBOL]
3
4  //Other examples
5  Month["VALE3"]
```

Retornos

Em caso de sucesso:

Retorna o número do mês.

Retorno:

int

Em caso de erro:

Retorna -1 se não for possível determinar o mês.

Retorno:

int

Year

Esta função retorna o número do ano atual com base no horário do símbolo.

Parâmetros

string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  Year[symbol = SYMBOL]
3
4  //Other examples
5  Year["VALE3"]
```

Retornos

Em caso de sucesso:

Retorna o ano atual (ex: 2025).

Retorno:

int

Em caso de erro:

Retorna -1 se não for possível determinar o ano.

Retorno:

int

StartTime

Esta função retorna o horário de início conforme o método informado. Pode representar abertura do dia, semana, mês, etc.

Parâmetros

ENUM_TIME_HISTORIC	method;		// Método de cálculo (ex: 'TYPE_DAY', 'TYPE_WEEK' ...).
string	symbol	= SYMBOL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  StartTime[method, symbol = SYMBOL]
3
4  //Using aliases:
5  StartT[method, symbol = SYMBOL]
6
7  //Other examples
8  StartTime[TYPE_DAY, "PETR4"]
```

Retornos

Em caso de sucesso:

Retorna o horário de início conforme o método.

Retorno:
datetime

Em caso de erro:

Retorna null se o método ou símbolo for inválido.

Retorno:
datetime

ValueOfTime

Esta função retorna um valor numérico representando o tempo, conforme o método especificado.

Podendo ser segundos, minutos, horas, etc..

Parâmetros

datetime	time;		// Horário a ser convertido.
ENUM_TIME	method;		// Método de retorno. (ex: 'TYPE_SEC', 'TYPE_MIN' ...).

Exemplos

```
1  //Default example:
2  ValueOfTime[time, method]
3
4  //Using aliases:
5  ValueT[time, method]
6
7  //Other examples
8  ValueOfTime[TimeNow, TYPE_SEC]
```

Retornos

Em caso de sucesso:

Retorna o valor numérico correspondente ao horário.

Retorno:

int

Em caso de erro:

Retorna -1 se o horário ou método forem inválidos.

Retorno:

int

AllVolumeOpen

Esta função retorna o volume total combinado de todas as posições e ordens abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  AVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllVolumeOpen["PETR4", 123456, -1, H1]
```

Retornos

Em caso de sucesso:

Retorna o volume total de todas as posições e ordens abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições e ordens abertas.

Retorno:
double

BuyVolumeOpen

Esta função retorna o volume total combinado de todas as posições e ordens de compra abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyVolumeOpen["VALE3", 123456, -1, D1]
```

Retornos

Em caso de sucesso:

Retorna o volume total das posições e ordens de compra abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições e ordens de compra abertas.

Retorno:
double

SellVolumeOpen

Esta função retorna o volume total combinado de todas as posições e ordens de venda abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellVolumeOpen["ITUB4", 123456, -1, W1]
```

Retornos

Em caso de sucesso:

Retorna o volume total das posições e ordens de venda abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições e ordens de venda abertas.

Retorno:
double

AllPosVolumeOpen

Esta função retorna o volume total combinado de todas as posições abertas.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllPosVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  APVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllPosVolumeOpen["BBAS3", 123456, -1, MN1]
```

Retornos

Em caso de sucesso:

Retorna o volume total de todas as posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

BuyPosVolumeOpen

Esta função retorna o volume total combinado de todas as posições de compra abertas.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyPosVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BPVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyPosVolumeOpen["PETR4", 123456, -1, H4]
```

Retornos

Em caso de sucesso:

Retorna o volume total das posições de compra abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de compra abertas.

Retorno:
double

SellPosVolumeOpen

Esta função retorna o volume total combinado de todas as posições de venda abertas.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellPosVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SPVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellPosVolumeOpen["VALE3", 123456, -1, D1]
```

Retornos

Em caso de sucesso:

Retorna o volume total das posições de venda abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de venda abertas.

Retorno:
double

AllOrderVolumeOpen

Esta função retorna o volume total combinado de todas as ordens pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllOrderVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  AOVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllOrderVolumeOpen["ITUB4", 123456, -1, W1]
```

Retornos

Em caso de sucesso:

Retorna o volume total de todas as ordens pendentes.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver ordens pendentes.

Retorno:
double

BuyOrderVolumeOpen

Esta função retorna o volume total combinado de todas as ordens de compra pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyOrderVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BOVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyOrderVolumeOpen["BBAS3", 123456, -1, MN1]
```

Retornos

Em caso de sucesso:

Retorna o volume total das ordens de compra pendentes.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver ordens de compra pendentes.

Retorno:
double

SellOrderVolumeOpen

Esta função retorna o volume total combinado de todas as ordens de venda pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellOrderVolumeOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SOVolO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellOrderVolumeOpen["PETR4", 123456, -1, H1]
```

Retornos

Em caso de sucesso:

Retorna o volume total das ordens de venda pendentes.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver ordens de venda pendentes.

Retorno:
double

AllOpen

Esta função retorna o total combinado de posições abertas e ordens pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  AOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllOpen["VALE3", 123456, -1, D1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições e ordens abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições ou ordens abertas.

Retorno:
int

BuyOpen

Esta função retorna o total combinado de posições de compra abertas e ordens de compra pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyOpen["ITUB4", 123456, -1, W1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições e ordens de compra abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições ou ordens de compra abertas.

Retorno:
int

SellOpen

Esta função retorna o total combinado de posições de venda abertas e ordens de venda pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellOpen["BBAS3", 123456, -1, MN1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições e ordens de venda abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições ou ordens de venda abertas.

Retorno:
int

AllPosOpen

Esta função retorna o total de posições abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllPosOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  APosO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllPosOpen["PETR4", 123456, -1, H4]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições abertas.

Retorno:
int

BuyPosOpen

Esta função retorna o total de posições de compra abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyPosOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BPosO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyPosOpen["VALE3", 123456, -1, D1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições de compra abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições de compra abertas.

Retorno:
int

SellPosOpen

Esta função retorna o total de posições de venda abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellPosOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SPosO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellPosOpen["ITUB4", 123456, -1, W1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de posições de venda abertas.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver posições de venda abertas.

Retorno:
int

AllOrderOpen

Esta função retorna o total de ordens pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllOrderOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  AOrderO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllOrderOpen["BBAS3", 123456, -1, MN1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de ordens pendentes.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver ordens pendentes.

Retorno:
int

BuyOrderOpen

Esta função retorna o total de ordens de compra pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyOrderOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BOrderO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyOrderOpen["PETR4", 123456, -1, H1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de ordens de compra pendentes.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver ordens de compra pendentes.

Retorno:
int

SellOrderOpen

Esta função retorna o total de ordens de venda pendentes, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellOrderOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SOrderO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellOrderOpen["VALE3", 123456, -1, D1]
```

Retornos

Em caso de sucesso:

Retorna a quantidade total de ordens de venda pendentes.

Retorno:
int

Em caso de erro:

Retorna 0 se não houver ordens de venda pendentes.

Retorno:
int

AllProfitOpen

Esta função retorna o lucro/prejuízo total combinado de todas as posições abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  AllProfitOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  AProfitO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  AllProfitOpen["ITUB4", 123456, -1, W1]
```

Retornos

Em caso de sucesso:

Retorna o lucro/prejuízo total de todas as posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

BuyProfitOpen

Esta função retorna o lucro/prejuízo total combinado de todas as posições de compra abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  BuyProfitOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  BProfitO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  BuyProfitOpen["BBAS3", 123456, -1, MN1]
```

Retornos

Em caso de sucesso:

Retorna o lucro/prejuízo total das posições de compra abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de compra abertas.

Retorno:
double

SellProfitOpen

Esta função retorna o lucro/prejuízo total combinado de todas as posições de venda abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
int	candle	= -1;	// Índice da vela de inicio (-1 para todas).
ENUM_TIMEFRAME	timeframe	= CURRENT;	// Tempo gráfico.

Exemplos

```
1  //Default example:
2  SellProfitOpen[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
3
4  //Using aliases:
5  SProfitO[symbol = REAL, magic = MAGIC, candle = -1, timeframe = CURRENT]
6
7  //Other examples
8  SellProfitOpen["PETR4", 123456, -1, H4]
```

Retornos

Em caso de sucesso:

Retorna o lucro/prejuízo total das posições de venda abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de venda abertas.

Retorno:
double

AvgPrice

Esta função retorna o preço médio ponderado pelo volume de todas as posições abertas, considerando o tipo de operação e os filtros fornecidos.

Parâmetros

ENUM_POSITION_TYPE	type	= TYPE_ALL;	// Tipo de posição (COMPRA/VENDA/TODAS).
string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  AvgPrice[type = TYPE_ALL, symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  PAvgP[type = TYPE_ALL, symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  AvgPrice(POSITION_TYPE_BUY, "VALE3", 123456)
```

Retornos

Em caso de sucesso:

Retorna o preço médio das posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

AvgDir

Esta função retorna a direção média ponderada das posições abertas, considerando o tipo de operação e os filtros fornecidos.

Parâmetros

ENUM_POSITION_TYPE	type	= TYPE_ALL;	// Tipo de posição (COMPRA/VENDA/TODAS).
string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  AvgDir[type = TYPE_ALL, symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  PAvgDir[type = TYPE_ALL, symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  AvgDir(POSITION_TYPE_SELL, "ITUB4", 123456)
```

Retornos

Em caso de sucesso:

Retorna a direção média das posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

AllAvgPrice

Esta função retorna o preço médio ponderado pelo volume de todas as posições abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  AllAvgPrice[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  AAvgPrice[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  AllAvgPrice("BBAS3", 123456)
```

Retornos

Em caso de sucesso:

Retorna o preço médio de todas as posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

BuyAvgPrice

Esta função retorna o preço médio ponderado pelo volume de todas as posições de compra abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  BuyAvgPrice[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  BAvgPrice[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  BuyAvgPrice("PETR4", 123456)
```

Retornos

Em caso de sucesso:

Retorna o preço médio das posições de compra abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de compra abertas.

Retorno:
double

SellAvgPrice

Esta função retorna o preço médio ponderado pelo volume de todas as posições de venda abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  SellAvgPrice[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  SAvgPrice[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  SellAvgPrice("VALE3", 123456)
```

Retornos

Em caso de sucesso:

Retorna o preço médio das posições de venda abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de venda abertas.

Retorno:
double

AllAvgDir

Esta função retorna a direção média ponderada de todas as posições abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  AllAvgDir[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  AAvgDir[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  AllAvgDir("ITUB4", 123456)
```

Retornos

Em caso de sucesso:

Retorna a direção média de todas as posições abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições abertas.

Retorno:
double

BuyAvgDir

Esta função retorna a direção média ponderada de todas as posições de compra abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  BuyAvgDir[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  BAvgDir[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  BuyAvgDir("BBAS3", 123456)
```

Retornos

Em caso de sucesso:

Retorna a direção média das posições de compra abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de compra abertas.

Retorno:
double

SellAvgDir

Esta função retorna a direção média ponderada de todas as posições de venda abertas, considerando os filtros fornecidos.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  SellAvgDir[symbol = REAL, magic = MAGIC]
3
4  //Using aliases:
5  SAvgDir[symbol = REAL, magic = MAGIC]
6
7  //Other examples
8  SellAvgDir("PETR4", 123456)
```

Retornos

Em caso de sucesso:

Retorna a direção média das posições de venda abertas.

Retorno:
double

Em caso de erro:

Retorna 0.0 se não houver posições de venda abertas.

Retorno:
double

PosTicket

Esta função retorna o ticket de uma posição aberta usando os parâmetros informados.

Parâmetros

int	index;		// Indice da posição. Começando com 0 para a mais atual.
ENUM_TRADE	type	= TYPE_ALL;	// Tipo de trade: 'TYPE_BUY', 'TYPE_SELL', 'TYPE_ALL'.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
string	symbol	= REAL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  PosTicket[index, type = TYPE_ALL, magic = MAGIC, symbol = REAL]
3
4  //Using aliases:
5  PTicket[index, type = TYPE_ALL, magic = MAGIC, symbol = REAL]
6
7  //Other examples
8  PosTicket[0, TYPE_BUY] // Retorna o ticket da compra mais recente
```

Retornos

Em caso de sucesso:

Retorna o ticket da posição.

Retorno:
ulong

Em caso de erro:

Retorna 0 se nenhuma posição for encontrada com os parâmetros informados.

Retorno:
ulong

PosDouble

Esta função retorna o valor de uma propriedade específica do tipo double (número com casas decimais) de uma posição aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.
ENUM_POS_DOUBLE	property;		// Propriedade da posição: 'TYPE_VOLUME' (volume/lotes), 'TYPE_OPEN' (preço de abertura).

Exemplos

```
1  //Default example:
2  PosDouble[ticket, property]
3
4  //Other examples
5  PosDouble[123456, TYPE_VOLUME] // Retorna o volume/lotes da posição
6  PosDouble[PosTicket[0], TYPE_OPEN] // Retorna o preço de abertura da posição
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
double

Em caso de erro:

Retorna 0.0 se a posição não existir ou a propriedade for inválida.

Retorno:
double

PosInt

Esta função retorna o valor de uma propriedade específica do tipo inteiro de uma posição aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.
ENUM_POS_INT	property;		// Propriedade da posição: 'TYPE_TIME' (timestamp de abertura), 'TYPE_MAGIC' (número mágico).

Exemplos

```
1  //Default example:
2  PosInt[ticket, property]
3
4  //Other examples
5  PosInt[123456, TYPE_TIME] // Retorna o timestamp de abertura da posição
6  PosInt[PosTicket[0], TYPE_MAGIC] // Retorna o número mágico da posição
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
int

Em caso de erro:

Retorna 0 se a posição não existir ou a propriedade for inválida.

Retorno:
int

PosString

Esta função retorna o valor de uma propriedade específica do tipo string (texto) de uma posição aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.
ENUM_POS_STRING	property;		// Propriedade da posição: 'TYPE_SYMBOL' (símbolo do ativo), 'TYPE_COMMENT' (comentário da operação).

Exemplos

```
1  //Default example:
2  PosString[ticket, property]
3
4  //Other examples
5  PosString[123456, TYPE_SYMBOL] // Retorna o símbolo do ativo da posição
6  PosString[PosTicket[0], TYPE_COMMENT] // Retorna o comentário da posição
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
string

Em caso de erro:

Retorna uma string vazia se a posição não existir ou a propriedade for inválida.

Retorno:
string

PosCurrent

Esta função retorna o preço atual de uma posição aberta identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosCurrent[ticket]
3
4  //Using aliases:
5  PCurrent[ticket]
6
7  //Other examples
8  PosCurrent[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço atual da posição.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:

double

PosOpen

Esta função retorna o preço de abertura de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosOpen[ticket]
3
4  //Using aliases:
5  PO[ticket]
6
7  //Other examples
8  PosOpen[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço de abertura da posição.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:

double

PosProfit

Esta função retorna o lucro ou prejuízo atual (em moeda do ativo) de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosProfit[ticket]
3
4  //Using aliases:
5  PProfit[ticket]
6
7  //Other examples
8  PosProfit[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o lucro/prejuízo atual da posição.

Retorno:
double

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
double

PosStop

Esta função retorna o preço definido para o Stop Loss de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosStop[ticket]
3
4  //Using aliases:
5  PSL[ticket]
6
7  //Other examples
8  PosStop[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do Stop Loss.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não tiver Stop Loss definido ou não for encontrada.

Retorno:

double

PosSwap

Esta função retorna o valor total de swap acumulado para uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosSwap[ticket]
3
4  //Using aliases:
5  PSWAP[ticket]
6
7  //Other examples
8  PosSwap[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o valor acumulado de swap.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:

double

PosTake

Esta função retorna o preço definido para o Take Profit de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosTake[ticket]
3
4  //Using aliases:
5  PTP[ticket]
6
7  //Other examples
8  PosTake[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do Take Profit.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não tiver Take Profit definido ou não for encontrada.

Retorno:

double

PosVolume

Esta função retorna o volume (tamanho) de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosVolume[ticket]
3
4  //Using aliases:
5  PVol[ticket]
6
7  //Other examples
8  PosVolume[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o volume da posição.

Retorno:

double

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:

double

PosIdentifier

Esta função retorna o identificador da posição, um valor único atribuído a cada nova posição aberta, que permanece inalterado durante toda a sua existência. Alterações na posição não modificam esse identificador.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosIdentifier[ticket]
3
4  //Using aliases:
5  PIdent[ticket]
6
7  //Other examples
8  PosIdentifier[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ticket da posição.

Retorno:
int

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
int

PosMagic

Esta função retorna o número mágico (identificador do robô) de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosMagic[ticket]
3
4  //Using aliases:
5  PMagic[ticket]
6
7  //Other examples
8  PosMagic[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o número mágico da posição.

Retorno:
ulong

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
ulong

PosReason

Esta função retorna o código que indica o motivo pelo qual a posição foi aberta.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosReason[ticket]
3
4  //Using aliases:
5  PReason[ticket]
6
7  //Other examples
8  PosReason[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do motivo de abertura. 0: Terminal desktop. 1: Aplicativo móvel. 2: Plataforma web. 3: Expert Advisor, script ou outro código MQL5

Retorno:
int

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
int

PosTime

Esta função retorna o horário de abertura de uma posição no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosTime[ticket]
3
4  //Using aliases:
5  PT[ticket]
6
7  //Other examples
8  PosTime[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime de abertura da posição.

Retorno:
datetime

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
datetime

PosTimeMsc

Esta função retorna o horário de abertura de uma posição no formato datetime em milissegundos desde 01/01/1970.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosTimeMsc[ticket]
3
4  //Using aliases:
5  PTMsc[ticket]
6
7  //Other examples
8  PosTimeMsc[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o timestamp em milissegundos.

Retorno:

long

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:

long

PosTimeUpdate

Esta função retorna o horário da última atualização (modificação) de uma posição no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosTimeUpdate[ticket]
3
4  //Using aliases:
5  PTUpdate[ticket]
6
7  //Other examples
8  PosTimeUpdate[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime da última atualização.

Retorno:
datetime

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
datetime

PosTimeUpdateMsc

Esta função retorna o horário da última atualização de uma posição no formato datetime em milissegundos desde 01/01/1970.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosTimeUpdateMsc[ticket]
3
4  //Using aliases:
5  PTUpdateMsc[ticket]
6
7  //Other examples
8  PosTimeUpdateMsc[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o timestamp em milissegundos.

Retorno:
long

Em caso de erro:

Retorna 0 se a posição não for encontrada.

Retorno:
long

PosType

Esta função retorna o tipo da posição (0 para compra, 1 para venda) identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosType[ticket]
3
4  //Using aliases:
5  PType[ticket]
6
7  //Other examples
8  PosType[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna 0 (compra) ou 1 (venda).

Retorno:

int

Em caso de erro:

Retorna -1 se a posição não for encontrada.

Retorno:

int

PosSymbol

Esta função retorna o símbolo do ativo negociado em uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosSymbol[ticket]
3
4  //Using aliases:
5  PSymbol[ticket]
6
7  //Other examples
8  PosSymbol[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o símbolo do ativo.

Retorno:

string

Em caso de erro:

Retorna string vazia se a posição não for encontrada.

Retorno:

string

PosComment

Esta função retorna o comentário/texto descritivo associado a uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosComment[ticket]
3
4  //Using aliases:
5  PComment[ticket]
6
7  //Other examples
8  PosComment[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o comentário da posição.

Retorno:

string

Em caso de erro:

Retorna string vazia se a posição não for encontrada.

Retorno:

string

PosExternal

Esta função retorna o identificador externo (gerado pelo servidor) de uma posição identificada pelo ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição.

Exemplos

```
1  //Default example:
2  PosExternal[ticket]
3
4  //Using aliases:
5  PExternal[ticket]
6
7  //Other examples
8  PosExternal[PosTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ID externo da posição.

Retorno:
string

Em caso de erro:

Retorna string vazia se a posição não for encontrada.

Retorno:
string

OrderTicket

Esta função retorna o ticket de uma ordem aberta usando os parâmetros informados.

Parâmetros

int	index;		// Indice da ordem. Começando com 0 para a mais atual.
ENUM_TRADE	type	= TYPE_ALL;	// Tipo de trade: 'TYPE_BUY', 'TYPE_SELL', 'TYPE_ALL'.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
string	symbol	= REAL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  OrderTicket[index, type = TYPE_ALL, magic = MAGIC, symbol = REAL]
3
4  //Using aliases:
5  OTicket[index, type = TYPE_ALL, magic = MAGIC, symbol = REAL]
6
7  //Other examples
8  OrderTicket[0, TYPE_BUY] // Retorna o ticket da compra mais recente
```

Retornos

Em caso de sucesso:

Retorna o ticket da ordem.

Retorno:
ulong

Em caso de erro:

Retorna 0 se nenhuma ordem for encontrada com os parâmetros informados.

Retorno:
ulong

OrderDouble

Esta função retorna o valor de uma propriedade específica do tipo double (número com casas decimais) de uma ordem aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.
ENUM_ORDER_DOUBLE	property;		// Propriedade da ordem: 'TYPE_VOLUME' (volume/lotes), 'TYPE_OPEN' (preço de abertura).

Exemplos

```
1  //Default example:
2  OrderDouble[ticket, property]
3
4  //Other examples
5  OrderDouble[123456, TYPE_VOLUME] // Retorna o volume/lotes da ordem
6  OrderDouble[OrderTicket[0], TYPE_OPEN] // Retorna o preço de abertura da ordem
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
double

Em caso de erro:

Retorna 0.0 se a ordem não existir ou a propriedade for inválida.

Retorno:
double

OrderInt

Esta função retorna o valor de uma propriedade específica do tipo inteiro de uma ordem aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.
ENUM_ORDER_INT	property;		// Propriedade da ordem: 'TYPE_TIME_SETUP' (timestamp de abertura), 'TYPE_MAGIC' (número mágico).

Exemplos

```
1  //Default example:
2  OrderInt[ticket, property]
3
4  //Other examples
5  OrderInt[123456, TYPE_TIME_SETUP] // Retorna o timestamp de abertura da ordem
6  OrderInt[OrderTicket[0], TYPE_MAGIC] // Retorna o número mágico da ordem
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
int

Em caso de erro:

Retorna 0 se a ordem não existir ou a propriedade for inválida.

Retorno:
int

OrderString

Esta função retorna o valor de uma propriedade específica do tipo string (texto) de uma ordem aberta, identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.
ENUM_ORDER_STRING	property;		// Propriedade da ordem: 'TYPE_SYMBOL' (símbolo do ativo), 'TYPE_COMMENT' (comentário da operação).

Exemplos

```
1  //Default example:
2  OrderString[ticket, property]
3
4  //Other examples
5  OrderString[123456, TYPE_SYMBOL] // Retorna o símbolo do ativo da ordem
6  OrderString[OrderTicket[0], TYPE_COMMENT] // Retorna o comentário da ordem
```

Retornos

Em caso de sucesso:

Retorna o valor da propriedade solicitada.

Retorno:
string

Em caso de erro:

Retorna uma string vazia se a ordem não existir ou a propriedade for inválida.

Retorno:
string

OrderVolumeInit

Esta função retorna o volume inicial especificado quando a ordem pendente foi criada.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderVolumeInit[ticket]
3
4  //Using aliases:
5  OVolInit[ticket]
6
7  //Other examples
8  OrderVolumeInit[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o volume inicial da ordem.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

double

OrderVolume

Esta função retorna o volume atual (tamanho) de uma ordem, que pode ter sido modificado após a criação.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderVolume[ticket]
3
4  //Using aliases:
5  OVol[ticket]
6
7  //Other examples
8  OrderVolume[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o volume atual da ordem.

Retorno:
double

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:
double

OrderOpen

Esta função retorna o preço de abertura especificado para uma ordem pendente.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderOpen[ticket]
3
4  //Using aliases:
5  OO[ticket]
6
7  //Other examples
8  OrderOpen[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço de abertura da ordem.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

double

OrderStop

Esta função retorna o preço definido para o Stop Loss de uma ordem pendente.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderStop[ticket]
3
4  //Using aliases:
5  OSL[ticket]
6
7  //Other examples
8  OrderStop[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do Stop Loss.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não tiver Stop Loss definido ou não for encontrada.

Retorno:

double

OrderTake

Esta função retorna o preço definido para o Take Profit de uma ordem pendente.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderTake[ticket]
3
4  //Using aliases:
5  OTP[ticket]
6
7  //Other examples
8  OrderTake[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do Take Profit.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não tiver Take Profit definido ou não for encontrada.

Retorno:

double

OrderCurrent

Esta função retorna o preço atual (de mercado) para o ativo da ordem pendente.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderCurrent[ticket]
3
4  //Using aliases:
5  OCurrent[ticket]
6
7  //Other examples
8  OrderCurrent[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço atual do ativo da ordem.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

double

OrderStopLimit

Esta função retorna o preço de ativação especificado para ordens do tipo Stop Limit.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderStopLimit[ticket]
3
4  //Using aliases:
5  OSLLimit[ticket]
6
7  //Other examples
8  OrderStopLimit[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço de ativação da Stop Limit.

Retorno:

double

Em caso de erro:

Retorna 0 se a ordem não for do tipo Stop Limit ou não for encontrada.

Retorno:

double

OrderSetup

Esta função retorna o momento em que a ordem foi criada no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderSetup[ticket]
3
4  //Using aliases:
5  OSetup[ticket]
6
7  //Other examples
8  OrderSetup[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime de criação da ordem.

Retorno:

datetime

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

datetime

OrderType

Esta função retorna o tipo da ordem (compra/venda, stop/limit, etc) conforme enumerador de tipos de ordem.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderType[ticket]
3
4  //Using aliases:
5  OType[ticket]
6
7  //Other examples
8  OrderType[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do tipo de ordem. (0: Compra mercado, 1: Venda mercado, 2: Buy Limit, 3: Sell Limit, 4: Buy Stop, 5: Sell Stop, 6: Buy Stop Limit, 7: Sell Stop Limit, 8: Fechar por oposta)

Retorno:
int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:
int

OrderState

Esta função retorna o estado atual da ordem conforme enumerador de estados.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderState[ticket]
3
4  //Using aliases:
5  OState[ticket]
6
7  //Other examples
8  OrderState[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do estado da ordem. (0: Verificando, 1: Aceita, 2: Cancelada, 3: Parcial, 4: Executada, 5: Rejeitada, 6: Expirada, 7: Registrando, 8: Modificando, 9: Cancelando)

Retorno:
int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:
int

OrderTimeExpiration

Esta função retorna o momento em que a ordem pendente irá expirar (se aplicável) no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderTimeExpiration[ticket]
3
4  //Using aliases:
5  OTExpira[ticket]
6
7  //Other examples
8  OrderTimeExpiration[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime de expiração da ordem.

Retorno:
datetime

Em caso de erro:

Retorna 0 se a ordem não tiver expiração ou não for encontrada.

Retorno:
datetime

OrderDone

Esta função retorna o momento em que a ordem foi executada ou cancelada no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderDone[ticket]
3
4  //Using aliases:
5  ODone[ticket]
6
7  //Other examples
8  OrderDone[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime de execução/cancelamento.

Retorno:

datetime

Em caso de erro:

Retorna 0 se a ordem ainda estiver pendente ou não for encontrada.

Retorno:

datetime

OrderSetupMsc

Esta função retorna o momento em que a ordem foi criada no formato timestamp em milissegundos desde 01/01/1970.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderSetupMsc[ticket]
3
4  //Using aliases:
5  OSetupMsc[ticket]
6
7  //Other examples
8  OrderSetupMsc[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o timestamp em milissegundos.

Retorno:

long

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

long

OrderDoneMsc

Esta função retorna o momento em que a ordem foi executada ou cancelada no formato timestamp em milissegundos desde 01/01/1970.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderDoneMsc[ticket]
3
4  //Using aliases:
5  ODoneMsc[ticket]
6
7  //Other examples
8  OrderDoneMsc[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o timestamp em milissegundos.

Retorno:

long

Em caso de erro:

Retorna 0 se a ordem ainda estiver pendente ou não for encontrada.

Retorno:

long

OrderFilling

Esta função retorna o tipo de política de preenchimento (FOK, IOC, etc) conforme enumerador de tipos de preenchimento.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderFilling[ticket]
3
4  //Using aliases:
5  OFilling[ticket]
6
7  //Other examples
8  OrderFilling[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código da política de preenchimento. (0: FOK (tudo ou nada), 1: IOC (tudo/parcial), 2: BOC (só livro), 3: Return (parcial continua))

Retorno:
int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:
int

OrderTime

Esta função retorna o momento da última modificação da ordem no formato datetime.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderTime[ticket]
3
4  //Using aliases:
5  OT[ticket]
6
7  //Other examples
8  OrderTime[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o datetime da última modificação.

Retorno:

datetime

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

datetime

OrderMagic

Esta função retorna o número mágico (identificador personalizado do robô) associado a uma ordem.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderMagic[ticket]
3
4  //Using aliases:
5  OMagic[ticket]
6
7  //Other examples
8  OrderMagic[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o número mágico da ordem.

Retorno:
ulong

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:
ulong

OrderReason

Esta função retorna o código que indica o motivo pelo qual a ordem foi criada (manual, EA, etc).

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderReason[ticket]
3
4  //Using aliases:
5  OReason[ticket]
6
7  //Other examples
8  OrderReason[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do motivo de criação. (0: Desktop, 1: Mobile, 2: Web, 3: Expert, 4: Stop Loss, 5: Take Profit, 6: Stop Out)

Retorno:

int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

int

OrderId

Esta função retorna o identificador único da ordem no sistema de negociação.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderId[ticket]
3
4  //Using aliases:
5  OId[ticket]
6
7  //Other examples
8  OrderId[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ID da ordem.

Retorno:

int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

int

OrderById

Esta função retorna o número do ticket correspondente a um ID de ordem no sistema.

Parâmetros

ulong	ticket;		// ID da ordem no sistema.

Exemplos

```
1  //Default example:
2  OrderById[ticket]
3
4  //Using aliases:
5  OById[ticket]
6
7  //Other examples
8  OrderById[987654]
```

Retornos

Em caso de sucesso:

Retorna o número do ticket da ordem.

Retorno:

int

Em caso de erro:

Retorna 0 se a ordem não for encontrada.

Retorno:

int

OrderSymbol

Esta função retorna o símbolo do ativo negociado em uma ordem.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderSymbol[ticket]
3
4  //Using aliases:
5  OSymbol[ticket]
6
7  //Other examples
8  OrderSymbol[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o símbolo do ativo.

Retorno:

string

Em caso de erro:

Retorna string vazia se a ordem não for encontrada.

Retorno:

string

OrderComment

Esta função retorna o comentário/texto descritivo associado a uma ordem.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem.

Exemplos

```
1  //Default example:
2  OrderComment[ticket]
3
4  //Using aliases:
5  OComment[ticket]
6
7  //Other examples
8  OrderComment[OrderTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o comentário da ordem.

Retorno:

string

Em caso de erro:

Retorna string vazia se a ordem não for encontrada.

Retorno:

string

DealTicket

Esta função retorna o ticket de uma posição fechada usando os parâmetros informados.

Parâmetros

int	index;		// Indice da posição. Começando com 0 para a mais atual.
ENUM_TRADE	type	= TYPE_ALL;	// Tipo de trade: 'TYPE_BUY', 'TYPE_SELL', 'TYPE_ALL'
ENUM_MARKET_ACTION	actionType	= TYPE_ALL;	// Tipo de fechamento: 'TYPE_IN', 'TYPE_OUT', 'TYPE_ALL'.
ENUM_TIME_HISTORIC	dateType	= TYPE_DAY;	// Tipo de data de inicio: 'TYPE_DAY', 'TYPE_WEEK'
ENUM_MARKET_GET	profitType	= TYPE_ALL;	// Tipo de retorno de lucro: 'TYPE_POSITIVE', 'TYPE_NEGATIVE'
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.
string	symbol	= REAL;	// Símbolo do ativo.

Exemplos

```
1  //Default example:
2  DealTicket[index, type = TYPE_ALL, actionType = TYPE_ALL, dateType = TYPE_DAY, profitType =
3
4  //Using aliases:
5  DTicket[index, type = TYPE_ALL, actionType = TYPE_ALL, dateType = TYPE_DAY, profitType = TY
6
7  //Other examples
8  DealTicket[0, TYPE_BUY, TYPE_IN] // Retorna o ticket da última posição de compra (ordem de
9  DealTicket[0, TYPE_BUY, TYPE_OUT] // Retorna o ticket da última posição de compra (ordem qu
```

Retornos

Em caso de sucesso:

Retorna o ticket da posição.

Retorno:

ulong

Em caso de erro:

Retorna 0 se nenhuma posição for encontrada com os parâmetros informados.

Retorno:

ulong

DealProfit

Esta função retorna o valor do lucro ou prejuízo (em moeda do ativo) de um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealProfit[ticket]
3
4  //Using aliases:
5  DProfit[ticket]
6
7  //Other examples
8  DealProfit[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o valor do lucro/prejuízo do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealOpen

Esta função retorna o preço de abertura de um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealOpen[ticket]
3
4  //Using aliases:
5  DO[ticket]
6
7  //Other examples
8  DealOpen[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço de abertura do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealCommission

Esta função retorna o valor da comissão cobrada em um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1 //Default example:
2 DealCommission[ticket]
3
4 //Using aliases:
5 DComm[ticket]
6
7 //Other examples
8 DealCommission[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o valor da comissão do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealFee

Esta função retorna o valor da taxa cobrada em um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealFee[ticket]
3
4  //Using aliases:
5  DFee[ticket]
6
7  //Other examples
8  DealFee[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o valor da taxa do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealStop

Esta função retorna o preço do stop loss definido para um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealStop[ticket]
3
4  //Using aliases:
5  DSL[ticket]
6
7  //Other examples
8  DealStop[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do stop loss do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealTake

Esta função retorna o preço do take profit definido para um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealTake[ticket]
3
4  //Using aliases:
5  DTP[ticket]
6
7  //Other examples
8  DealTake[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o preço do take profit do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealSwap

Esta função retorna o valor do swap acumulado para um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealSwap[ticket]
3
4  //Using aliases:
5  DSwap[ticket]
6
7  //Other examples
8  DealSwap[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o valor do swap do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealVolume

Esta função retorna o volume negociado em um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealVolume[ticket]
3
4  //Using aliases:
5  DVol[ticket]
6
7  //Other examples
8  DealVolume[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o volume do negócio.

Retorno:

double

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

double

DealMagic

Esta função retorna o identificador mágico (número de referência) associado a um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealMagic[ticket]
3
4  //Using aliases:
5  DMagic[ticket]
6
7  //Other examples
8  DealMagic[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o número mágico do negócio.

Retorno:
ulong

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:
ulong

DealReason

Esta função retorna o código que indica o motivo pelo qual o negócio foi executado.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealReason[ticket]
3
4  //Using aliases:
5  DReason[ticket]
6
7  //Other examples
8  DealReason[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do motivo da execução. (0=Cliente, 1=Mobile, 2=Web, 3=Expert, 4=Stop Loss, 5=Take Profit, 6=Stop Out, 7=Rollover, 8=Margem, 9=Split, 10=Ação corporativa)

Retorno:

int

Em caso de erro:

Retorna -1 se o negócio não for encontrado.

Retorno:

int

DealOrder

Esta função retorna o número do ticket da ordem que gerou o negócio.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealOrder[ticket]
3
4  //Using aliases:
5  DOrder[ticket]
6
7  //Other examples
8  DealOrder[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ticket da ordem associada.

Retorno:

long

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

long

DealId

Esta função retorna o identificador único (ID) de um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealId[ticket]
3
4  //Using aliases:
5  DId[ticket]
6
7  //Other examples
8  DealId[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ID do negócio.

Retorno:

long

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

long

DealEntry

Esta função retorna um código que indica o tipo de entrada de um negócio.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealEntry[ticket]
3
4  //Using aliases:
5  DEntry[ticket]
6
7  //Other examples
8  DealEntry[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do tipo de entrada. (0=Entrada, 1=Saída, 2=Reversão, 3=Fechamento por oposta)

Retorno:

int

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

int

DealOut

Esta função indica se o negócio atual representa uma saída no sistema.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealOut[ticket]
3
4  //Using aliases:
5  DOut[ticket]
6
7  //Other examples
8  DealOut[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna true se o negócio for uma saída.

Retorno:

bool

Em caso de erro:

Retorna false se o negócio for uma entrada ou se o negócio não for encontrado.

Retorno:

bool

DealIn

Esta função indica se o negócio atual representa uma posição de entrada no sistema.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealIn[ticket]
3
4  //Using aliases:
5  DIn[ticket]
6
7  //Other examples
8  DealIn[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna true se o negócio for uma entrada.

Retorno:

bool

Em caso de erro:

Retorna false se o negócio for uma saída ou se o negócio não for encontrado.

Retorno:

bool

DealTime

Esta função retorna a data e hora (em formato datetime) quando o negócio foi executado.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealTime[ticket]
3
4  //Using aliases:
5  DT[ticket]
6
7  //Other examples
8  DealTime[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna a data/hora da execução do negócio.

Retorno:

datetime

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

datetime

DealTimeMsc

Esta função retorna o tempo de execução de negociações em milissegundos desde 01.01.1970

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealTimeMsc[ticket]
3
4  //Using aliases:
5  DTMsc[ticket]
6
7  //Other examples
8  DealTimeMsc[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna a data/hora em milissegundos da execução do negócio.

Retorno:

long

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:

long

DealType

Esta função retorna o tipo do negócio.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealType[ticket]
3
4  //Using aliases:
5  DType[ticket]
6
7  //Other examples
8  DealType[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o código do tipo de negócio. (0=Compra, 1=Venda, 2=Saldo, 3=Crédito, 4=Cobrança extra, 5=Correção, 6=Bônus, 7=Comissão extra, 8=Comissão diária, 9=Comissão mensal, 10=Comissão diária do agente, 11=Comissão mensal do agente, 12=Juros, 13=Compra cancelada, 14=Venda cancelada, 15=Dividendo, 16=Dividendo isento, 17=Imposto)

Retorno:
uint

Em caso de erro:

Retorna 0 se o negócio não for encontrado.

Retorno:
uint

DealComment

Esta função retorna o comentário/texto descritivo associado a um negócio específico.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealComment[ticket]
3
4  //Using aliases:
5  DComment[ticket]
6
7  //Other examples
8  DealComment[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o comentário do negócio.

Retorno:

string

Em caso de erro:

Retorna string vazia se o negócio não for encontrado.

Retorno:

string

DealExternal

Esta função retorna identificador de negócios em um sistema de negociação externo (na troca).

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealExternal[ticket]
3
4  //Using aliases:
5  DExternal[ticket]
6
7  //Other examples
8  DealExternal[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o ID externo do negócio.

Retorno:

string

Em caso de erro:

Retorna string vazia se o negócio não for encontrado.

Retorno:

string

DealSymbol

Esta função retorna o nome do símbolo no qual o negócio foi executado.

Parâmetros

long	ticket;		// Número do ticket que identifica o negócio.

Exemplos

```
1  //Default example:
2  DealSymbol[ticket]
3
4  //Using aliases:
5  DSymbol[ticket]
6
7  //Other examples
8  DealSymbol[DealTicket[0]]
```

Retornos

Em caso de sucesso:

Retorna o símbolo do negócio.

Retorno:

string

Em caso de erro:

Retorna string vazia se o negócio não for encontrado.

Retorno:

string

BuyAuto

Esta função abre uma posição de compra no mercado, com opções de take profit, stop loss e comentário. O tipo de ordem (compra imediata ou pendente) é definido automaticamente.

Parâmetros

double	price;		// Preço de entrada para a ordem de compra.
double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  BuyAuto[price, volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol =
3
4  //Other examples
5  BuyAuto[ASK, 0.1, 100, 50] // Compra 0.1 lote no preço atual com TP 100 pontos e SL 50 pont
6  BuyAuto[ASK, 0.5] // Compra 0.5 lote sem TP/SL
7  BuyAuto[ASK, 0.5, TickToPoint[20], TickToPoint[20]] // Compra 0.5 lote com TP/SL em 20 tick
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:

bool

SellAuto

Esta função abre uma posição de venda no mercado, com opções de take profit, stop loss e comentário. O tipo de ordem (venda imediata ou pendente) é definido automaticamente.

Parâmetros

double	price;		// Preço de entrada para a ordem de venda.
double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  SellAuto[price, volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol
3
4  //Other examples
5  SellAuto[BID, 0.1, 100, 50] // Venda 0.1 lote no preço atual com TP 100 pontos e SL 50 pont
6  SellAuto[BID, 0.5] // Venda 0.5 lote sem TP/SL
7  SellAuto[BID, 0.5, TickToPoint[20], TickToPoint[20]] // Venda 0.5 lote com TP/SL em 20 tick
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:

bool

Buy

Esta função abre uma posição de compra no preço de mercado atual (Ask), com parâmetros opcionais de take profit, stop loss e comentário personalizado.

Parâmetros

double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  Buy[volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol = REAL]
3
4  //Other examples
5  Buy[0.1, 100, 50] // Compra 0.1 lote no preço de mercado com TP 100 pontos e SL 50 pontos
6  Buy[0.5] // Compra 0.5 lote sem TP/SL
7  Buy[0.5, TickToPoint[20], TickToPoint[20]] // Compra 0.5 lote com TP/SL em 20 ticks
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:
bool

Sell

Esta função abre uma posição de venda no preço de mercado atual (Bid), com parâmetros opcionais de take profit, stop loss e comentário personalizado.

Parâmetros

double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  Sell[volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol = REAL]
3
4  //Other examples
5  Sell[0.1, 100, 50] // Vende 0.1 lote no preço de mercado com TP 100 pontos e SL 50 pontos
6  Sell[0.5] // Vende 0.5 lote sem TP/SL
7  Sell[0.5, TickToPoint[20], TickToPoint[20]] // Vende 0.5 lote com TP/SL em 20 ticks
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:
bool

BuyOrder

Esta função coloca uma ordem pendente de compra (Buy Limit/Stop) com parâmetros opcionais de take profit, stop loss e comentário personalizado.

Parâmetros

double	price;		// Preço de ativação para a ordem pendente.
double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  BuyOrder[price, volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol
3
4  //Other examples
5  BuyOrder[1.12000, 0.1, 100, 50] // Ordem pendente de compra em 1.12000 com TP 100 pontos e
6  BuyOrder[1.11500, 0.5] // Ordem pendente de compra em 1.11500 sem TP/SL
7  BuyOrder[1.11500, 0.5, TickToPoint[20], TickToPoint[20]] // Ordem pendente de compra em 1.1
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:

bool

SellOrder

Esta função coloca uma ordem pendente de venda (Sell Limit/Stop) com parâmetros opcionais de take profit, stop loss e comentário personalizado.

Parâmetros

double	price;		// Preço de ativação para a ordem pendente.
double	volume;		// Volume/lote da operação.
int	tpPoints	= 0;	// Pontos para o take profit (0 para desativar).
int	slPoints	= 0;	// Pontos para o stop loss (0 para desativar).
string	comment	= "SB-PlaceByFunction";	// Comentário associado à ordem.
string	symbol	= REAL;	// Símbolo do ativo (vazio para usar o símbolo atual).

Exemplos

```
1  //Default example:
2  SellOrder[price, volume, tpPoints = 0, slPoints = 0, comment = "SB-PlaceByFunction", symbol
3
4  //Other examples
5  SellOrder[1.13000, 0.1, 100, 50] // Ordem pendente de venda em 1.13000 com TP 100 pontos e
6  SellOrder[1.13500, 0.5] // Ordem pendente de venda em 1.13500 sem TP/SL
7  SellOrder[1.13500, 0.5, TickToPoint[20], TickToPoint[20]] // Ordem pendente de venda em 1.1
```

Retornos

Em caso de sucesso:

Retorna true se a ordem for executada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna false se a ordem não for executada com sucesso.

Retorno:

bool

PosClose

Esta função fecha completamente uma posição aberta no mercado, identificada pelo seu ticket. O fechamento é feito ao preço atual de mercado.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser fechada.

Exemplos

```
1  //Default example:
2  PosClose[ticket]
3
4  //Other examples
5  PosClose[PosTicket[0]] // Fecha completamente a ultima posição aberta
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a posição for fechada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição não for encontrada ou não puder ser fechada.

Retorno:
bool

PosClosePartial

Esta função fecha parcialmente uma posição aberta no mercado, reduzindo seu volume pelo valor especificado.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser parcialmente fechada.
double	volume;		// Volume/lote a ser fechado.

Exemplos

```
1  //Default example:
2  PosClosePartial[ticket, volume]
3
4  //Other examples
5  PosClosePartial[PosTicket[0], 0.5] // Fecha 0.5 lotes da ultima posição aberta
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a posição for parcialmente fechada com sucesso.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição não for encontrada, o volume for inválido ou não puder ser fechada.

Retorno:
bool

OrderClose

Esta função cancela uma ordem pendente (limit/stop) identificada pelo seu ticket.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem pendente a ser cancelada.

Exemplos

```
1  //Default example:
2  OrderClose[ticket]
3
4  //Other examples
5  OrderClose[OrderTicket[0]] // Cancela a ultima ordem pendente
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a ordem for cancelada com sucesso.

Retorno:

bool

Em caso de erro:

Retorna falso (false) se a ordem não for encontrada ou não puder ser cancelada.

Retorno:

bool

PosCloseAll

Esta função fecha todas as posições abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  PosCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  PosCloseAll[] // Fecha todas as posições abertas em todos os símbolos
6  PosCloseAll[EURUSD] // Fecha todas as posições abertas somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de posições fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma posição for fechada.

Retorno:
int

PosBuyCloseAll

Esta função fecha todas as posições de compra (Buy) abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  PosBuyCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  PosBuyCloseAll[] // Fecha todas as posições de compra em todos os símbolos
6  PosBuyCloseAll[EURUSD] // Fecha todas as posições de compra somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de posições de compra fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma posição de compra for fechada.

Retorno:
int

PosSellCloseAll

Esta função fecha todas as posições de venda (Sell) abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  PosSellCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  PosSellCloseAll[] // Fecha todas as posições de venda em todos os símbolos
6  PosSellCloseAll[EURUSD] // Fecha todas as posições de venda somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de posições de venda fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma posição de venda for fechada.

Retorno:
int

OrderCloseAll

Esta função fecha todas as ordens abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  OrderCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  OrderCloseAll[] // Fecha todas as ordens abertas em todos os símbolos
6  OrderCloseAll[EURUSD] // Fecha todas as ordens abertas somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de ordens fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma ordem for fechada.

Retorno:
int

OrderBuyCloseAll

Esta função fecha todas as ordens de compra (Buy) abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  OrderBuyCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  OrderBuyCloseAll[] // Fecha todas as ordens de compra em todos os símbolos
6  OrderBuyCloseAll[EURUSD] // Fecha todas as ordens de compra somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de ordens de compra fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma ordem de compra for fechada.

Retorno:
int

OrderSellCloseAll

Esta função fecha todas as ordens de venda (Sell) abertas no mercado, podendo ser filtradas por símbolo.

Parâmetros

string	symbol	= REAL;	// Símbolo do ativo.
ulong	magic	= MAGIC;	// Número mágico para filtrar as operações.

Exemplos

```
1  //Default example:
2  OrderSellCloseAll[symbol = REAL, magic = MAGIC]
3
4  //Other examples
5  OrderSellCloseAll[] // Fecha todas as ordens de venda em todos os símbolos
6  OrderSellCloseAll[EURUSD] // Fecha todas as ordens de venda somente no EURUSD
```

Retornos

Em caso de sucesso:

Retorna o número de ordens de venda fechadas com sucesso.

Retorno:
int

Em caso de erro:

Retorna 0 se nenhuma ordem de venda for fechada.

Retorno:
int

PosModify

Esta função permite alterar os níveis de take profit e/ou stop loss de uma posição aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take. (0= ignora).
double	stop	= 0;	// Valor do novo stop loss. (0= ignora).

Exemplos

```
1  //Default example:
2  PosModify[ticket, method, take = 0, stop = 0]
3
4  //Other examples
5  PosModify[123456, TYPE_PRICE, 1.12500, 1.11500] // Altera TP para 1.12500 e SL para 1.11500
6  PosModify[123456, TYPE_POINT, 300, 150] // Altera TP para +300 pontos e SL para -150 pontos
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição não for encontrada ou a modificação falhar.

Retorno:
bool

PosModifyTake

Esta função permite alterar somente o nível de take profit de uma posição aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take profit (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  PosModifyTake[ticket, method, take = 0]
3
4  //Other examples
5  PosModifyTake[123456, TYPE_PRICE, 1.12500] // Altera TP para 1.12500 (valor absoluto)
6  PosModifyTake[123456, TYPE_POINT, 300] // Altera TP para +300 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição não for encontrada ou a modificação falhar.

Retorno:
bool

PosModifyStop

Esta função permite alterar somente o nível de stop loss de uma posição aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	stop	= 0;	// Valor do novo stop loss (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  PosModifyStop[ticket, method, stop = 0]
3
4  //Other examples
5  PosModifyStop[123456, TYPE_PRICE, 1.11500] // Altera SL para 1.11500 (valor absoluto)
6  PosModifyStop[123456, TYPE_POINT, 150] // Altera SL para -150 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição não for encontrada ou a modificação falhar.

Retorno:
bool

OrderModify

Esta função permite alterar os níveis de take profit e/ou stop loss de uma ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take. (0= ignora).
double	stop	= 0;	// Valor do novo stop loss. (0= ignora).
double	price	= 0;	// Valor do novo preço. (0= ignora).

Exemplos

```
1  //Default example:
2  OrderModify[ticket, method, take = 0, stop = 0, price = 0]
3
4  //Other examples
5  OrderModify[123456, TYPE_PRICE, 1.12500, 1.11500] // Altera TP para 1.12500 e SL para 1.115
6  OrderModify[123456, TYPE_POINT, 300, 150] // Altera TP para +300 pontos e SL para -150 pont
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a ordem não for encontrada ou a modificação falhar.

Retorno:

bool

OrderModifyTake

Esta função permite alterar somente o nível de take profit de uma ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take profit (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  OrderModifyTake[ticket, method, take = 0]
3
4  //Other examples
5  OrderModifyTake[123456, TYPE_PRICE, 1.12500] // Altera TP para 1.12500 (valor absoluto)
6  OrderModifyTake[123456, TYPE_POINT, 300] // Altera TP para +300 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a ordem não for encontrada ou a modificação falhar.

Retorno:
bool

OrderModifyStop

Esta função permite alterar somente o nível de stop loss de uma ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	stop	= 0;	// Valor do novo stop loss (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  OrderModifyStop[ticket, method, stop = 0]
3
4  //Other examples
5  OrderModifyStop[123456, TYPE_PRICE, 1.11500] // Altera SL para 1.11500 (valor absoluto)
6  OrderModifyStop[123456, TYPE_POINT, 150] // Altera SL para -150 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a ordem não for encontrada ou a modificação falhar.

Retorno:
bool

OrderModifyPrice

Esta função permite alterar somente o preço de uma ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a ordem a ser modificada.
double	price;		// Valor do novo preço.

Exemplos

```
1  //Default example:
2  OrderModifyPrice[ticket, price]
3
4  //Other examples
5  OrderModifyPrice[123456, 1.11500] // Altera preço para 1.11500
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a ordem não for encontrada ou a modificação falhar.

Retorno:
bool

Modify

Esta função permite alterar os níveis de take profit e/ou stop loss de uma posição ou ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take. (0= ignora).
double	stop	= 0;	// Valor do novo stop loss. (0= ignora).

Exemplos

```
1  //Default example:
2  Modify[ticket, method, take = 0, stop = 0]
3
4  //Other examples
5  Modify[123456, TYPE_PRICE, 1.12500, 1.11500] // Altera TP para 1.12500 e SL para 1.11500 (v
6  Modify[123456, TYPE_POINT, 300, 150] // Altera TP para +300 pontos e SL para -150 pontos do
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição ou ordem não for encontrada ou a modificação falhar.

Retorno:
bool

ModifyTake

Esta função permite alterar somente o nível de take profit de uma posição ou ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	take	= 0;	// Valor do novo take profit (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  ModifyTake[ticket, method, take = 0]
3
4  //Other examples
5  ModifyTake[123456, TYPE_PRICE, 1.12500] // Altera TP para 1.12500 (valor absoluto)
6  ModifyTake[123456, TYPE_POINT, 300] // Altera TP para +300 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição ou ordem não for encontrada ou a modificação falhar.

Retorno:
bool

ModifyStop

Esta função permite alterar somente o nível de stop loss de uma posição ou ordem aberta existente, utilizando diferentes métodos de cálculo.

Parâmetros

ulong	ticket;		// Número do ticket que identifica a posição a ser modificada.
ENUM_TRADE_MODIFY	method;		// Método de cálculo: 'TYPE_PRICE', 'TYPE_POINT'.
double	stop	= 0;	// Valor do novo stop loss (0 mantém inalterado).

Exemplos

```
1  //Default example:
2  ModifyStop[ticket, method, stop = 0]
3
4  //Other examples
5  ModifyStop[123456, TYPE_PRICE, 1.11500] // Altera SL para 1.11500 (valor absoluto)
6  ModifyStop[123456, TYPE_POINT, 150] // Altera SL para -150 pontos do preço de abertura
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) se a posição ou ordem não for encontrada ou a modificação falhar.

Retorno:
bool

ObjHLine

Permite desenhar uma linha horizontal em um preço específico no gráfico, com personalização de cor, espessura e estilo de linha.

Parâmetros

string	name;		// Identificador único da linha no gráfico.
double	price;		// Preço em que a linha horizontal será desenhada.
color	clr	= clrRed;	// Cor da linha.
int	width	= 1;	// Espessura da linha.
ENUM_LINE_STYLE	style	= TYPE_SOLID;	// Estilo da linha (sólida, tracejada, etc.).

Exemplos

```
1  //Default example:
2  ObjHLine[name, price, clr = clrRed, width = 1, style = TYPE_SOLID]
3
4  //Using aliases:
5  ObjHL[name, price, clr = clrRed, width = 1, style = TYPE_SOLID]
6
7  //Other examples
8  ObjHLine["LinhaSL", 1.11500] // Cria linha vermelha sólida no preço 1.11500
9  ObjHL["LinhaTP", 1.12000, clrBlue, 2, TYPE_DASH] // Linha azul tracejada mais espessa
```

Retornos

Em caso de sucesso:

Retorna o preço da linha criada.

Retorno:
double

Em caso de erro:

Retorna 0.0 caso a criação falhe.

Retorno:

double

ObjVLine

Permite desenhar uma linha vertical em um ponto de tempo específico no gráfico, com personalização de cor, espessura e estilo de linha.

Parâmetros

string	name;		// Identificador único da linha no gráfico.
datetime	value;		// Data e hora em que a linha vertical será desenhada.
color	clr	= clrRed;	// Cor da linha.
int	width	= 1;	// Espessura da linha.
ENUM_LINE_STYLE	style	= TYPE_SOLID;	// Estilo da linha (sólida, tracejada, etc.).

Exemplos

```
1  //Default example:
2  ObjVLine[name, value, clr = clrRed, width = 1, style = TYPE_SOLID]
3
4  //Using aliases:
5  ObjVL[name, value, clr = clrRed, width = 1, style = TYPE_SOLID]
6
7  //Other examples
8  ObjVLine["InicioSessao", TIME_CURRENT] // Linha vermelha no início da sessão
9  ObjVLine["Fechamento", TIME_CURRENT, clrBlue, 2, TYPE_DOT] // Linha azul pontilhada na hora
```

Retornos

Em caso de sucesso:

Retorna o tempo associado à linha criada.

Retorno:
datetime

Em caso de erro:

Retorna 0 em caso de falha.

Retorno:

datetime

ObjTrendLine

Permite desenhar uma linha de tendência conectando dois pontos de preço em tempos diferentes, com personalização de cor, espessura e estilo.

Parâmetros

string	name;		// Identificador único da linha de tendência.
datetime	time1;		// Tempo do ponto inicial.
double	price1;		// Preço do ponto inicial.
datetime	time2;		// Tempo do ponto final.
double	price2;		// Preço do ponto final.
color	clr	= clrRed;	// Cor da linha.
int	width	= 1;	// Espessura da linha.
ENUM_LINE_STYLE	style	= TYPE_SOLID;	// Estilo da linha.

Exemplos

```
1  //Default example:
2  ObjTrendLine[name, time1, price1, time2, price2, clr = clrRed, width = 1, style = TYPE_SO
3
4  //Using aliases:
5  ObjTL[name, time1, price1, time2, price2, clr = clrRed, width = 1, style = TYPE_SOLID]
6
7  //Other examples
8  ObjTrendLine["Suporte", Time[10], Low[10], Time[0], Low[0]] // Linha de suporte conectand
9  ObjTrendLine["Resistencia", Time[20], High[20], Time[0], High[0], clrBlue, 2, TYPE_DASH]
```

Retornos

Em caso de sucesso:

Retorna o preço atual da linha criada.

Retorno:

double

Em caso de erro:

Retorna 0.0 caso a criação falhe.

Retorno:
double

ObjRemove

Permite remover um objeto gráfico previamente criado no gráfico, identificado pelo seu nome.

Parâmetros

string	name;		// Nome do objeto a ser removido.

Exemplos

```
1  //Default example:
2  ObjRemove[name]
3
4  //Using aliases:
5  ObjRem[name]
6
7  //Other examples
8  ObjRemove['LinhaSL'] // Remove a linha chamada LinhaSL
9  ObjRemove['Resistencia'] // Remove objeto chamado Resistencia
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se o objeto foi removido com sucesso.

Retorno:

bool

Em caso de erro:

Retorna falso (false) se o objeto não existir ou a remoção falhar.

Retorno:

bool

ObjClear

Remove todos os objetos cujo nome inicie com um prefixo específico e opcionalmente por tipo.

Parâmetros

string	prefix;		// Prefixo dos objetos a serem removidos.
int	type	= -1;	// Tipo de objeto a remover (-1 remove todos os tipos).

Exemplos

```
1  //Default example:
2  ObjClear[prefix, type = -1]
3
4  //Using aliases:
5  ObjClr[prefix, type = -1]
6
7  //Other examples
8  ObjClear["Linha"] // Remove todos os objetos que começam com "Linha"
9  ObjClear["Sinal", OBJ_TREND] // Remove somente objetos de tendência com prefixo "Sinal"
```

Retornos

Em caso de sucesso:

Retorna o número de objetos removidos.

Retorno:

int

Em caso de erro:

Retorna 0 se nenhum objeto foi removido.

Retorno:

int

ObjExist

Permite verificar a existência de um objeto no gráfico através do seu nome.

Parâmetros

string	name;		// Nome do objeto a ser verificado.

Exemplos

```
1  //Default example:
2  ObjExist[name]
3
4  //Using aliases:
5  ObjEx[name]
6
7  //Other examples
8  ObjExist["LinhaSL"] // Retorna true se a linha existir", "ObjExist["Resistencia"] // Retorn
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se o objeto existe.

Retorno:

bool

Em caso de erro:

Retorna falso (false) se o objeto não existir.

Retorno:

bool

ObjGetTime

Permite recuperar a coordenada de tempo de um ponto de um objeto gráfico, especificando índice e buffer.

Parâmetros

string	name;		// Nome do objeto.
int	index	= -1;	// Índice do ponto (valor negativo indica o valor de criação).
int	buffer	= 0;	// Número do buffer a ser acessado.

Exemplos

```
1  //Default example:
2  ObjGetTime[name, index = -1, buffer = 0]
3
4  //Using aliases:
5  ObjGT[name, index = -1, buffer = 0]
6
7  //Other examples
8  ObjGetTime["LinhaSL"] // Obtém o tempo associado à linha
9  ObjGetTime["Suporte", 0] // Obtém o tempo do primeiro ponto do objeto Suporte
10 ObjGetTime["Suporte", 1] // Obtém o tempo do segundo ponto do objeto Suporte
```

Retornos

Em caso de sucesso:

Retorna o valor datetime correspondente ao ponto do objeto.

Retorno:
datetime

Em caso de erro:

Retorna 0 caso não encontre o objeto.

Retorno:
datetime

ObjGetPrice

Permite recuperar a coordenada de preço de um ponto de um objeto gráfico, especificando índice e buffer.

Parâmetros

string	name;		// Nome do objeto.
int	index	= -1;	// Índice do ponto (valor negativo indica o valor de criação).
int	buffer	= 0;	// Número do buffer a ser acessado.

Exemplos

```
1  //Default example:
2  ObjGetPrice[name, index = -1, buffer = 0]
3
4  //Using aliases:
5  ObjGP[name, index = -1, buffer = 0]
6
7  //Other examples
8  ObjGetPrice["LinhaSL"] // Obtém o preço associado à linha
9  ObjSetTime["Resistencia", 0] // Obtém o preço do primeiro ponto do objeto Resistencia
10 ObjSetTime["Resistencia", 1] // Obtém o preço do segundo ponto do objeto Resistencia
```

Retornos

Em caso de sucesso:

Retorna o valor double correspondente ao ponto do objeto.

Retorno:
double

Em caso de erro:

Retorna 0.0 caso não encontre o objeto.

Retorno:
double

ObjSetTime

Permite modificar a coordenada de tempo de um ponto específico de um objeto gráfico.

Parâmetros

string	name;		// Nome do objeto.
datetime	time;		// Novo valor de tempo para o ponto do objeto.
int	buffer	= 0;	// Número do buffer a ser modificado.

Exemplos

```
1  //Default example:
2  ObjSetTime[name, time, buffer = 0]
3
4  //Using aliases:
5  ObjST[name, time, buffer = 0]
6
7  //Other examples
8  ObjSetTime["LinhaSL", TIME_CURRENT] // Define o tempo atual para a linha
9  ObjSetTime["LinhaSL", TIME_CURRENT, 0] // Ajusta o tempo do primeiro ponto ou linha
10 ObjSetTime["Suporte", TIME_CURRENT, 1] // Ajusta o tempo do segundo ponto ou linha
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:
bool

Em caso de erro:

Retorna falso (false) caso falhe.

Retorno:
bool

ObjSetPrice

Permite modificar a coordenada de preço de um ponto específico de um objeto gráfico.

Parâmetros

string	name;		// Nome do objeto.
double	price;		// Novo valor de preço para o ponto do objeto.
int	buffer	= 0;	// Número do buffer a ser modificado.

Exemplos

```
1  //Default example:
2  ObjSetPrice[name, price, buffer = 0]
3
4  //Using aliases:
5  ObjSP[name, price, buffer = 0]
6
7  //Other examples
8  ObjSetPrice["LinhaSL", 1.11500] // Define novo preço para a linha
9  ObjSetPrice["Resistencia", 1.12000, 0] // Ajusta o preço do primeiro ponto ou linha
10 ObjSetPrice["Resistencia", 1.12000, 1] // Ajusta o preço do segundo ponto ou linha
```

Retornos

Em caso de sucesso:

Retorna verdadeiro (true) se a modificação for bem-sucedida.

Retorno:

bool

Em caso de erro:

Retorna falso (false) caso falhe.

Retorno:

bool