

Bem vindo ao curso de MQL4.

Nestas páginas, tentaremos esclarecer e entender o místico e o confuso sobre o MQL4, mostrando-lhes as explicações detalhadas e exemplos comentados.

Nesta série de capítulos, mostrarei como você pode utilizar a linguagem MQL4 para criar seus Consultores especialistas (Experts Advisors ou simplesmente EA), Indicadores e Scripts.

Caso você seja um programador, ou conheça, a linguagem C ou C++, indubitavelmente você conhece uma grande parte de MQL4, antes mesmo de começarmos qualquer uma das lições deste curso. Caso você nunca tenha escrito nenhum programa em nenhuma linguagem computacional, não se preocupe, pois, eu guiarei você, a fim de entender os conceitos de linguagem de programação em geral.

---

## Agora, que tal começarmos pelo básico?

MQL4? Como, porque e onde?

MQL4 é o mnemônico formado a partir da frase "**MetaQuotes Language 4**". MetaQuotes é a companhia que desenvolveu a plataforma para trading MetaTrader. E para ser mais robusta e forte que outras plataformas de negociação ela decidiu acrescentar uma linguagem de programação, onde o usuário (você) poderia escrever suas próprias estratégias.

A linguagem pode ser usada por você para criar um dos seguintes tipos de programas :

- Custom Indicators: são programas com os quais você usa funções e outros indicadores técnicos, para gerar e mostrar informações nos gráficos de trabalho, informações estas que possibilitam você a tomar decisões sobre o mercado. Aqui, você não pode abrir nem controlar suas ordens de trabalho. A cada variação de cota do mercado, a função principal do indicador é chamada para que você atualize o sistema de acordo com os novos valores.
- Experts Advisors: são programas com os quais você automatiza seus sistemas de negociação. Por exemplo, com ele você pode automatizar a abertura de suas ordens de compra/venda, stops e inclusive administrar seus riscos. A cada variação de cota do mercado, a função principal do Expert é chamada para que você analise e tome as decisões sobre suas ordens em relação ao mercado no momento em que estes valores são atualizados.
- Scripts: são programas onde você automatiza seqüências de trabalho que normalmente fariam manualmente no sistema. Diferentemente dos Custom Indicators e Expert Advisors os Scripts são executados somente uma vez (sob demanda) e não cada vez que uma cota muda. E, naturalmente, não pode acessar funções que trabalham com indicadores.

Até agora nos vimos "O que é o MQL4" e "Porque usar o MQL4"

Agora vamos ver "Onde usar o MQL4".

Para escrever seu código em MQL4, como qualquer coisa no mundo, você necessita escolher um destes dois caminhos : "Caminho Fácil" ou o "Caminho Difícil".

---

## O caminho Difícil

Pelo caminho difícil, você utiliza o seu editor de textos favorito e utiliza a linha de comandos (command prompt) para compilar seu programa. O NotePad não é uma má escolha como editor de textos para sua programação, mas não esqueça do seguinte:

- Ao salvar seu texto (código) você deve usar o formato texto simples (sem formatações)
- O arquivo deve preferencialmente ser salvo com a extensão .mq4 (isto torna mais fácil abri-lo no editor de textos do MetaTrader, o MetaEditor). Porém você pode utilizar qualquer extensão que quiser.

Depois de salvar o programa você necessita seguir alguns passos extras para deixar seu programa pronto para ser utilizado. Estes são os passos de compilação. Compilar é a ação de transformar o programa que você escreveu, na linguagem que você entende, para uma linguagem que o computador possa executar, ou a chamada linguagem de máquina. MetaTrader possui um programa que é utilizado para compilar o programa que você escreveu, este programa se chama MetaLang.Exe.

MetaLang.exe é um programa que possui 2 parametros de entrada, e que como saída grava um programa com a extensão .ex4 (arquivo este que o MetaTrader entende).

- O primeiro parâmetro são as opções, e a única opção disponível é -q (quit).
- O segundo parâmetro é o nome de seu arquivo fonte com o código que você programou, a este nome você deve juntar ao caminho completo do diretório onde ele se encontra (a localização exata do arquivo em seu computador).

Basicamente a sintaxe do compilador por linha de comando possui este modelo

metalang | opções | NomeDoArquivo

Vamos a um exemplo para o melhor entendimento

- 1) busque onde se encontra o arquivo MetaLang.exe. Geralmente ele se encontra no diretório no qual voce instalou o MetaTrader (No meu caso em "C:\Archivos de programa\Forex\Interbank FX")
- 2) Crie um arquivo de lote (Bat) com o nome Compile.Bat (ou qualquer outro nome que voce quiser)
- 3) Dentro deste arquivo escreva as seguintes linhas (Não esqueça de modificar o caminho de acordo com sua localização do MetaTrader

```
C:
CD "C:\Archivos de programa\Forex\Interbank FX\Experts\Scripts"
..\..\MetaLang -q MeuPrimeiroScript.mq4
```

- 4) Execute o Arquivo bat a partir da linha e comando
  - Menu Iniciar -> Executar
  - Na caixa de texto da janela digite "cmd.exe" (sem as aspas)
  - Clique em executar
  - Na nova janela, digite o caminho + nome do arquivo bat, eu gravei ele no diretório C:\SistemasMT e teclou enter
  - Você deve obter uma tela parecida com a de abaixo

## IMAGEM

- Após isso você obtém o arquivo MeuPrimeiroScript.mq4
- digite exit e teclou enter na janela de comandos para encerrar o trabalho

---

## O caminho Fácil

Para facilitar nossa vida e evitar esta "perda de tempo" o MetaTrader disponibilizou uma ótima IDE (Integrated Development Editor ou Editor de desenvolvimento integrado) chamado MetaEditor, que possui estas qualidades

- Editor de texto com identificação de palavras chaves e símbolos por diferentes cores que aparecem enquanto você esta construindo ou digitando seu código. Isto facilita muito a vida pois evita que você use erroneamente palavras chaves durante o processo de desenvolvimento.

- Sistema de desenvolvimento com tecnologia MDI (Multi Document Interface ou Interface de documentos múltiplos) isso significa que você pode ter vários arquivos abertos ao mesmo tempo em seu editor.

- Seu programa é facilmente compilado. Simplesmente teclou F5 quando o código que você quer compilar seja o documento atualmente ativo, isso fará que o editor faça todo o trabalho do caminho difícil para você e disponibilize em um só toque de tecla o arquivo .mq4 que estará pronto para uso (lógico, desde que o arquivo não contenha nenhum erro de semântica ou outro qualquer identificado pelo compilador), caso haja algum erro o mesmo será indicado e apontado na janela de dialogo do

editor, o que possibilita um acesso mais rápido e fácil a linha onde ocorreu o referido erro

- Acesso rápido ao sistema de ajuda, basta colocar o cursor em cima de alguma palavra no seu código, caso esta palavra seja identificada pelo editor como parte integrante da base de MQL4 automaticamente o arquivo de ajuda trará o texto referente aquela palavra.

- abaixo a aparência que tem o MetaEditor, para que você já vá se familiarizando

IMAGEM

# Meta Editor

---

## Acessando o MetaEditor

Não basta você ter um linguagem de programação a sua disposição, para fazer o que você necessita para realizar sua análise ou suas negociações. Muito mais que isso, você necessita facilidades e flexibilidades para utilizar sua linguagem. MQL4 não estaria completa se não fornecesse uma ferramenta capaz de dar a você um mínimo de comodidade e legibilidade a seu trabalho. Para isso existe o MetaEditor. Ele nada mais é que um editor de textos avançados, capaz de identificar para você os vários aspectos de MQL4, bem como, tornar o processo de compilação de um código seu, muito mais fácil e cômodo.

Para chamar o Meta Editor, você pode acessar seu atalho no Menu Iniciar do Windows, na pasta onde o seu MetaTrader se encontrar. Porém uma maneira mais fácil é chamá-lo diretamente do MetaTrader, para isso você possui três caminhos distintos para chegar a este objetivo. Considere a figura

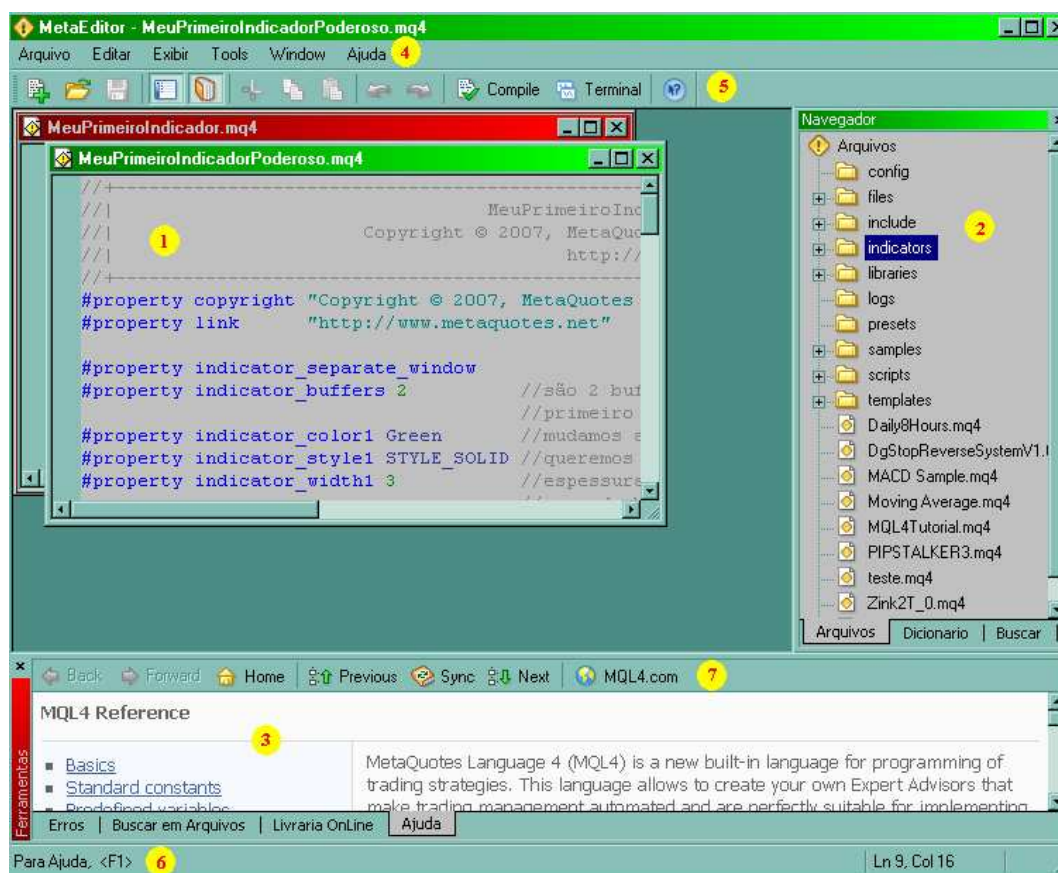


- 1) Menu "Ferramentas" e "Editor da Linguagem MetaQuotes"
- 2) Simplesmente teclando F4 quando estiver no MetaTrader
- 3) Acessando este Ícone na Barra de Ferramentas

---

## Familiarizando-se com o MetaEditor

O Meta editor possui as características da figura abaixo:



- 1) Área destinada ao desenvolvimento de seu código
- 2) Navegador, aqui você pode localizar mais facilmente os códigos disponíveis em seu diretório Experts dentro do seu MetaTrader.
- 3) Ferramentas, aqui você terá uma interação direta, quando necessitar de alguma ajuda, ou quando compilar um programa ou ainda quando fizer uma busca em algum arquivo em disco.
- 4) Menu do MetaEditor
- 5) Barra de Ferramentas
- 6) Barra de Status
- 7) Menu da janela de ferramentas

## O Menu de Opções do MetaEditor

### Menu de Opções

Apesar do meu MetaEditor estar em português, existem algumas mensagens que aparece em inglês, creio que são problemas de tradução, no fim deste capítulo ensino como modificar as mensagens do MetaEditor

Arquivo	Novo	Começa o projeto de um novo código, chamando o assistente do MetaEditor
	Open	Abre um arquivo existente
	Fechar	Fecha o arquivo que atualmente tem o foco do

		teclado, você pode ter mais de um arquivo aberto, porem somente um deles terá a atenção (foco) do teclado.
	Save	Salva o arquivo que tem o foco atual do teclado
	Save As	Salva o arquivo que tem o foco atual do teclado, porem com um outro nome que você poderá especificar
	Save All	Salva todos os arquivos que estão abertos.
	Compilar	Compila o código do arquivo que esteja com o foco do teclado.
	Imprimir Setup	Configura a impressora que será utilizada para imprimir
	Print Preview	Visualiza como ficara a impressão do arquivo do código atual
	Print	Imprime o arquivo do código atual
	Lista dos ultimos abertos	Mantêm uma lista dos n últimos arquivos que foram acessados
Editar	Desfazer	Desfaz a ultima digitação
	Refazer	Recupera a ultima digitação desfeita
	Cortar	Retira e transfere para a área de transferência do Windows um texto selecionado
	Copiar	Copia para a área de transferência do Windows um texto selecionado
	Colar	Coloca, a partir da posição atual do cursor, no seu código, o texto que estiver disponível na área de transferência do Windows
	Delete	Apaga um texto selecionado
	Select All	Seleciona todo o código do arquivo atual
	Find	Busca uma ocorrência de de texto no seu código
	Find Next	Busca próxima ocorrência, baseado na ultima busca
	Find Previous	Busca a ocorrência anterior, baseado na ultima busca
	Replace	Substitui um texto por outro em seu código
	Buscar em Arquivos	Executa busca de textos em Arquivos, esta opção tem interação com a janela de ferramentas.
	Toggle BreakPoint	
	Clear All Break Point	
	Book Mark	Coloca marcas em seu texto, de modo que você














		possa ter um acesso rápido a determinadas seções do seu código.
		Toggle Coloca ou Tira uma marca de texto
		Próximo Vai para a próxima marca de texto configurada
		Anterior Vai para a marca de texto anterior a atual que foi configurada
		Limpar Elimina todas as marcas de texto em seu código
	List Names	Abre uma pequena janela, onde o cursor se encontra em seu texto, com todas as funções e palavras chaves disponíveis na linguagem MQL4, muito útil para você verificar a sintaxe enquanto você digita. Mas não se preocupe, O MetaEditor, é amigável suficiente para abrir automaticamente esta janela depois de você digitar a terceira letra de uma palavra e ele conseguir identificar como uma palavra de sua lista
Exibir	Parameter Info	Quando você digita uma das funções padrões de MQL4, e não se lembra quais são os parâmetros que você pode utilizar na mesma esta opção lhe oferece uma ajuda rápida lembrando você de como utilizar a referida função
	Languages	Aqui você pode modificar o idioma na qual o MetaEditor conversa com voce.
	Toolbar	Mostra ou esconde a Barra de tarefas
	Status Bar	Mostra ou esconde a Barra de Status
	Tool Box	Mostra ou esconde a janela de Ferramentas
	Navegador	Mostra ou esconde a janela do Navegador
Tools	Customize	Deixa que você modifique as barras de ferramentas do MetaEditor
	Terminal de negociações	Chama o MetaTrader, onde você faz suas negociações.
Windows	Opções	Configura o MetaEditor de acordo com suas atribuições.
	Nova Janela	Abre uma nova janela texto, com o mesmo código do arquivo que atualmente tem o foco do teclado
	Cascata	Organiza suas janelas abertas em Cascata
	Tile Horizontal	Organiza suas janelas Horizontalmente



	Tile Vertical	Organiza suas janelas Verticalmente
	Arrumar Icones	Configura suas janelas minimizadas
	Fechar Todos	Fecha todas as janelas Abertas
	Lista de janelas abertas	Mantém a lista de arquivos abertos do MetaEditor
Ajuda	Ajuda Tópicos	Ajuda especifica sobre o MetaEditor
	Sobre	Características do MetaEditor

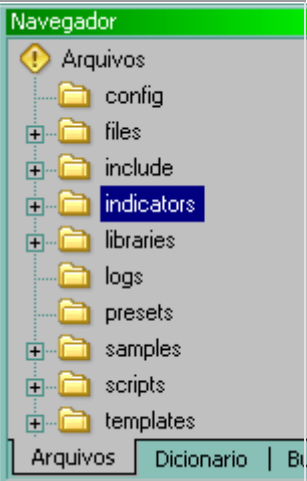
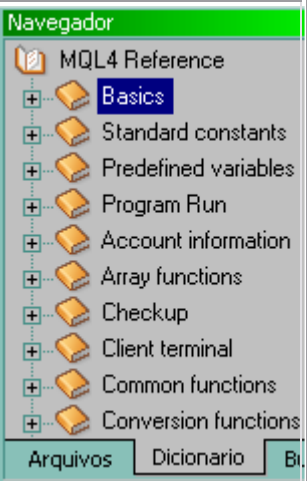
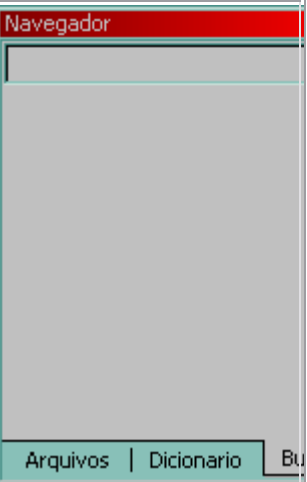
---

## A Barra de Ferramentas do MetaEditor

Barra de Ferramentas	
	Inicia um novo arquivo de código MQL4 através do assistente
	Abre um arquivo existente
	Salva arquivo atual em disco
	Exibe/Esconde Janela de Ferramentas
	Exibe/Esconde Janela do Navegador
	Recorta texto selecionado e manda para a área de transferencia
	copia texto selecionado para a área de transferencia
	Coloca, a partir da posição atual do cursor, no seu código, o texto que estiver disponível na área de transferência do Windows
	Desfaz a ultima digitação
	Recupera a ultima digitação desfeita
 Compile	Compila o código do arquivo que esteja com o foco do teclado.
 Terminal	Chama o MetaTrader, onde você faz suas negociações.
	Mostra ajuda para item que sera selecionado

---

## O Navegador do MetaEditor

Janela do Navegador		
		
Aqui você encontra todos os diretório e subdiretório que o MetaTrader utiliza diretamente, bem como aos arquivos neles encontrados	Aqui se encontra o Menu de ajuda da linguagem MQL4, onde você pode tirar suas duvidas sobre as regras da mesma	Simplesmente para digitar um texto a ser buscado no arquivo de ajuda da linguagem MQL4

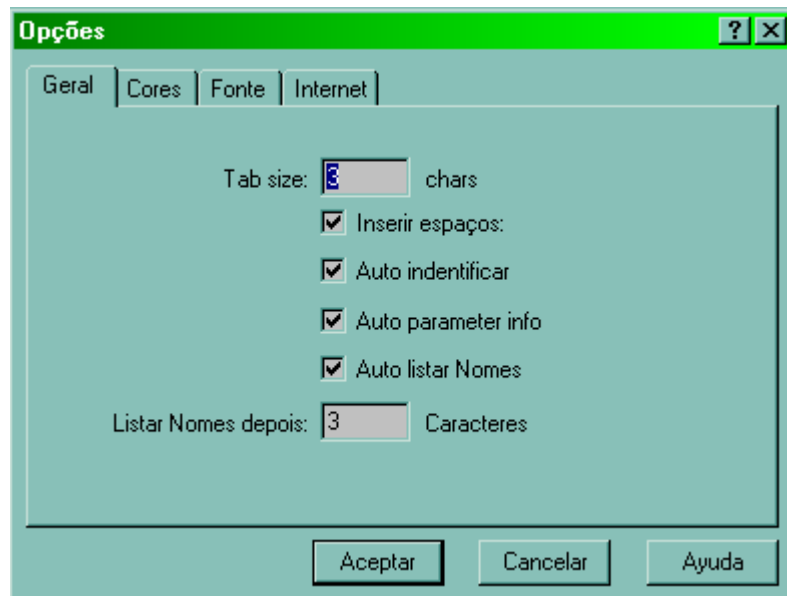
## Configurando o MetaEditor

Agora, aprenderemos como modificar algumas características do MetaEditor para que nos pareça mais amigável, em nosso trabalho do dia a dia.

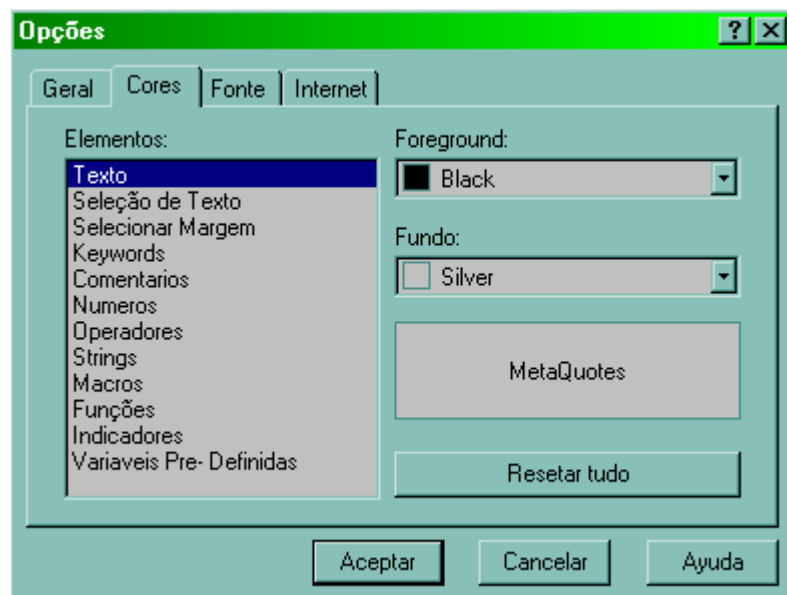
Primeiramente as opções gerais.

- Tab size : é o tamanho em caracteres que terão nossas tabulações, elas aparecerão quando você teclar a tecla TAB, na verdade nada mais é o numero máximo de espaços em branco ate a próxima tabulação.
- Inserir espaços : Ao passar para a próxima linha e se o sistema "Auto identificar" estiver ligado, o MetaEditor vai inserir espaços em branco em vez de tabulações.
- Auto identificar : quando você teclar o <enter> para a próxima linha, o MetaEditor se posicionara exatamente embaixo do começo da ultima linha
- Auto parameter info : ao identifica uma função e quando você teclar o parênteses, aparece uma pequena janela de ajuda de como os parâmetros de vem ser passados na chamada da referida função, mas isto somente funciona com as funções padrões do MQL4, para as funções que você desenvolve não existe esta ajuda.
- Auto Listar nomes : diz para o MetaEditor mostras as possíveis funções ou palavras chaves que você pode utilizar e que são padroes do MQL4, isso ajuda você a não cometer erros de sintaxe.

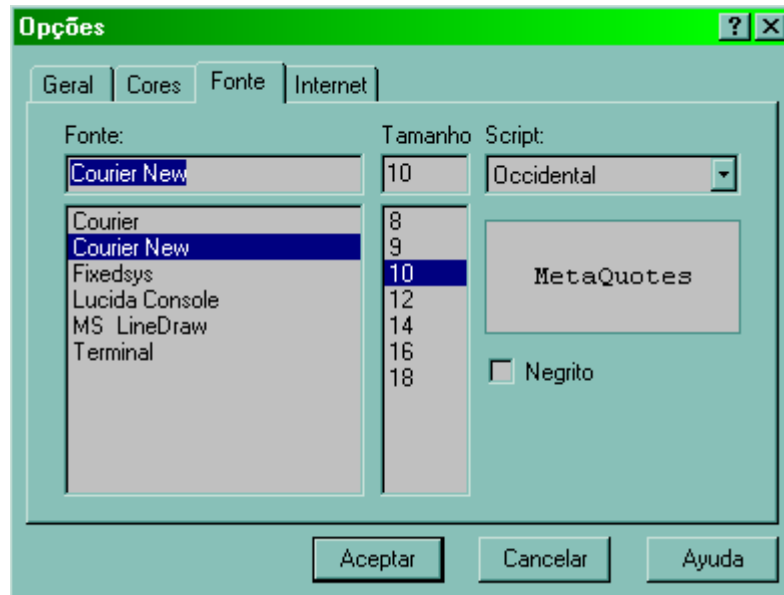
- Listar nomes depois de : Mostra a janela de nomes do MQL4, depois de você digitar o caractere da palavra, desde que o começo tenha alguma ocorrência na lista de nomes



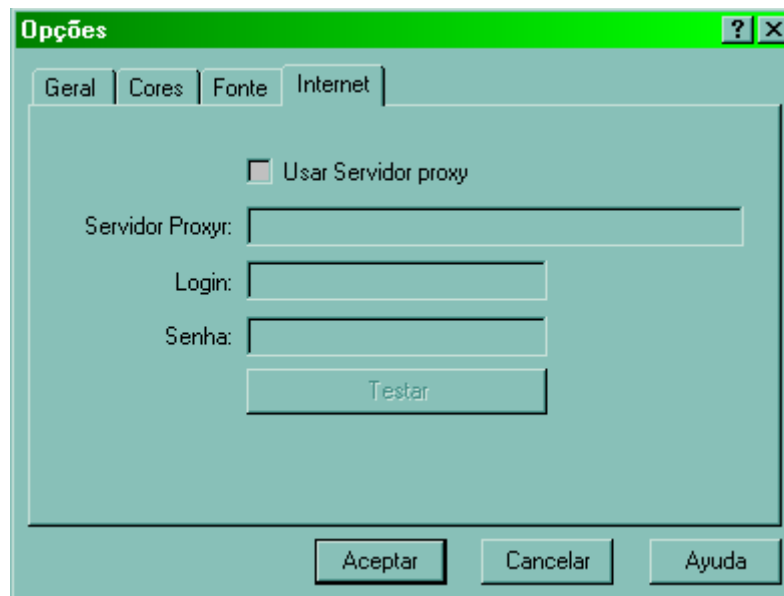
Você notara que a medida que você digita as palavras no editor, elas assumem uma cor diferente. Isto se da ao fato de que o MetaEditor usa o esquema de identificação de palavras e identificadores baseado em cores. Isto torna se código mais legível e fácil de entender. Aqui simplesmente você pode dar as cores que quiser aos diferentes grupos de palavras que o MetaEditor identifica.



Se você não gosta da fonte utilizada pelo MetaEditor, ou se acha ela pequena demais ou muito grande, na próxima tabulação das opções você pode modificá-la e deixá-la da maneira que você quiser. Porém você só pode utilizar fontes de tamanho fixo, estas fontes, não importa a letra, elas ocupam a mesma largura na tela, isto é o "i" é tem o mesmo espaço reservado que o "m".



Bom por ultimo, se você acessa sua internet com um servidor proxy, é aqui que você o configura, isto é para utilizar a ajuda on-line já janela de ferramentas.



## Considerações finais sobre o MetaEditor

Existe mais uma particularidade sobre o MetaEditor, porém ela será discutida mais adiante nas lições onde necessitaremos. Esta particularidade se chama o Assistente do MetaEditor. Com ele, o MetaEditor, identifica o propósito de nosso código e já monta uma máscara inicial para o mesmo, poupando um bom trabalho de digitação.

## Sintaxe da Linguagem MLQ4

---

Espero que estejam gostando do nosso curso. Agora que você já está familiarizado com o MetaEditor, chegou a hora de começarmos a realmente aprender as bases do MQL4. Como já citado anteriormente, se você tiver conhecimento da linguagem C ou C++, então, você já tem um grande conhecimento de MQL4, pois a sintaxe de MQL4 é muito semelhante a de C e C++. O termo sintaxe, verificado no dicionário da língua portuguesa é definido como:

do Latin : syntaxe < Gr. sýntaxis, arranjo, disposição.

Substantivo Feminino, parte da estrutura gramatical de uma língua que contém as regras relativas à combinação das palavras em unidades maiores (como as orações), e as relações existentes entre as palavras dentro dessas unidades; parte da gramática que estuda estas relações.

Em termos de linguagem de programação, sintaxe é definida como:

A sintaxe de uma linguagem de programação pode ser descrita por uma gramática independente de contexto e representada graficamente através da notação da forma de Backus-Naur (BNF). BNF é uma meta-sintaxe usada para expressar gramáticas livres de contexto: isto é, um modo formal de descrever linguagens formais. O conjunto de palavras (tokens), compostos de acordo com essas regras, constituem o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador.

Assim quando estudamos a sintaxe de uma linguagem (computacional ou não) nós estamos estudando um conjunto de regras de gramática e de escrita que consistem em:

- Formatos
- Identificadores
- Palavras reservadas

Exemplo : Deixe-me cortar o bolo.

---

## Formato

Quando você escreve seu código, você pode livremente usar espaços, tabulações e de linhas que vazias. Se você formata seu código de maneira que fique mais agradável aos olhos (Leitura e entendimento). Por exemplo estas 3 formas de definir variáveis são válidas em MQL4:

```
double MacdCurrent, MacdPrevious, SignalCurrent;  
  
double  
  
MacdCurrent,  
  
MacdPrevious,  
  
SignalCurrent;  
  
double      MacdCurrent,      MacdPrevious,      SignalCurrent;
```

Mas, como você vê, a primeira linha é mais legível e fácil de compreender. E como tudo no mundo, há exceções à regra:

1) você não pode usar a linha nova no "controle da compilação" (Preprocessors)

Você saberá mais sobre o "controle da compilação" numa das seguintes lições, mas recorde apenas que isto é uma exceção. Para o exemplo a linha seguinte do código é inválida e o compilador MQL4 reclamar:

```
#property  
copyright "Copyright © 2004, MetaQuotes Software  
Corp."
```

Esta seria a forma correta da sintaxe para o "controle da compilação":

```
#property copyright "Copyright © 2004, MetaQuotes Software  
Corp."
```

2) você não pode usar a linha ou o espaço novo no meio de valores constantes, de identificadores ou de palavras-chaves.

Para o exemplo esta linha é válida:

```
extern int MA_Period=13;
```

"extern" e "int" são aqui as palavras-chaves, "MA\_Period" é um identificador e "13" é um valor constante. Você saberá mais nas lições seguintes sobre estas terminologias. Por exemplo as linhas seguintes são inválidas:

```
extern int MA_Period=1
3;

extern int MA_Period=1      3;
```

---

## Comentários

Para fazer o mundo de programação mais fácil, toda a linguagem de programação tem seu estilo de comentários da escrita. Você usa comentários para escrever linhas em seu código (ou parte de uma linha) que o compilador ignorará, porém, elas farão seu código mais compreensível. Suponha que você escreva um programa no verão e no inverno você quer o ler. Sem comentários, mesmo você sendo o criador criador do código, você provavelmente não compreenderá, em primeira instancia, todas estas linhas que você escreveu. MQL4 (& C/C++) usam dois tipos de estilos dos comentários

**1) Comentário de linha :** uma única linha para comentários, a linha do comentário começa com "//" e termina com a linha nova. Por exemplo:

```
//Este é um comentário de linha

extern int MA_Period=13;  //Este também é um comentário de linha
```

**2) Comentário de varias linhas:** começa o comentário com "/\*" e termina com "\*/". Em outras palavras tudo que estiver entre "/\*" e "\*/" inclusive novas linhas será considerado comentário (você pode ter também comentários de linha dentro dos comentários, o que vale é o comentário de varias linhas). Com essa facilidade é possível você eliminar uma parte do código que no momento você não deseja (mas no futuro pode ter que usar) simplesmente colocando ele entre os símbolos de comentários de varias linhas. Por exemplo:

```
/* este
é um
comentário

de varias
linhas*/

/* este
é um
comentário      // com um comentário de uma só linha dentro

de varias
linhas*/
```

---

## Identificadores

Um identificador é o nome que você escolhe a suas variáveis, constantes e funções. Por o exemplo MA\_Period aqui é um identificador:

```
extern int MA_Period=13;
```

Há poucas regras e limitações para escolher nomes de identificadores:

- 1) O comprimento máximo (tamanho) do identificador não deve exceder 31 caracteres.
- 2) O identificador deve começar com uma letra (maiúscula ou minúscula) ou o símbolo sublinhando ( \_ ). Assim, não se pode começado o nome de um identificador com um número ou um outro símbolo (que não seja o símbolo sublinhando).
- 3) Você não pode usar nenhuma palavras chave como um identificador. Você verá a lista das palavras chaves mais adiante nesta mesma lição.
- 4) Os nomes dos identificadores são caso sensíveis ao caso (diferenciam letras maiúsculas de minúsculas). Assim, MA\_PERIOD não o mesmo que o ma\_period ou o MA\_Period ou Ma\_PeRioD (ou qualquer outra combinação de letras maiúsculas com minúsculas no nome com as mesmas letras nas mesmas posições). Deixe-nos fazer exame de alguns exemplos:

nomes válidos	nomes inválidos	
Nome1	Minha_Primeira_Variavel_Longa_1	mais de 31 caracteres
N1o2m3e4	~Nome	símbolo que não é o sublinha
Minha__Variavel	123Nome	começa com numero
_Nome	No#me	símbolo que não é o sublinha
Nome_1	double	palavra-chave (reservada)

---



## Palavras-chaves

Para cada língua (idioma) existem "palavras" que a ela usa para ações específicas ou determinar alguma coisa. Em linguagens computacionais é a mesma coisa. Assim, são algumas palavras são reservados ao uso da linguagem e você não pode usá-los como um nome do identificador ou para nenhuma outra finalidade que não seja aquela que elas foram criadas. Esta é a lista das palavras-chaves reservadas na linguagem MQL4:

Tipos de dados	Classe de memória	Operadores	outros
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		for	
string		else	
void		if	
		return	
		switch	
		while	

Como você pode observar, são poucas as palavras chaves, porem elas representam todo o poder da linguagem MQL4. Quero ainda exemplificar, baseado em palavras chaves, alguma linhas de programação invalidas:

```
extern int datetime =13; // datetime é palavra reservada
int extern =20;          // extern é palavra reservada
double continue = 0;     // continue é palavra reservada
```

# Tipos de dados da Linguagem MLQ4

---

## O que é um tipo de dado?

Toda a linguagem de programação tem um conjunto dos nomes da representação das informações (de agora em diante chamados de dados) que ela armazena na memória. Por exemplo se a memória tiver armazenado números entre -2147483648 a 2147483647, a maioria das linguagens de programação nomeará como "integer" esse tipo de dado. Então, dependendo do que você necessitar armazenar numa determinada região da memória você deverá dizer ao compilador o tipo de dado que vai colocar lá.

---

## Variáveis

As variáveis são os nomes que daremos às partes da memória em que os dados podem ser armazenados. Para visualizar melhor, pensar que isso é como um retrato, imagine que a memória é uma série de caixas diferentes do tamanho. O tamanho da caixa é área de armazenamento da memória requerida nos bytes (Um byte é um dos tipos de dados integrais em computação. É usado com frequência para especificar o tamanho ou quantidade da memória ou da capacidade de armazenamento de um computador, independentemente do tipo de dados lá armazenados). Agora imagine essas caixas e acompanhe os raciocínios abaixo:

- A fim de usar uma caixa para armazenar dados, a caixa deve ser dada um nome; este processo é conhecido como a declaração.
- No processo da declaração você usa uma palavra que diz ao computador qual é o tipo e o tamanho da caixa que você quer se usar, esta palavra é conhecida como a Palavra-Chave.
- Ajuda muito se você der a uma caixa um nome significativo que se relacione ao tipo de informação que você colocará nela, e que torne mais fácil de encontrar os dados.
- Os dados são colocados em uma caixa através da atribuição dos dados à esta caixa.
- Quando nós atribuímos o valor da caixa na mesma linha que você declarou, este processo é conhecido como a iniciação.

Quando nós criamos uma variável que nós somos dizendo ao computador que nós queremos atribuir um comprimento especificado da memória (nos bytes) a nossa variável, desde armazenar um número simples, a uma letra ou um número grande, com certeza, cada um destes processos não ocupará o mesmo espaço na memória. Deste modo você deve informar ao computador quais são o tipo dos dados e qual o comprimento dos dados. Para isto servem os tipos de dados.

Por exemplo se escrevermos esta linha de código :

```
int MyVariable=0;
```

Este é o meio pelo qual nós estamos requisitando ao computador para ajustar um bloco de um comprimento de 4 bytes na memória com o nome de "MyVariable". No exemplo acima nós especificamos ao computador como se fosse a seguinte declaração :

```
int          ==> representa uma palavra-chave
int          ==> representa um tipo de dados que especifica
números inteiros
int          ==> representa uma declaração
MyVariable   ==> Nome da variável
=0;          ==> Atribuímos um valor inicial a variável
```

Nós saberemos mais sobre variáveis nas lições futuras. Bem, agora, continuando, em MQL4, estes são os tipos de dados:

- Inteiro ([int](#))
- Lógico ([bool](#))
- Literais ([char](#))
- Cadeia de Literais ([string](#))
- Ponto Flutuante ([double](#))
- Cores ([color](#))
- Data e Hora ([datetime](#))

---

### Tipo de dados para valores inteiros ([int](#))

Um inteiro (ocupa 4 bytes na memória), é uma região de memória onde pode ser armazenado um número inteiro (sem casas decimais) que possa começar com a + ou a - e pode conter uma certa quantidade de dígitos. E seu valor da escala está entre - 2147483648 a 2147483647. MQL4 apresenta o inteiro no formato decimal ou hexadecimal (é um sistema de numeração que usa 16 símbolos). Por exemplo os números seguintes são inteiros:

```
Sistema de numeração Decimal      : 12, 3, 2134, 0, -230
Sistema de numeração Hexadecimal : 0x0A, 0x12, 0X12, 0x2f, 0xA3,
0xa3, 0X7C7
```

Use a palavra-chave `int` para criar uma variável para armazenar um número inteiro

```
int intInteger = 0;
int intAnotherIntger = -100;
int intHexIntger = 0x12;
```

Decimal: A notação decimal é a escrita dos números na base de 10, e usa os dígitos (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9) para representar números. Estes dígitos são usados frequentemente com um ponto decimal que indique o começo de uma parte fracionária, e com um dos símbolos do sinal + (mais) ou (menos) para indicar o sinal.

O hexadecimal é um sistema numeração com uma base de 16 símbolos e que são escritos geralmente usando os símbolos 0-1-2-3-4-5-6-7-8-9 e A-B-C-D-E-F ou a-b-c-d-e-f. Para entender melhor vamos converter um numero decimal para hexadecimal e vice-versa

Converter o numero decimal 12345 para Hexadecimal. Basta fazermos divisões sucessivas :

```
começamos com      12345 / 16 = 771 e sobra 9
agora dividimos     771 / 16 = 48 e sobra 3
agora dividimos     48 / 16 = 3 e sobra 0
```

para montar o numero hexadecimal utilizamos o resultado da ultima divisão com todos os restos de todas as divisões, logo teremos que 12345 em decimal é 3039 em hexadecimal.

Agora converteremos o numero hexadecimal AB12 para decimal. Basta fazermos Potenciações com o numero 16, multiplicações e somas:

```
2 = 2 x 16^0 = 2 x 1 = 2
1 = 1 x 16^1 = 1 x 16 = 16
B = 11 x 16^2 = 11 x 256 = 2816
A = 10 x 16^3 = 10 x 4096 = 40960
-----
43794
```

Como vimos, para montar o numero decimal somamos todas as multiplicações, logo teremos que AB12 em hexadecimal é 43794 em decimal. Observação: no sistema hexadecimal A=10, B=11, C=12, D=13, E=14, F=15

Explicar porque usar números hexadecimais esta fora do escopo deste curso. Porem se você tiver interessado busque no google (favor utilizar nossa caixa de busca na pagina de entrada) para descobrir porque o sistema hexadecimal é tão importante na computação.

---

## Tipos de dados para valores Lógicos (bool)

A variável lógica (ocupa 4 bytes na memoria, em uma representação interna é um inteiro) é um tipo de dados que pode ter somente duas reapresentações de valores, o verdadeiro e o falso (ou seus os numéricos, 0 e 1). E ocupa 1 bloco da memória. Em MQL4 as palavras-chaves **false**, **False** e **FALSE** ou **true**, **True** e **TRUE** são os iguais. São conhecidos como valores Booleanos, assim nomeados por homenagem ao

matemático grande Boole George. Nós usamos a palavra chave `bool` criar uma variável Lógica, de agora em diante chamada de booleana. Por exemplo:

```
bool I = true;
bool bFlag = 1;
bool bBool=FALSE;
```

---

## Tipo de dados para valores Literais (`int`)

MQL4 nomeia este tipo de dados "literal" ou Character (ocupa 4 bytes na memória, em uma representação interna é um inteiro). Um literal (denominado de agora em diante como caracter) é um numero de 0 a 255 que define um elemento alfabéticos, numéricos, e especiais definido no ASCII (código de padrão americano para o intercâmbio de informação). Os caracteres têm valores do inteiro correspondente à posição na tabela ASCII. Você escreve a constante de caráter usando as aspas simples (') antes e depois do caráter a ser definido. Por exemplo:

```
'a', '$', 'Z'
```

Usaremos a palavra chave "`int`" para criar um tipo de dados de caracter

```
int chrA = 'A';
int chrB = '$';
```

Existem alguns caracteres chamados caracteres especiais e não podem apresentar-se diretamente dentro das aspas simples porque têm mecanismos reservados na linguagem MQL4. Aqui nós usamos algo como uma seqüência de escape para mostrar estes caracteres especiais, E isso se faz prefixando o caractere especial junto com o caractere de barra invertida (\). Por exemplo:

```
int chrA = '\\'; // quando quiser usar a barra invertida
int chrB = '\\n'; // especifica nova linha em um texto
```

Esta é a representação das seqüências de caracteres com escape utilizadas em MQL4 :

carriage return <code>\r</code>	retorno do carro da impressora ou do cursor
new line <code>\n</code>	nova linha
horizontal tab <code>\t</code>	tabulação horizontal
reverse slash <code>\\</code>	barra invertida
single quote <code>\'</code>	aspas simples

double quote                      \"	aspas duplas
hexadecimal ASCII- code    \xhh	código hexadecimal onde xx é o numero a ser representado

---

## Tipo de dados valores de cadeias de Literais

O tipo de dados cadeia de literais (tem uma estrutura que ocupa 8 bytes na memória mais o tamanho em bytes da cadeia a ser armazenada mais um valor nulo ao final da cadeia) ou cadeia de caracteres (de agora em diante designado por string) é uma disposição dos caracteres incluídos entre um par de aspas duplas ("). A disposição dos caracteres é de um após outro em uma seqüência, como se fosse uma fila, numerados seqüencialmente a partir da posição inicial na memória, começando no índice 0. Após o último caráter dos dados, um caráter NULO é colocado na posição seguinte indicando o termino da cadeia. Não importa se houver posições não utilizadas. Um caráter NULO é um caráter especial (representado pelo código 0 do ASCII) usado marcar a extremidade deste tipo de cadeia. Veja a representação simples da string "Brasil" na disposição de caracteres.

B	r	a	s	i	l	NULO	7	8	9	...
0	1	2	3	4	5	6	7	8	9	...

MQL4 limita o tamanho da variável da string a 255 caracteres e qualquer tentativa de armazenar mais de 255 caracteres gerará este erro: (string muito longa (máximo de 255 caracteres)). Você pode usar todo o caráter especial - mencionado acima em sua string desde que usado com a barra invertida (\). Nós usamos a palavra-chave "**string**" para criar variável. Por o exemplo:

```
string str1 = "Hello world1, com a cortesia de DooMGuard";
string str2 = "Copyright © 2005, \"Forex-tds forum\"."; //Nota : usa as aspas
duplas dentro da string
string str3 = "1234567890";
```

---

## Tipo de dados valores de Ponto Flutuante

Os números de ponto flutuante (ocupa 8 bytes na memória) são do conjunto de números reais (isto é, um número que possa conter uma parte fracionária ao lado da parte inteira, separado com ponto (.). Exemplos:

3.0, -115.5, 15 e 0.0001
--------------------------

E seu valor pode estar entre 2.2e-308 até 1.8e308. Nós usamos a palavra-chave "**double**" para criar uma variável de ponto flutuante. Por exemplo:

```
double dblNumber1 = 10000000000000000;  
double dblNumber3 = 1/4;  
double dblNumber3 = 5.75;
```

---

## Tipo de dados valores de cores

O tipo de dados para cores (ocupa 4 bytes na memória) é um tipo de dados MQL4 especial, que representa uma cor que aparecerá nos gráficos do MetaTrader quando você cria seu próprio indicador, o usuário pode a mudar na tabulação de propriedade de seu indicador. Você pode ajustar a variável da cor de três maneiras:

- 1 - Pelo nome da cor: Para os nomes de cores conhecidas (chamadas jogo de cores web) você podem atribuir o nome da cor à variável da cor.
- 2 - Pela representação de Caractere (MQL4 chamou assim): Neste método você usa a palavra-chave (C) seguido por três números separados por vírgula e entre aspas simples, estes números representam intensidades das cores primárias: vermelho, verde e o azul (conhecida como o valor do RGB da cor). Estes valores têm que estar no entre: 0 a 255. E você pode escrever estes valores no formato decimal ou hexadecimal.
- 3 - Pelo valor do inteiro: Cada cor no jogo de cores tem seu valor inteiro que você pode escrever no formato decimal ou hexadecimal. E você pode atribuir o valor do inteiro da cor à variável da cor. O formato hexadecimal da cor é como este: 0xBBGGRR onde BB é o valor azul, GG é valor verde e o RR é o valor vermelho. Por o exemplo:

```
// Constantes Numéricas  
C'128,128,128' // cinza  
C'0x00,0x00,0xFF' // Azul  
  
// named color  
Red // Vermelho  
Yellow // Amarelo  
Black // Preto  
  
// Representação por número inteiro  
0xFFFFFFFF // branco  
16777215 // branco  
0x008000 // verde  
32768 // verde
```

Usaremos a palavra-chave "**color**" para definir uma variável para cores

```
color clr1= Red;  
color clr2= C'128,128,128' ;  
color clr3=32768;
```

## Quadro de paletas de cores com seus respectivos nomes (respeitar minúsculas e maiúsculas)

Black	DarkGreen	DarkSlateGray	Olive	Green	Teal	Navy	Purple
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOliveGreen	SaddleBrown	ForestGreen	OliveDrab
SeaGreen	DarkGoldenrod	DarkSlateBlue	Sienna	MediumBlue	Brown	DarkTurquoise	DimGray
LightSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	Crimson	SteelBlue
Goldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	LimeGreen	OrangeRed
DarkOrange	Orange	Gold	Yellow	Chartreuse	Lime	SpringGreen	Aqua
DeepSkyBlue	Blue	Magenta	Red	Gray	SlateGray	Peru	BlueViolet
LightSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	SlateBlue	DarkKhaki
IndianRed	MediumOrchid	GreenYellow	MediumAquamarine	DarkSeaGreen	Tomato	RosyBrown	Orchid
MediumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	MediumSlateBlue	Tan
DarkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	SkyBlue	LightSalmon
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSkyBlue	LightSteelBlue	LightBlue
PaleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGray	Wheat	NavajoWhite
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	LightGoldenRod	BlanchedAlmond
LemonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	LightCyan	Linen
Lavender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	Honeydew	AliceBlue
LavenderBlush	MintCream	Snow	White				

### Tipo de dados de Date e Hora (datetime)

O tipo de dados de Date e Hora (ocupa 4 bytes na memória) é um tipo de dados MQL4 especial, que liga o dado a data e a hora. Para atribuir uma data e hora literalmente use a letra (D) e entre aspas simples informe a data e a hora, que consiste em 6 partes: o valor do ano, o mês, a data, a hora, os minutos, e os segundos. O valor de **datetime** tem de esta entre: 1 de janeiro de 1970 a 31 de Dezembro de 2037.

```
D'2004.01.01 00:00' // inicio do ano
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12' //igual a D'1980.07.19 12:00:00'
D'01.01.2004' //igual a D'01.01.2004 00:00:00'
```

Usaremos a palavra-chave "**datetime**" para criar a variável :

```
datetime dtMyBirthDay= D'1972.10.19 12:00:00';
datetime dt1= D'2005.10.22 04:30:00';
```



# Operadores e Expressões da Linguagem MLQ4

---

## Que é o arranjo das operações & das expressões?

Você sabe o que são as operações matemáticas muito bem. Se eu lhe disser que

- A) + (mais)
- B) - (menos)
- C) \* (multiplicar)
- D) / (dividir)

São os operadores aritméticos básicos, você com certeza recordará muito rápido o que eles fazem. Eu ouço-o dizer "certo, eu sei as operações; agora poderia me dizer me o que é o meaning da expressão?"

Identificadores (você os recorda? Caso não recorde, reveja a lição da SINTAXE) junto com as operações produzem as expressões. Confuso? Deixe-me ilustrar um exemplo:

`x = (y*z) / w;`

ao analisermos esta parte da linha "`(y*z) / w`", observamos:

y, z e w são identificadores

=, \* e / são operadores

Juntos nesta linha (poderiam se 2 linhas ou mais também) eles formam uma expressão.

As combinações de uma expressões fazem uma indicação. E quando as indicações se juntam fazem uma função e quando as funções se juntam fazem um programa. No restante desta lição nós estaremos falando sobre os operadores usados em MQL4. Assim, deixe-nos começar com os operadores aritméticos básicos:

---

## Operações Aritméticas

Em MQL4 há 9 operações aritméticas Esta é a lista delas com o uso de cada uma:

Operador	Nome	Exemplo	Descrição
+	Operador da adição	<code>A = B+C;</code>	Adicione A a B e atribua o resultado a C.
-	Operador da subtração	<code>A = B-C;</code>	Subtraia C de B e atribua o resultado a C.
+ -	Operadores do troca de sinal	<code>A = -A;</code>	Mude o sinal de A de positivo ao negativo
*	Operador da multiplicação	<code>A = B*C;</code>	Multiplique B e C e atribua o resultado a A

/	Operador da divisão	<code>A = B/C;</code>	Divida B em C e atribua o resultado a A
%	Operador de modulo da divisão	<code>A = B%C;</code>	A é o resto da divisão entre B e C. exemplos: 10%2 resultara em 0 10%3 resultara em 1
++	Operador de incremento	<code>A++;</code>	Aumente A em uma unidade 1. exemplo: se A = 1 entao A++ resultara em A = 2
--	Operador de decremento	<code>A--;</code>	Diminua A em uma unidade 1. exemplo: se A = 1 entao A-- resultara em A = 0

Nota: você pode combinar operadores de incremento e decremento com outras expressões:

```
A = (B++)*5;
```

Mas se você preferir pode escrevê-la assim

```
B++;
A = B*5;
```

## Operações de Atribuição

A finalidade de toda a expressão está produzindo um resultado e os operadores de atribuição que atribuem o resultado do lado esquerdo a variável do lado direito. Por o exemplo:

```
A = B*C;
```

Aqui nós multiplicamos B e C e atribuímos o resultado a A. (=) está aqui o para executar de atribuição. Em MQL4 há 11 operações das atribuições Esta é a lista deles com o uso de cada uma:

Operador	Nome	Exemplo	Descrição
=	Operador de atribuição	<code>A = B;</code>	Atribua B a A.
+=	Operador de atribuição com soma	<code>A += B;</code>	É igual a: <code>A = A + B;</code> Adicione B a A e atribua o resultado a A.
-=	Operadores de atribuição com subtração	<code>A -= B;</code>	É igual a: <code>A = A - B;</code> subtraia B a A e atribua o resultado a A.
*=	Operadores de	<code>A *= B;</code>	É igual a:

	atribuição com multiplicação		<code>A = A * B;</code> multiplique B por A e atribua o resultado a A.
<code>/=</code>	Operadores de atribuição com divisão	<code>A /= B;</code>	É igual a: <code>A = A / B;</code> divida A por B e atribua o resultado a A.
<code>%=</code>	Operadores de atribuição com resto da divisão	<code>A %= B;</code>	É igual a: <code>A = A % B;</code> divida A por B e atribua o resto da divisão a A.
<code>&gt;&gt;=</code>	Operadores de atribuição com deslocamento	<code>A &gt;&gt;= B;</code>	É igual a: <code>A = A &gt;&gt; B;</code> desloque A em B bits a direita e atribua o resultado a A.
<code>&lt;&lt;=</code>	Operadores de atribuição com deslocamento	<code>A &lt;&lt;= B;</code>	É igual a: <code>A = A &lt;&lt; B;</code> desloque A em B bits a esquerda e atribua o resultado a A.
<code>&amp;=</code>	Operadores de atribuição com Lógica E (AND)	<code>A &amp;= B;</code>	É igual a: <code>A = A &amp; B;</code> compare se A e B são verdadeiros atribua resultado a A.
<code> =</code>	Operadores de atribuição com Lógica OU (OR)	<code>A  = B;</code>	É igual a: <code>A = A   B;</code> compare se A ou B são verdadeiros atribua resultado a A.
<code>^=</code>	Operadores de atribuição com Lógica OU Exclusivo (XOR)	<code>A ^= B;</code>	É igual a: <code>A = A ^ B;</code> compare se exclusivamente se A ou B são verdadeiros atribua resultado a A.
Nota: você aprenderá mais sobre operadores lógicos mais adiante nesta mesma lição			

## Operações Relacionais

Os operadores relacionais comparam dois valores (operandos) e resultam em um valor falso (**False** ou Zero) ou verdadeiro (**True** ou diferente de Zero). Seria como obter a resposta a seguinte pergunta "é João é mais alto do que Alfredo? Sim ou Não.?". Resposta sim seria **True** (sim, João é mais alto que Alfredo) e Não seria **False** (não, João não é mais alto que Alfredo).

Por exemplo

```
4 == 4; //true
4 < 4;  //false
4 <= 4; //true;
```

Em MQL4 há 6 operações relacionais Esta é a lista deles com o uso de cada um:

Operador	Nome	Exemplo	Descrição
==	é igual a	a == B	Verdadeiro se A for igual a B. Falso se A for diferente de B.
!=	é diferente de	a != B	Verdadeiro se A for diferente de B. Falso se A for igual a B.
>	é maior que	a > B	Verdadeiro se A for maior que B. Falso se A for menor ou igual a B.
<	é menor que	a < B	Verdadeiro se A for menor que B. Falso se A for maior ou igual a B.
>=	é maior ou igual a	a >= B	Verdadeiro se A for maior ou igual q B. Falso se A for menor que B.
<=	é menor ou igual a	a <= B	Verdadeiro se A for menor ou igual a B. Falso se A for maior que B.

## Operações Lógicas

Na ciência da computação, as álgebras booleanas são estruturas algébricas que "capturam a essência" das operações lógicas E, OU e NÃO, bem como das operações da teoria de conjuntos soma, produto e complemento. Receberam o nome de George Boole, matemático inglês, que foi o primeiro a defini-las como parte de um sistema de lógica em meados do século XIX. Mais especificamente, a álgebra booleana foi uma tentativa de utilizar técnicas algébricas para lidar com expressões no cálculo proposicional. Hoje, as álgebras booleanas têm muitas aplicações na eletrônica.

Foram pela primeira vez aplicadas a interruptores por Claude Shannon, no século XX. Os operadores da álgebra booleana podem ser representados de várias formas. É freqüente serem simplesmente escritos como E, OU ou NÃO (são mais comuns os seus equivalentes em inglês: AND, OR e NOT).

Na descrição de circuitos também podem ser utilizados NAND (NOT AND), NOR (NOT OR) e XOR (OR exclusivo). Os matemáticos usam com freqüência + para OU e . para E (visto que sob alguns aspectos estas operações são análogas à adição e multiplicação noutras estruturas algébricas) e representam NÃO com uma linha traçada sobre a expressão que está a ser negada.

MQL4 usa os 3 operadores lógicos os mais importantes. Esta é a lista deles com o uso de cada um:

Operador	Nome	Exemplo	Descrição
----------	------	---------	-----------

&&	e	A && B	Verdadeira se A e B forem Verdadeiro Falso para qualquer outra combinação
	ou	A    B	Verdadeira se A ou B forem Verdadeiro se A e B forem Verdadeiros Falso se A e B forem Falsos
!	negado	!A	Verdadeiro se A for Falso Falso se A for Verdadeiro

## Operações de bits (binários)

A lógica binária, ou bitwise operation é a base de todo o cálculo computacional. Na verdade, são estas operações mais básicas que constituem todo o poderio dos computadores. Qualquer operação, por mais complexa que pareça, é traduzida internamente pelo processador para estas operações.

**NOT :** O operador unário NOT, ou negação binária resulta no complemento do operando, i.e., será um bit '1' se o operando for '0', e será '0' caso contrário, conforme podemos confirmar pela tabela de verdade

	0	1
!	1	0

**AND :** O operador binário AND, ou conjunção binária devolve um bit 1 sempre que ambos operandos sejam '1', conforme podemos confirmar pela tabela de verdade:

&	0	1
0	0	0
1	0	1

**OR :** o operador binário OR, ou disjunção binária devolve um bit 1 sempre que pelo menos um dos operandos seja '1', conforme podemos confirmar pela tabela de verdade:

	0	1
0	0	1
1	1	1

**XOR :** O operador binário XOR, ou disjunção binária exclusiva devolve um bit 1 sempre que apenas um dos operandos é '1', conforme podemos confirmar pela tabela de verdade:

^	0	1
0	0	1
1	1	0

Shift : O operador unário de bit shifting, ou deslocamento bit-a-bit, equivale à multiplicação ou divisão por 2 do operando que, ao contrário dos casos anteriores, é um grupo de bits, e consiste no deslocamento para a esquerda ou para a direita do grupo de bits. O bit inserido é sempre 0, e o bit eliminado pode ser opcionalmente utilizado (flag CF dos registros do processador).

```
( 101011(43) >> 1 ) = 010101[1] (21)
( 101011(43) << 1 ) = [1]010110 (22)
```

MQL4 usa os 3 operadores lógicos os mais importantes. Esta é a lista deles com o uso de cada um:

Operador	Nome	Exemplo	Descrição
&	e	A & B	Compara duas partes e gera um resultado de 1 se ambos as partes forem 1; se não, retorna 0. A = 00000100 ( 4) B = 00000111 ( 7) & = 00000100 ( 4)
	ou	A   B	Compara dois bocados e gera um resultado de 1 se os bocados forem complementares; se não, retorna 0 A = 00000100 ( 4) B = 00000111 ( 7)   = 00000111 ( 7)
^	ou exclusivo	A ^ B	Compara duas partes e gera um resultado de 1 se um ou outro ou ambas as partes forem 1; se não, retorna 0. A = 00000100 ( 4) ^ = 00000011 ( 3)
~	complemento	~A	Usado para inverter todos as partes do operando. A = 00000100 ( 4) ~ = 11111011 (251)
>>	desloca para direita	A >> B	Move as partes para a direita, rejeita o as partes a direita, e atribui a parte esquerda um valor de 0. Cada movimento à direita divide eficazmente o valor ao meio. A = 00000100 ( 4) B = 00000111 ( 7) >> = 00000011 ( 3)
<<	desloca para esquerda	A << B	Move as partes para a esquerda, rejeita as partes a esquerda, e atribui a parte direita o valor de 0. Cada movimento à esquerda

			multiplica eficazmente por 2. A = 00000100 ( 4) B = 00000111 ( 7) >> = 00001110 ( 14)
Nota : todos os operandos associados com bits (bitwise) devem utilizados com ser inteiros.			

## Operações de bits (binários)

Há alguns operadores que se usam em MQL4 e não pertencem a uma das categorias específica:

- A) o operador de indexação "[]" (Chaves)
- B) o operador da ligação de controle "()" (Parenteses)
- C) o operador do separador dos argumentos da função ",", (Virgula)

Nós saberemos mais sobre as seus usos nas lições seguintes, recorde apenas estes 3 operadores como "outros operadores"

## Procedência de Operadores (prioridades)

Se você não indicar explicitamente a ordem em que você quer as operações em uma expressão composta sejam executados, a ordem é determinada pela precedência atribuída aos operadores no uso dentro da expressão. Os operadores com uma precedência mais elevada começam sendo avaliados primeiramente. Por exemplo, o operador da divisão tem uma precedência mais elevada do que o operador da adição. Assim, as duas seguintes indicações são equivalentes:

```
x + y / 100;
```

```
x + (y / 100); // mais legível para entender
```

Ao escrever expressões compostas, você deve ser explícito e indicar com os parênteses () que os operadores devem ser avaliados primeiramente. Você especifica a ordem de avaliação usando parênteses, sendo que os parênteses mais internos são avaliados primeiro. Esta prática fará seu código mais fácil de ler e manter. A seguinte tabela mostra a precedência atribuída aos operadores no MQL4. Os operadores nesta tabela são listados na ordem da precedência: Quanto mais acima na tabela um operador aparece, o mais elevado sua precedência ou prioridade. Os operadores com prioridade mais elevada são avaliados antes dos operadores com uma prioridade relativamente mais baixa. Os operadores no mesmo grupo têm a prioridade igual. Quando os operadores da precedência igual aparecem na mesma expressão, uma régua deve governar que seja avaliada primeiramente. Todos os operadores binários à exceção dos operadores de atribuição são avaliados da esquerda para a direita. Os operadores de atribuição são avaliados para a direita à esquerda

Operador	Nome	Descrição
( )	Parênteses	chamadas de funções
[ ]	Colchetes	seleção de elementos numa matriz
!	negação	negação lógica
~	complemento	bit a bit
-	sinal	Mudança de sinal (positivo/negativo de um numero)
*	multiplicação	da esquerda para a direita
/	divisão	
%	resto da divisão	
+	soma	da esquerda para a direita
-	subtração	da esquerda para a direita
<<	deslocamento a esquerda	da esquerda para a direita
>>	deslocamento a direita	da esquerda para a direita
<	menor que	
<=	menor ou igual a	
>	maior que	
>=	maior ou igual a	
==	igual	
!=	diferente	
&	e	
^	ou exclusivo	
	ou	
=	assinação	
+=	assinação com soma	
-=	assinação com subtração	
*=	assinação com multiplicação	
/=	assinação com divisão	
%=	assinação com resto da divisão	
>>=	assinação com deslocamento a direita	bit a bit
<<=	assinação com deslocamento a esquerda	bit a bit



<code>&amp;=</code>	assinação com e	bit a bit
<code> =</code>	assinação com ou	bit a bit
<code>^=</code>	assinação com ou exclusivo	bit a bit
<code>,</code>	virgula	da esquerda para a direita

# Algoritmos

---

Antes de prosseguir com nosso estudo, faz-se necessário algumas considerações, sobre como os programas computacionais trabalham, por este motivo, abri espaço para falar sobre algoritmos. Algoritmos pode ser descrito como o sequenciamento lógico do pensamento a ser seguido para a resolução de um problema. Meu Objetivo, com esta lição é mostrar o você o mundo dos algoritmos, lógico que somente lendo este capítulo você não se tornara um experto no assunto, mas poderá entender algumas coisas.

---

## Definição de Algoritmo

Um algoritmo é uma seqüência não ambígua de instruções que é executada até que determinada condição se verifique. Mais especificamente, em matemática, constitui o conjunto de processos (e símbolos que os representam) para efetuar um cálculo. O conceito de algoritmo é freqüentemente ilustrado pelo exemplo de uma receita, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Um algoritmo corretamente executado não irá resolver um problema se estiver implementado incorretamente ou se não for apropriado ao problema.

Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa. Sua implementação pode ser feita por um computador, por outro tipo de autômato ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros. Tal diferença pode ser reflexo da complexidade computacional aplicada, que depende de estruturas de dados adequadas ao algoritmo. Por exemplo, um algoritmo para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça enquanto outro algoritmo especifica que você deve primeiro vestir a calça e depois as meias e os sapatos. Fica claro que o primeiro algoritmo é mais difícil de executar que o segundo apesar de ambos levarem ao mesmo resultado.

---

## Etimologia

A palavra algoritmo tem origem no sobrenome, Al-Khwarizmi, do matemático persa do século IX Mohamed ben Musa, cujas obras foram traduzidas no ocidente cristão no século XII, tendo uma delas recebido o nome "Algorithmi de numero indorum", sobre os algoritmos usando o sistema de numeração decimal (indiano). Outros autores, contudo, defendem a origem da palavra em Al-goreten (raiz - conceito que se pode aplicar aos cálculos).

---

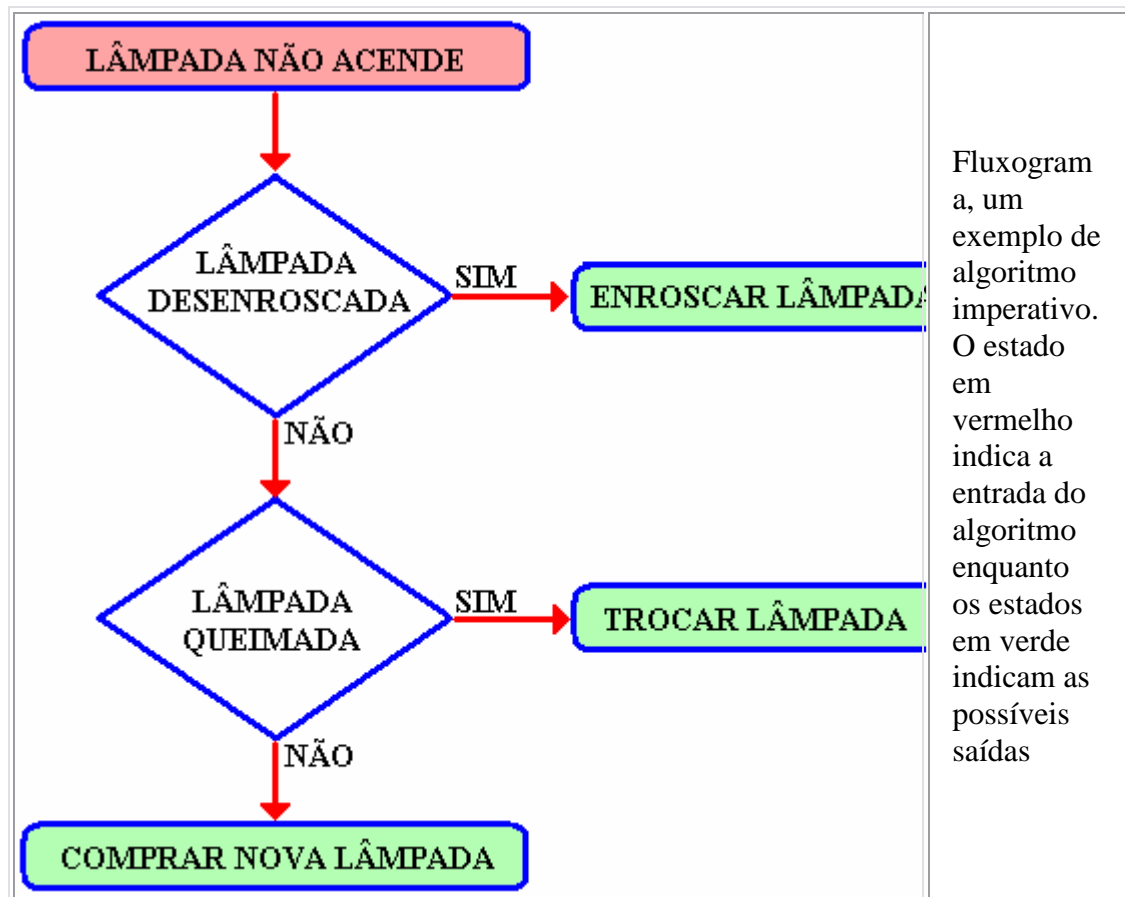
## **Formalismo**

Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos e em que ordem eles devem ser executados, como por exemplo, os passos a serem tomados para calcular as notas que serão impressas nos boletins dos alunos de uma escola. Logo, o algoritmo pode ser considerado uma seqüência de operações que podem ser simuladas por uma máquina de Turing completa (é um dispositivo teórico, conhecido como máquina universal).

Quando os procedimentos de um algoritmo envolvem o processamento de dados, a informação é lida de uma fonte de entrada, processada e retornada sob novo valor após processamento, o que geralmente é realizado com o auxílio de uma ou mais estruturas de dados (Dados organizados de forma coerente).

Para qualquer processo computacional, o algoritmo precisa estar rigorosamente definido, especificando a maneira que ele se comportará em todas as circunstâncias. O algoritmo pode ser provado matematicamente, bem como a quantidade de tempo e espaço (complexidade) necessários para a sua execução. Estes aspectos dos algoritmos são alvo da análise de algoritmos.

A maneira mais simples de se pensar um algoritmo é por uma lista de procedimentos bem definida, no qual as instruções são executadas passo a passo a partir do começo da lista, uma idéia que é pode ser facilmente visualizada através de um fluxograma. Tal formalização adota as premissas da programação imperativa, que é uma forma mecânica para visualizar e desenvolver um algoritmo. Concepções alternativas para algoritmos variam em programação funcional e programação lógica.



### Término do algoritmo

Alguns autores restringem a definição de algoritmo para procedimentos que eventualmente terminam. Minsky constatou que se o tamanho de um procedimento não conhecido de antemão, tentar descobri-lo é problema, já que o procedimento pode ser executado infinitamente, de forma que nunca se terá a resposta. Alan Turing provou em 1936 que não existe máquina de Turing para realizar tal análise para todos os casos, logo não há algoritmo para realizar tal tarefa para todos os casos. Tal condição é conhecida atualmente como problema da parada. Para algoritmos intermináveis o sucesso não pode ser determinado pela interpretação da resposta e sim por condições impostas pelo próprio desenvolvedor do algoritmo durante sua execução.

### Implementação

A maioria dos algoritmos é desenvolvida para ser implementada em um programa de computador. Apesar disso eles também podem ser implementados por outros modos tais como uma rede neural biológica (tal como no cérebro quando efetuamos operações aritméticas) em circuitos elétricos ou até mesmo em dispositivos mecânicos. Para programas de computador existem uma grande variedade de linguagens de programação, cada uma com características específicas que podem

facilitar a implementação de determinados algoritmos ou atender a propósitos mais gerais.

---

### Analise de Algoritmos

A análise de algoritmos é um ramo da ciência da computação que estuda as técnicas de projeto de algoritmos e os algoritmos de forma abstrata, sem estarem implementados em uma linguagem de programação em particular ou implementadas de algum outro modo. Ela preocupa-se com os recursos necessários para a execução do algoritmo tais como o tempo de execução e o espaço de armazenamento de dados. Deve-se perceber que para um dado algoritmo pode-se ter diferentes quantidades de recursos alocados de acordo com os parâmetros passados na entrada. Por exemplo, se definirmos que o fatorial de um número natural é igual ao fatorial de seu antecessor multiplicado pelo próprio número, fica claro que a execução de `fatorial(10)` consome mais tempo que a execução de `fatorial(5)`. Um meio de exibir um algoritmo afim de analisá-lo é através da implementação por pseudocódigo em português estruturado. O exemplo a seguir é um algoritmo em português estruturado que retorna (valor de saída) a soma de dois valores (também conhecidos como parâmetros ou argumentos, valores de entrada) que são introduzidos na chamada da função:

```
função SomaDeDoisValores (A numérico, B numérico)
início
    declare SOMA numérico
    SOMA <-- A + B
    retorne (SOMA)
fim
```

---

### Classificação por implementação

Pode-se classificar algoritmos pela maneira pelo qual foram implementados.

- A) Recursivo ou iterativo - um algoritmo recursivo possui a característica de invocar a si mesmo repetidamente até que certa condição seja satisfeita e ele é terminado, que é um método comum em programação funcional. Algoritmos iterativo usam estruturas de repetição tais como laços, ou ainda estruturas de dados adicionais tais como pilhas, para resolver problemas. Cada algoritmo recursivo possui um algoritmo iterativo equivalente e vice versa, mas que pode ter mais ou menos complexidade em sua construção.
- B) Lógico - um algoritmo pode ser visto como uma dedução lógica controlada. O componente lógico expressa os axiomas usados na computação e o componente de controle determina a maneira como a dedução é aplicada aos axiomas. Tal conceito é base para a programação lógica.

- C) Serial ou paralelo - algoritmos são geralmente assumidos por serem executados instrução à instrução individualmente, como uma lista de execução, o que constitui um algoritmo serial. Tal conceito é base para a programação imperativa. Por outro lado existem algoritmos executados paralelamente, que levam em conta arquiteturas de computadores com mais de um processador para executar mais de uma instrução ao mesmo tempo. Tais algoritmos dividem os problemas em sub-problemas e o delegam a quantos processadores estiverem disponíveis, agrupando no final o resultado dos sub-problemas em um resultado final ao algoritmo. Tal conceito é base para a programação paralela. De forma geral, algoritmos iterativos são paralisáveis; por outro lado existem algoritmos que não são paralisáveis, chamados então problemas inerentemente seriais.
- D) Determinístico ou não-determinístico - algoritmos determinísticos resolvem o problema com uma decisão exata a cada passo enquanto algoritmos não-determinísticos resolvem o problema ao deduzir os melhores passos através de estimativas sob forma de heurísticas.
- E) Exato ou aproximado - enquanto alguns algoritmos encontram uma resposta exata, algoritmos de aproximação procuram uma resposta próxima a verdadeira solução, seja através de estratégia determinística ou aleatória. Possuem aplicações práticas sobretudo para problemas muito complexos, do qual uma resposta correta é inviável devido à sua complexidade computacional.
- 

## Classificação por paradigma

Pode-se classificar algoritmos pela metodologia ou paradigma de seu desenvolvimento, tais como:

- A) Divisão e conquista - algoritmos de divisão e conquista reduzem repetidamente o problema em sub-problemas, geralmente de forma recursiva, até que o sub-problema é pequeno o suficiente para ser resolvido. Um exemplo prático é o algoritmo de ordenação merge sort. Uma variante dessa metodologia é o decremento e conquista, que resolve um sub-problema e utilização a solução para resolver um problema maior. Um exemplo prático é o algoritmo para pesquisa binária.
- B) Programação dinâmica - pode-se utilizar a programação dinâmica para evitar o re-cálculo de solução já resolvidas anteriormente.
- C) Algoritmo ganancioso - um algoritmo ganancioso é similar à programação dinâmica, mas difere na medida que as soluções dos sub-problemas não precisam ser conhecidas a cada passo, uma escolha gananciosa pode ser feita a cada momento com o que até então parece ser mais adequado.
- D) Programação linear - são problemas de otimização nos quais a função objetivo e as restrições são todas lineares.

- E) **Redução** - a redução resolve o problema ao transformá-lo em outro problema. É chamado também transformação e conquista.
- F) **Busca e enumeração** - vários problemas podem ser modelados através de grafos. Um algoritmo de exploração de grafo pode ser usado para caminhar pela estrutura e retornar informações úteis para a resolução do problema. Esta categoria inclui algoritmos de busca e backtracking.
- G) **Paradigma heurístico e probabilístico** - algoritmos probabilísticos realizam escolhas aleatoriamente. Algoritmos genéticos tentam encontrar a solução através de ciclos de mutações evolucionárias entre gerações de passos, tendendo para a solução exata do problema. Algoritmos heurísticos encontram uma solução aproximada para o problema.
- 

### **Classificação por campo de estudo**

Cada campo da ciência possui seus próprios problemas e respectivos algoritmos adequados para resolvê-los. Exemplos clássicos são algoritmos de busca, de ordenação, de análise numérica, de teoria de grafos, de manipulação de cadeias de texto, de geometria computacional, de análise combinatória, de aprendizagem de máquina, de criptografia, de compressão de dados e de interpretação de texto.

---

### **Classificação por complexidade**

Alguns algoritmos são executados em tempo linear, de acordo com a entrada, enquanto outros são executados em tempo exponencial ou até mesmo nunca terminam de serem executados. Alguns problemas possuem múltiplos algoritmos enquanto outros não possuem algoritmos para resolução.

# Laços (Ciclos) na Linguagem MQL4

---

O controle de fluxo normal do programa que você escreve em MQL4 (e em outras línguas também) é executado do começo ao fim, passo a passo. Um passo é uma linha do código que diz o computador para fazer algo. Por Exemplo

```
Print("Hello World");  
return 0;
```

Um ponto e vírgula no fim da instrução é uma parte crucial da sintaxe mas geralmente fácil de esquecer, e isso que geralmente representa cerca de 90% dos erros. Mas a execução coma para baixo não é o único caso e ela tem duas exceções. Essas exceções são os laços e as decisões (condições). Nos programas que você escreve como um ser humano, você decide o que fazer de acordo com determinada circunstância. Nestes casos os fluxos do controle possibilitam saltar de uma parte do programa a outra. As palavras chaves que possibilitam tais saltos são chamadas de indicações de controle. Tais controles consistem em laços e em decisões.

---

## Laços (Ciclos ou Loops)

Laços que fazem com que uma parte de seu programa seja repetida em um determinado número de vezes. E esta repetição continua quando enquanto alguma circunstância for verdadeira verdadeira e termina quando se torna falsa. Quando o laço termina ele passa o controle à do programa a próxima instrução após o término do laço. Em MQL4 há dois tipos de laços:

- 1) Laço **for**
  - 2) Laço **while**
- 

## Laços **for**

O laço **for** é considerado o laço o mais fácil porque todos seus elementos do controle são encontrados em um só lugar. O laço **for** executa uma parte do código um número fixo de vezes. Para o exemplo:

```
int j;  
for(j=0; j<15; j++)  
    Print(j);
```

Como o **for** funciona?



Bem ele consiste em uma palavra-chave, o **for**, seguido por três instruções separadas por vírgula que se encontram entre chaves: "`for(j=0; j<15; j++)`". Estas três expressões são :

```
j=0 - expressão da iniciação
j<15 - expressão do teste
j++ - expressão do increment
```

O corpo do laço é o código a ser executado o número fixo de vezes (15 neste caso, de 0 ate 14, lembre-se que ele faz o laço enquanto j for menor que 15) e neste caso seria "`Print(j);`"

Nota: entre o a instrução **for** e o passo dentro do laço não se usa ponto e vírgula. Isso é porque o **for** e a intrução do corpo do laço são considerados ´como sendo uma só instrução no programa. A **expressão da iniciação** está executada somente uma vez, quando o laço começa. E sua finalidade dar à variável do laço um valor inicial (0 em nosso exemplo). Você pode declarar o variável fora do laço(como vimos antes) ou você pode fazer a declaração dentro dos parênteses do laço, assim:

```
int j;
for(int j=0; j<15; j++)
    Print(j);
```

As duas linhas precedentes do código são iguais, a não ser pela validade de cada um variável (você saberá mais sobre a declaração e validade de variáveis na lição das variáveis). O método de declaração exterior da variável faz cada linha no bloco do código faz com que cada linha saiba sobre esta variável, enquanto que a declaração interna faz com que somente o laço saiba da existência da variável. Você pode usar mais que uma expressão da iniciação no laço **for**, separando cada uma com vírgula, assim:

```
int j;
int i;
for(j=0, i=0; j<15; j++)
    Print(j, " ", i);
```

**A expressão do teste:** testa sempre uma expressão relacional na qual se use operadores relacionais (consulte por favor aos operadores relacionais na lição correspondente). Avaliado pelo laço cada vez que o laço executado para determinar se o laço continua ou para. Continuará se o resultado da expressão for verdadeiro e parará se ele falso. Em nosso exemplo o laço do corpo continuará imprimindo "`Print(j, " ", i);`" enquanto J<15, ou seja caso j seja igual a 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 ou 15. E quando j seja 15, o laço para e o controle passa à próxima instrução que seguir apos o laço.

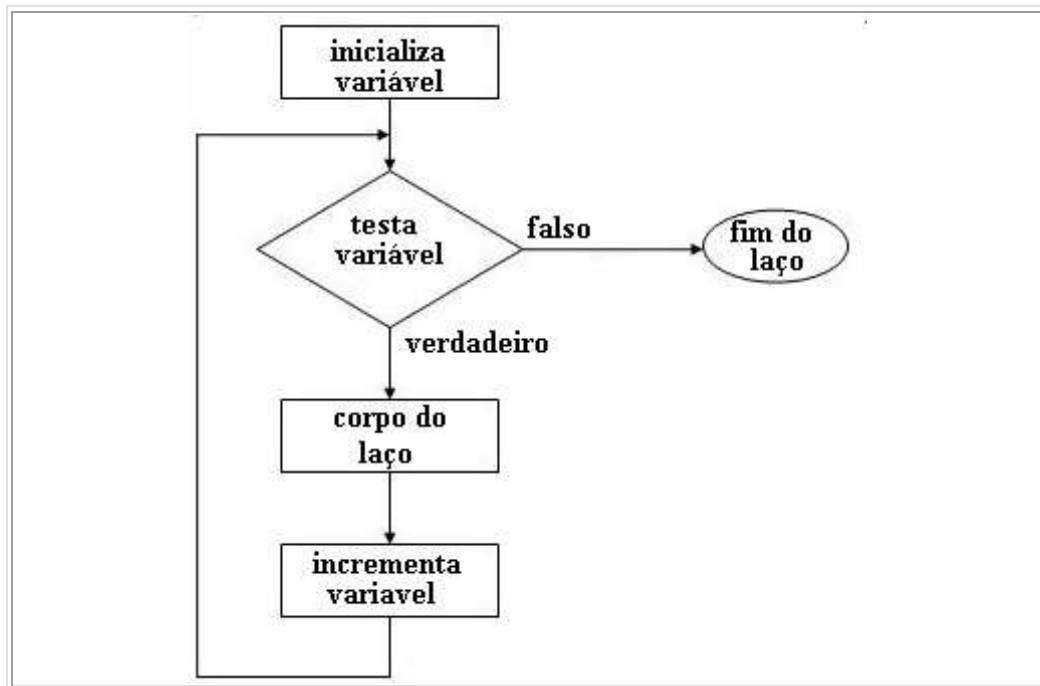
**A expressão do incremento:** A expressão do incremento muda o valor da variável do laço (j em nosso exemplo) aumentando seu valor em uma (1) unidade a cada vez que o computador termina o laço.

Como a expressão da iniciação, na expressão do incremento você pode usar mais de uma expressão do incremento no para o laço separando o com vírgula, assim

```
int j;  
int i;  
for(int j=0; j<15; j++, j++)  
    Print(j, " ", i);
```

Porém você pode somente usar uma expressão do teste.

Fluxograma de funcionamento do laço **for**



Uma outra observação sobre a expressão do incremento, é não somente pode aumentar a variável do laço, mas pode executar e operação de decréscimos, assim:

```
for(int j=15; j>0; j--)  
    Print(j);
```

No nosso exemplo usamos somente um passo no corpo do laço: "**Print(j);**". Quando você for usar mais de um passo dentro do laço, você deve colocar todos os passos entre chaves. Para fixar bem, tudo que estiver entre chaves é considerado um bloco agrupado no programa.

```
for(int j=15; j>0; j--)  
{  
    Print(j);  
    PlaySound("alert.wav");  
}
```

No código que acima o corpo do laço contém duas instruções, o programa executará a primeira instrução e depois a segunda a cada execução do laço. Não se esqueça de pôr um ponto e vírgula no fim de cada instrução.

**A instrução `break`:** a palavra chave `break` quando o presente em qualquer parte do laço fará com que a execução do laço termine e o controle seja passado a próxima instrução após o laço. Por exemplo

```
for(int j=15; j>0; j--){
    if(i==10)
        break;
    Print(j);
}
```

O exemplo acima executará o laço até que `j` alcance 10, pois quando atingir este valor a palavra chave `break` causará uma ruptura e terminará o laço. O código produzirá estes valores: 0.1.2.3.4.5.6.7.8.9. e sairá do laço sem passar pelos demais valores.

**A instrução `continue`:** faz com que o controle do laço salte para o incremento da variável e inicie o laço novamente. Por exemplo:

```
for(int j=15; j>0; j--){
    if(i==10)
        continue;
    Print(j);
}
```

O exemplo acima executará o laço completo, porém somente imprimirá estes valores: 0.1.2.3.4.5.6.7.8.9.11.12.13.14

**Uma última observação sobre o laço `for`:** Você pode deixar uma ou todas as expressões dentro para o laço sem serem preenchidas "`for(;;)`", porém, neste caso, o laço será executado eternamente.

---

## Laços `while`

O laço `for` é geralmente usado no caso você sabe quantas vezes o laço será executado. O que acontece se você não souber quantas vezes você quer executar o laço? Quando o laço deveria parar de ser executado?. Para estas ocasiões você usará o laço `while` com uma condição de teste. Mas não tem expressões da iniciação ou do incremento. Este é um exemplo:

```
int i=0;

while(i<15)
{
    Print(i);
    i++;
}
```

No exemplo acima você observará os passos:

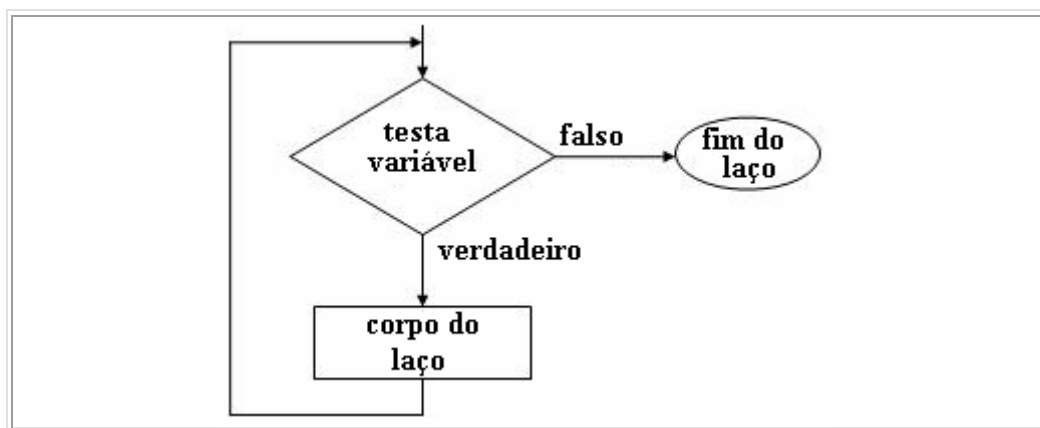
1) A variável do laço tem declarado e inicialização antes do laço, e você não pode declará-la ou idealizá-la dentro dos parênteses do laço (como é permitido fazer no laço `for`).

2) A indicação de `i++` aqui não é a expressão do incremento como você pode pensar, mas o corpo do laço deve conter alguma indicação que muda a variável do laço, se não o laço nunca terminaria.

### Como o exemplo acima trabalha?

A instrução `while` contém somente a expressão do teste, e ao examiná-la em cada laço, se for verdadeiro o laço continuaria e se for falsa o laço terminaria e a passagem do controle do programa iria para a próxima instrução após o laço. No exemplo o laço executará até que `i` alcance o valor 16.

Fluxograma de funcionamento do laço `while`



Tudo que eu disse antes para o laço `for`, aqui no laço `while`, tem aspectos similares

- 1) Você pode usar a instrução `break` e `continue` com o mesmo efeito.
- 2) Você pode colocar vários passos dentro do laço, desde que todas as instruções estejam entre chaves.
- 3) "`while(true)`" tem o mesmo efeito de "`for(;;)`"

## Como tomar decisões na linguagem MQL4

---

Até aqui, nós vimos que os laços são uma de duas maneiras que nós nos usamos mudar o fluxo normal da execução de programa. A segunda maneira são as decisões. As decisões em um programa causam um salto de uma só vez a uma parte diferente do programa, dependendo do valor de uma expressão. Estes são os tipos das instruções das decisões disponíveis em MQL4:

- `if` - única condição
  - `switch` - uma condição dirigente para cada situação
- 

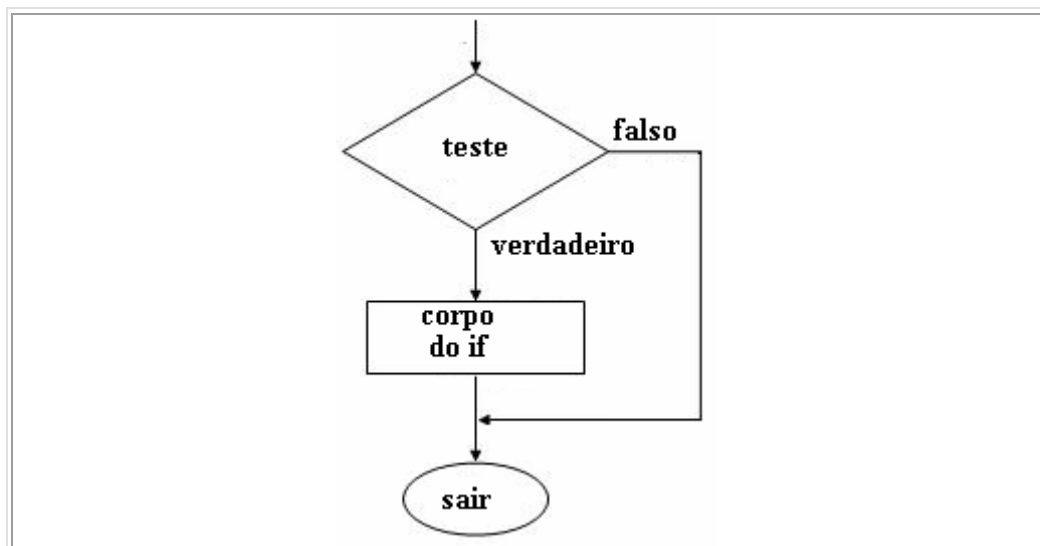
### Instrução `if ...`

A instrução `if` é a mais simples instrução de decisão, eis um exemplo:

```
if( x < 100 )  
    Print(j);
```

Aqui se a palavra chave seguida por parênteses, dentro dos parênteses a expressão do teste (`x < 100`), quando o resultado da expressão do teste for verdadeiro o corpo do `if` será executado: `Print(j);`, e se for falso, a passagem do controle irá para a próxima instrução após o bloco do `if`.

Fluxograma do da decisão `if`



Várias instruções (passos) no corpo do `if`: Como nos laços, o corpo do `if` pode ter em mais do que uma instrução, a única condição é que esteja entre chaves. Por exemplo:

```
if( x < 100 ) {  
    Print(j);  
    PlaySound("alert.wav");  
}
```

Observe o símbolo "==" na expressão do teste; é um dos operadores que relacionais você estudou na lição de operações e de expressões. Esta é uma fonte dos muitos dos erros que você pode cometer quando escreve seu código fonte, quando você se esquece e se usa do operador de atribuição "=" no lugar do de comparação "==".

Os laços e as estruturas da decisão podem ser o bloco interior um outro bloco; você pode aninhar o `if` dentro dos laços, laços dentro de `if`, `if` dentro do `if`, e assim por diante. Está aqui um exemplo:

```
for(int i=2; i<10; i--)  
    if(i%2==0)  
    {  
        Print("Este é um número par");  
        PlaySound("warning.wav");  
    }
```

No exemplo acima, temos uma estrutura de aninhamento, um bloco `if` no interior de um bloco `for` (laço). Você observará que não há nenhuma chave no do corpo do laço, isto é porque se a instrução `for` e as instrução `if` formam um só corpo, como se fossem consideradas ser uma única instrução, isso porque a única instrução do laço `for` é o bloco `if`.

---

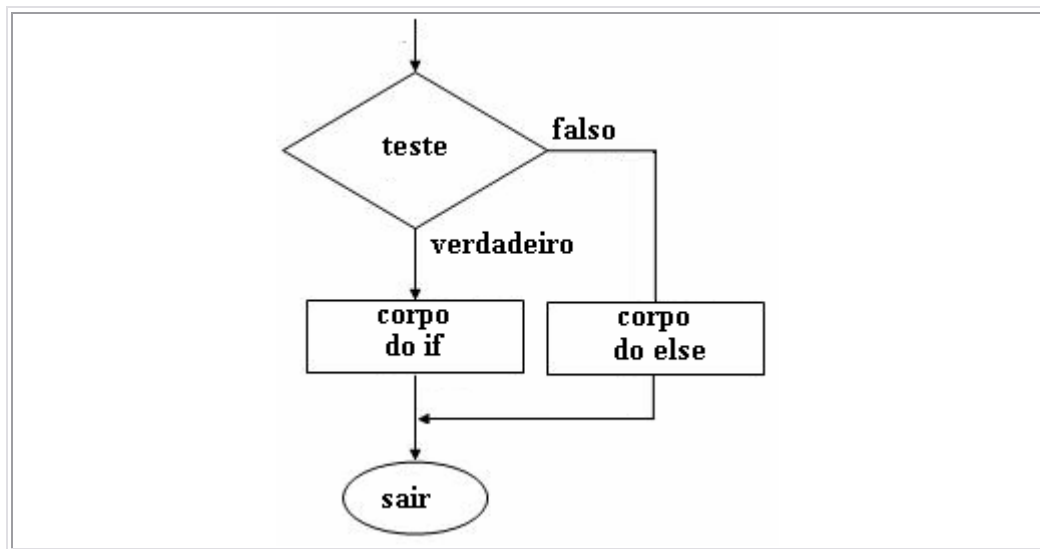
### Instrução if ... else ...

Agora pense, que se uma condição a ser testada tenha um conjunto de passos a ser seguido para a quando for avaliada como verdadeira e outro conjunto de passos para quando for avaliada como falsa. Consiste a instrução `if` seguida pelo seu bloco para avaliação verdadeira, a seguir a palavra chave `else` seguida por uma outra instrução ou por um bloco das instruções. Como este exemplo:

```
if(PrecoAtual>MenorPreco)  
    Print("Este preço é maior, esqueça...");  
else  
    Print("Esta preço pode ser considerado...");
```

Se expressão na instrução `if` for verdadeira, a mensagem ("Este preço é maior, esqueça...") será mostrada, se não for verdadeiro, então a outra mensagem ("Esta preço pode ser considerado...") é que aparecerá.

Fluxograma do da decisão `if ... else ...`



Novamente aqui, voce pode aninhar blocos. Há um problema potencial no aninhado `if ...` mais as indicações, você puderem inadvertidamente combinar um outro com o erro se. Para resolver este caso você pode fazer uma de duas coisas: 1 que você pode limitou se... se emparelhasse mais com as cintas como isto:

```
for(int i=2; i<10; i--)  
  if(i%2==0)  
  {  
    Print("Este é um número par");  
    PlaySound("Par.wav");  
  }  
else  
  {  
    Print("Este é um número impar");  
    PlaySound("Impar.wav");  
  }
```

Existem alguns erros que podem passar inadvertidos e podem dar muita dor de cabeça a um programador, veja os 2 codigos abaixo

```
if(A>B)  
  Print("A é maior que B.");  
if(A==B)  
  Print("A é igual a B.");  
else  
  Print("A é diferente de B");
```

```
if(A>B)  
{  
  Print("A é maior que B.");  
  if(A==B)  
    Print("A é igual a B.");  
  else  
    Print("A é diferente de B");  
}
```

veja que no segundo caso, temos um bloco para o primeiro `if`, com mais de uma instrução, e a comparação de igualdade entre A e B nunca será verdadeiro, pois o bloco só será executado se a primeira condição `if` for verdadeira, só seja se `A > B` então A nunca será igual a B. Portanto, muita atenção quando escrever seus programas.

---

## Instrução `switch`

Se você tiver uma árvore grande de decisões, e todas as decisões dependerem do valor da mesma variável, você pode usar uma indicação `switch`. Está aqui um exemplo:

```
switch(x)
{
    case 'A':
        Print("x é igual a A");
        break;
    case 'B':
    case 'C':
        Print("x é igual a B ou C");
        break;
    default:
        Print("x não é igual nem a A, nem a B e nem a C");
        break;
}
```

No exemplo acima a palavra chave `switch` é seguido por parênteses, dentro dos parênteses que você encontrará a constante a ser avaliada pelo `switch`, esta constante pode ser um inteiro, um caráter ou uma expressão constante. A expressão constante não deve incluir uma variável. por exemplo: `case X+Y:` é a constante inválida para `switch`.

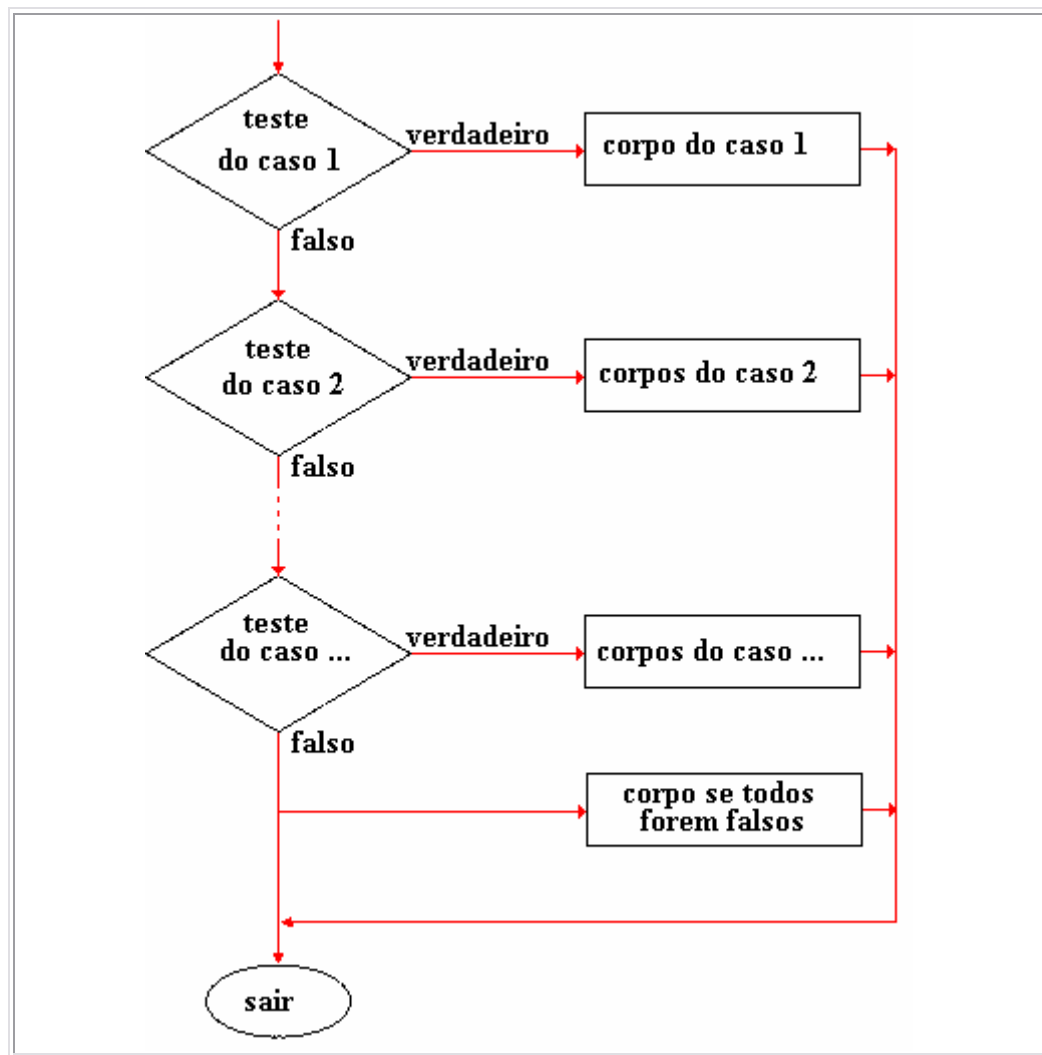
Como o exemplo acima funciona?

A instrução `switch` combina o x com uma das constantes dos casos (`case`). Seria como se imaginasse vários `ifs` um após ao outro. O raciocínio seria assim :

- No `case x=='A'` o programa imprimirá a "x é igual a A" e a instrução `break` fará com que o programa passe o controle para a próxima instrução após o bloco `switch`.
- No `case x=='B'` dos `case x=='C'`, o programa imprimirá a "x é igual a B ou C" e a instrução `break` fará com que o programa passe o controle para a próxima instrução após o bloco `switch`.
- No caso de `x!= "algumas das constantes dos casos"` o programa imprimirá a "x não é igual nem a A, nem a B e nem a C" e a instrução `break` fará com que o programa passe o controle para a próxima instrução após o bloco `switch`.



Fluxograma do da decisão `switch`



# Funções na Linguagem MQL4

## Que são funções?

A função é um conjunto de comandos agrupados em um bloco que recebe um nome e através deste nome pode ser executado. Um programa na linguagem MQL4 é composto por **funções**, isto é, porções menores de código que realizam determinadas tarefas, e por **dados**, ou seja, variáveis ou tabelas que são inicializadas antes do início do programa. Existe três funções especiais em MQL4

- **init()** : executada quando seu Indicador ou Consultor é executado pela primeira vez, aqui você define todas as coisas que você quer dentro do sistema, que provavelmente não dependem de uma inicialização, por exemplo, aqui você pode mandar ler um arquivo com os dados da ultima excussão. Essa função é obrigatória em Indicadores ou Consultores porem não é utilizada em scripts.
- **deinit()** : executado quando se executa pela ultima vez o seu Indicador ou Consultor, por exemplo quando você retira ele do gráfico ou quando fecha o MT. Aqui pode guardar dados em um arquivo se assim desejar para serem recuperados na próxima excussão. Essa função é obrigatória em Indicadores ou Consultores porem não é utilizada em scripts.
- **start()** : é a função principal do seu programa, no caso de Indicador ou Consultor ela é chamada pelo MetaTrade a cada modificação nas cotas atuais do par cujo gráfico o Indicador ou Consultor estiverem anexado. Já no caso do script ela é chamada somente quando você manda executar o script, ao terminar de executar o código desta função, termina o script. Se necessitar usar o script novamente tem de requisitar sua excussão manualmente. Esta função é obrigatória em Indicadores, Consultores e Scripts, se não colocar ela seu código nunca será executado.

Nota: ser obrigatório significa que você deve definir a função, mesmo que ela não contenha nenhum código dentro.

As funções são organizadas em módulos agrupados nos arquivos fonte. Em MQL4, voce pode organizar seu código em arquivos separados ligando cada arquivo em um modulo principal. Toda função possui uma lista de parâmetros, um corpo e, eventualmente, um valor de retorno. O corpo pode conter declarações de variáveis locais, ou seja, variáveis que são ativadas quando a execução alcançar o corpo da função.

---

## Porque usar funções?

- Para permitir o reaproveitamento de código já construído (por você ou por outros programadores);
  - Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
  - Para permitir a alteração de um trecho de código de uma forma mais rápida.
- Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
  - Para facilitar a leitura do programa-fonte de uma forma mais fácil;
  - Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.
- 

## Formato geral de uma função em MQL4?

```
tipo_da_funcao NomeDaFuncao(Lista_de_Parametros)
{
    // declaração de variáveis

    // corpo da função

    // valor de retorno
}
```

## Parâmetros das funções

A Lista\_de\_Parametros, também é chamada de Lista\_de\_Argumentos, é opcional. Toda função em MQL4 retorna um valor que você define como sendo um tipo de dados dos que já estudamos, porém, existe um tipo de dado especial para especificar que uma função não retorna um valor, o `void`. Void na verdade é uma extensão do tipo `int`

```

int C=0;

// retorna um valor inteiro
int CalculaFatoria(int Numero)
{
    if (Numero>0) { Print("Erro no Fatorial argumento negativo");
return(0); }
    if (Numero>1)
        return (CalculaFatoria(Numero-1));
    else
        return(1);
}

// não retorna nenhum valor
void InformaValorDeC()
{
    Print("Valor de C=",C);
    // embora não retorne valor
    // nada impede você de declarar
    // um valor de retorno
    return(0); // não é necessário em funções tipo void porem
usar não causa erro nenhum
}

```

A fim de tornar mais amplo o uso de uma função, a linguagem MQL4 permite o uso de parâmetros. Estes parâmetros possibilitam que se defina sobre quais dados a função deve operar. A função `CalculaFatoria(int Numero)`, por exemplo, recebe como parâmetro um `Numero` para calcular o fatorial, permitindo que se defina seu comportamento a partir deste valor. Para definir os parâmetros de uma função o programador deve explicitá-los como se estivesse declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas, assim

```

int C=0;

// retorna um valor inteiro
int Soma(int A, int B, int C)
{
    return(A+B+C);
}

```

Note que no caso das funções, cada parâmetro declarado deve conter sua palavra chave de identificação de tipo, senão o compilador ira reclamar.

Os parâmetros da função na sua declaração são chamados parâmetros formais. Na chamada da função os parâmetros são chamados parâmetros atuais. Os parâmetros são passados para uma função de acordo com a sua posição. Ou seja, o primeiro parâmetro atual(da chamada) define o valor o primeiro parâmetro formal (na definição da função, o segundo parâmetro atual define o valor do segundo parâmetro formal e assim por diante. Os nomes dos parâmetros na chamada não tem relação com os nomes dos parâmetros na definição da função

---

## Chamando (executando) as funções

Já sabemos como criar uma função, e agora aprenderemos como utilizá-la. Se seu código tem poucas linhas e estas ou se estas linhas são utilizadas em um só lugar, não há necessidade de criar mais funções, coloque tudo no mesmo lugar, porém se tiver muitas linhas ou se determinadas partes do código necessitam ser chamadas em partes diferentes do código, então divida seu código em funções. As funções tornam o código mais legível e mais fácil de entender.

Para chamar uma função necessitamos somente colocar o nome da função e entre parênteses os seus argumentos separados por vírgula, se não tiver argumentos deve se abrir e fechar parênteses simplesmente. Como no exemplo abaixo

```
int C=0;

// calcula um fatorial
int CalculaFatoria(int Numero)
{
    if (Numero>0) { Print("Erro no Fatorial argumento negativo");
return(0); }
    if (Numero>1)
        return (CalculaFatoria(Numero-1));
    else
        return(1);
}

// imprime o valor de C
void InformaValorDeC()
{
    Print("Valor de C=",C);
}

// imprime o valor de C
int start()
{
    int s;

    s = 2;
    C = CalculaFatoria(s);           // chama a função do fatorial
    InformaValorDeC();               // chama a função de impressao
    C = C + CalculaFatoria(3);       // chama a função do fatorial
    InformaValorDeC();               // chama a função de impressao

    retur(0);
}
```

---

## Recursividade

Na linguagem MQL4, assim como em muitas outras linguagens de programação, uma função pode chamar a si própria. Uma função assim é chamada função recursiva. Todo cuidado é pouco ao se fazer funções recursivas. A primeira coisa a

se providenciar é um critério de parada. Este vai determinar quando a função deverá parar de chamar a si mesma. Isto impede que a função se chame infinitas vezes o que poderá ocasionar um erro em tempo de execução, chamado, estouro de pilha, ou seja não existe mais memória para trabalhar com a função. Há certos algoritmos que são mais eficientes quando feitos de maneira recursiva, mas a recursividade é algo a ser evitado sempre que possível, pois, se usada incorretamente, tende a consumir muita memória e ser lenta. Lembre-se que memória é consumida cada vez que o computador faz uma chamada a uma função. Com funções recursivas a memória do computador pode se esgotar rapidamente. Um exemplo desta função é o calculo do fatorial.

```
int CalculaFatoria(int Numero)
{
    if (Numero>0) { Print("Erro no Fatorial argumento negativo");
return(0); }
    if (Numero>1)
        return (CalculaFatoria(Numero-1));
    else
        return(1);
}
```

---

### Endentação ou Identação ou ainda Edentação

é um termo aplicado ao código fonte de um programa de computador para indicar que os elementos hierarquicamente dispostos têm o mesmo avanço relativamente à posição (y,0) ((linha, coluna)). A endentação tem um papel meramente "estético" (na maioria das linguagens) tornando a leitura do código fonte mais fácil (o que designamos por read-friendly). A verdadeira mais valia deste processo é visível em arquivos de código fonte extensos, não se fazendo sentir a sua necessidade em arquivos pequenos (relativamente ao número de linhas) Para qualquer programador deve ser um critério a ter em conta, principalmente, por aqueles que pretendam partilhar o seu código com outros. Em palavras simples, endentação é a organização do código fonte. Entre cada "{" e "}", por exemplo, tabulamos as linhas. Assim pode-se identificar melhor o que nesse bloco está contido. Auxilia muito na busca de erros ou implementações. Em MQL4 a endentação não é obrigatória, que dizer, o compilador do código MQL4 não irá diferenciar se o programa será escrito com endentação:

Qual destes códigos para a mesma função é mais legível para você:

```
// com edentação
int CalculaFatoria(int Numero)
{
    if (Numero>0) { Print("Erro no Fatorial argumento negativo");
return(0); }
    if (Numero>1)
        return (CalculaFatoria(Numero-1));
    else
        return(1);
}
```

```
// sem edentação
int CalculaFatoria(int Numero)
{
if (Numero>0) { Print("Erro no Fatorial argumento negativo");
return(0); }
if (Numero>1)
return (CalculaFatoria(Numero-1));
else
return(1);
}
```

```
// sem edentação
int CalculaFatoria(int Numero){ if (Numero>0) { Print("Erro no
Fatorial argumento negativo"); return(0); } if (Numero>1) return
(CalculaFatoria(Numero-1)); else return(1); }
```

Para o compilador é o mesmo código, mas visualmente falando você consegue identificar no primeiro código o que faz parte da função e de cada `if` ou `if else`. Em todos os casos a diferença visual mostra como você organiza seu código.

# Variáveis na Linguagem MQL4

---

## Para que servem as variáveis?

Como eu lhe disse antes, as variáveis são os nomes que damos às seções da memória em que os dados podem ser armazenados. Quando você nomeia uma variável e lhe atribui um tipo de dados, você na verdade esta reservando uma área de memória para armazenar uma informação (quantos bytes de memória serão utilizados). Quando você informa no seu código o tipo de dado que uma variável terá, na verdade você esta dizendo ao compilador do MQL4 quanta memória será reservada para esta referida variável, pense armazenar um numero ou uma string, não necessariamente se utilizara a mesma quantidade de memória. Quando você atribui um valor a esta área de memória, este valor fica disponível em todo seu programa através deste nome de variável, se você desejar mudar o valor, simplesmente atribua um novo valor a este nome de variável e pronto, a magia de armazenar valores na memória esta completa. A maior vantagem de usar nomes para as variáveis é que você pode atribuir um nome que identifique a qualidade e tipo de informação que determinada região de memória possui. Por exemplo:

```
int NumeroDaConta=0;  
string NomeDaCorretora;
```

Analisaremos agora a primeira linha do nosso exemplo. Através desta linha estamos dizendo ao compilador que necessitamos um bloco de memória com o comprimento de 4 bytes, especificamos também que temos a intenção de armazenar um numero inteiro (isto é sem a parte decimal, em outras palavra numero naturais) e como se não fosse o bastante pedimos para o compilador colocar como valor inicial de nossa variável o numero zero (0). Assim o compilador entenderia esta linha :

int	: uma palavra chave
int	: tipo de dados da variável
int	: uma declaração
NumeroDaConta	: nome de uma variável
=0	: inicialização da variável

No MQL4 temos os seguintes tipos de dados (como ja vimos) e suas respectivas declarações

- Inteiro (`int`)
- Lógico (`bool`)
- Literais ou caractere (`int`)
- Cadeia de Literais (`string`)
- Ponto Flutuante (`double`)
- Cores (`color`)
- Data e Hora (`datetime`)

As palavras chaves estão em cor azul...

---



## Declaração

Declarar uma variável significa introduzi-la ao mundo do seu programa e especificar seu tipo de dados. Usando as palavras chaves que aprendemos em TIPOS de DADOS e descritos acima (`int`, `bool`, `char`, `string`, `string`, `double`, `color` e `datetime`) com o nome que você escolheu à variável. Por exemplo:

```
int MinhaVariavel;  
int A, B, C, D;  
int H=1, Z=2, X=3,  
    Y=4;
```

Aqui você declarou uma variável nomeada `MinhaVariavel` que é um tipo do inteiro (`int`). Antes de declarar esta variável você não pode utilizá-la em seu código. Se você a usar sem declarar o compilador MQL4 ira reclamar e lhe apresentara uma mensagem de erro dizendo que a variável não foi definida, parecia com esta mensagem: **'MinhaVariavel' - variable not defined. 1 error(s), 0 warning(s).**

Na segunda linha declaramos varias variáveis com uma única palavra chave de declaração. Nas linhas 3 e 4 declaramos e inicializamos as variáveis, veja a flexibilidade, declarando uma variável em cada linha você pode colocar o descritor para que ela serve no lado, aumentando o grau de documentação do seu código fonte, Exemplo:

```
int LE=2,    // lucro esperado em pips  
    PM=4;    // Perda Máxima em pips
```

Outra informação importante, MQL4 é sensível ao caso, ouse seja, diferencia maiúsculas de minúsculas em suas declarações então todas as variáveis no exemplo abaixo são diferentes nomes de variáveis

```
int par=2,  
    Par=4,  
    PAr=6,  
    paR=8,  
    PaR=10,  
    pAR=12,  
    PAR=14;
```

Este é um fato que cria muitos erros na programação: Declarar uma variável de uma jeito e usar de outro, combinando maiúsculas com minúsculas.

---

## Inicialização

Inicializar a variável significa atribuir-lhe um valor inicial; Você pode inicializar a variável na mesma linha da declaração, assim

```
int MinhaVariavel = 10;
```

Ou se voce preferir pode declarar a variável em um lugar e idealizá-la em um outro lugar do seu código, assim

```
int MinhaVariavel;  
...  
...  
...  
MinhaVariavel = 10;
```

Porém uma regra nunca deve ser esquecida : **a declaração de uma variável deve vir antes de sua inicialização.**

---

## Validades (escopo) de Uma variável

Há dois espaços para a validade das variáveis: **local** e **global**. Estes espaços dizem ao compilador onde a variável pode ser usada e onde ela é válida. Global, é todo o espaço que o seu código fonte pode ocupar, ou seja todo seu programa. Local é o espaço de um bloco de programa, como por exemplo uma função. Assim variáveis locais são endereços de memória nomeados que apontam para a mesma posição de memória em qualquer parte do programa, o escopo de variáveis locais é limitado ao mesmo nível de aninhamento no qual elas são declaradas, em outras palavras, variáveis locais, são espaços de memória nomeados que valem somente dentro do bloco de programa em que foram declarados (lembre que um bloco deve estar entre chaves). Portanto, variáveis locais não podem ser utilizadas fora do bloco onde foram declaradas.

```
int C=3;           // valida em todo o programa  
  
double MinhaFuncao(double a)  
{  
    int d=2;       // valida somente em MinhaFuncao  
    return (a*d*c);  
}  
  
double OutraFuncao(double a)  
{  
    return (a*d*c); // vai resultar um erro, embora  
                   // c seja valida nesta função  
                   // d não é variável valida aqui  
}  
  
double OutraFuncaoDeNoco(double a)  
{  
    return (a*c);   // Sem erro pois c é valido aqui também  
}
```

Como você vê, "c" é uma variável global pois foi declarada no bloco principal do programa. Tanto "d" quanto "a" são variáveis locais, ou seja somente são validas nos blocos onde elas foram declaradas. Outro fato importante a lembrar é que todas as variáveis globais são automaticamente inicializadas com zero se você não inicializar. Isso significa que estes seriam os valores para estas variáveis ao serem declaradas globais se você não idealizá-las

- Inteiro (int).....: 0
- Lógico (bool).....: False
- Literais ou caractere(int)..: NULL
- Cadeia de Literais (string): "" (vazia)
- Ponto Flutuante (double)....: 0.0
- Cores (color).....: Black
- Data e Hora (datetime).....: 01/01/1970 00:00:00

---

## Variáveis externas

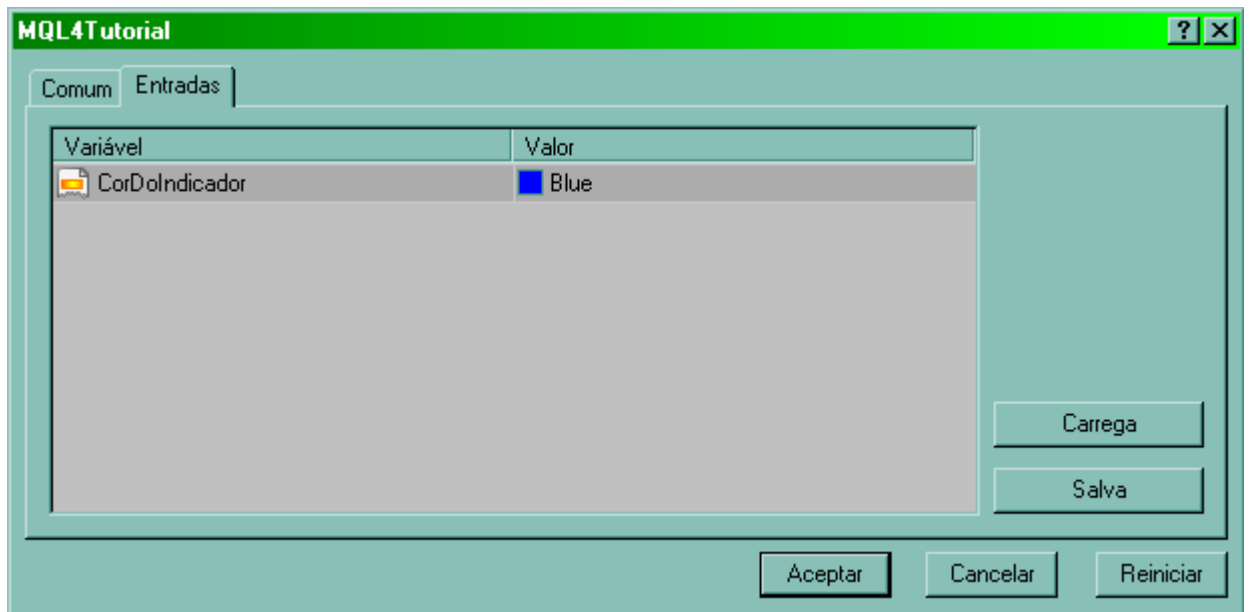
A palavra chave "extern" é utilizada para declarar um tipo especial das variável, todos aqueles tipos de variáveis que devem ser usados para definir entrada de dados ao seu programa, essas variáveis são utilizadas na janela de propriedades do seu Consultor Especialista (Expert Advisor) ou indicador, para que você permita ao usuário especificar alguma configuração especial para que seu sistema utilize. Porexemplo

```
extern color CorDoIndicador=C'0x00,0x00,0xFF'; // blue

int init()
{
    return (0);
}
```

CorDoIndicador aparecera na janela de propriedades do Consultor ou do indicador, permitindo que o usuário mude seu valor antes de iniciar a excussão do seu código. Mas atenção esse tipo de variável, embora possa ser definida em um script, tem o mesmo efeito de uma variável global, pois no script não se apresenta tela de propriedades quando se inicia sua excussão.

Abaixo a imagem da janela de propriedades onde aparece o local de interação com o usuário e seu programa:



## Matrizes

Matriz é uma série de elementos (variáveis) do mesmo tipo alocadas consecutivamente na memória que podem ser referenciados individualmente adicionando-se um índice a um nome único. Isso significa que, por exemplo, podemos guardar 5 valores do tipo `int` sem ter que declarar 5 variáveis diferentes cada uma com um identificado diferente. Ao invés disso, usando uma matriz, podemos guardar cinco valores diferentes do mesmo tipo, `int`, por exemplo, com um identificador único. Por exemplo, uma matriz que contém 5 valores inteiros do tipo `int` chamada `billy` poderia ser representada dessa maneira:

	0	1	2	3	4
billy					
	<code>int</code>	<code>int</code>	<code>in</code>	<code>int</code>	<code>int</code>

Onde cada painel em branco representa um elemento da matriz, que nessa caso são valores inteiros do tipo `int`. Esses são numerados de 0 até 4, pois em matrizes o primeiro índice é sempre 0, independentemente de seu tamanho. Como qualquer outra variável, uma matriz precisa ser declarada antes de ser usada. Uma declaração típica de uma matriz em MQL4 é:

```
tipo nome[elementos];
```

onde `tipo` é um tipo de objeto válido (`int`, `double`, `...`), `nome` é um identificador de variável válido e o campo `elementos`, que está entre colchetes [], especifica quantos desses elementos a matriz contém.

Ao declarar uma matriz de escopo local (dentro de uma função), se não especificarmos nada, não será inicializada, então seu conteúdo será indeterminado

até que guardemos alguns valores nela. Se declararmos uma matriz global (fora de qualquer função) seu conteúdo será inicializado com todos seus elementos preenchidos com zeros. Assim, se declararmos em um escopo global:

```
int billy[5];
```

Todos os elementos de billy serão preenchidos inicialmente com 0. Mas, além disso, quando declaramos uma matriz, temos a possibilidade de atribuir valores iniciais para cada um de seus elementos usando chaves { }. Por exemplo:

```
int billy[5] = {16, 2, 77, 40, 12071};
```

Essa declaração teria criado (na memória) uma matriz como a seguinte

	0	1	2	3	4
billy	16	2	77	40	12071
	int	int	in	int	int

O número de elementos em uma matriz que inicializamos dentro das chaves { } precisa estar de acordo com o tamanho de elementos que declaramos para a matriz dentro dos colchetes [ ]. Por exemplo, no exemplo da matriz billy, nós declaramos que tinha 5 elementos e na lista de valores iniciais dentro das chaves { } nós preenchemos com 5 valores diferentes, um para cada elemento. Devido ao fato disso ser considerado uma repetição inútil, o MQL4 inclui a possibilidade de deixar os colchetes vazios [ ] e o tamanho da matriz será definido pelo número de valores incluídos entre chaves { }:

```
int billy[] = {16, 2, 77, 40, 12071};
```

---

## Acesso aos valores de uma Matriz

Em qualquer parte do programa no qual a matriz é visível, podemos acessar qualquer de seus valores individualmente para leitura ou modificação como se fosse uma variável normal. O formato é o seguinte:

```
Nome[índice]
```

De acordo com os exemplos anteriores no qual billy tinha 5 elementos e cada um desses elementos eram do tipo int, os nomes que podemos usar para nos referenciar a cada elemento são os seguintes:

	Billy[0]	Billy[1]	Billy[2]	Billy[3]	Billy[4]
billy	16	2	77	40	12071
	int	int	in	int	int

Por exemplo, para guardar o valor 75 no terceiro elemento de billy, um comando aceitável seria esse:

```
billy[2] = 75;
```

E, por exemplo, para passar o valor do terceiro elemento de billy para uma variável a, poderíamos digitar:

```
a = billy[2];
```

Sendo assim, para todos os propósitos, a expressão `billy[2]` é como qualquer outra variável de tipo `int`. Note que o terceiro elemento de `billy` é especificado como `billy[2]`, pois o primeiro elemento é `billy[0]`, o segundo é `billy[1]`, e sendo assim, o terceiro é `billy[2]`. Pela mesma razão, seu último elemento é `billy[4]`. Pois se escrevêssemos `billy[5]`, estaríamos nos referenciando ao sexto elemento de `billy` e então excedendo o tamanho da matriz. Em MQL4 é perfeitamente válido exceder o limite válido de índices para uma matriz, o que pode causar problemas pois eles não causam erros de compilação, mas podem causar resultados inesperados ou erros sérios durante a execução. A razão pela qual isso é permitido será vista mais a frente quando começarmos a usar ponteiros.

---

## Matrizes Multi Dimensionais

Matrizes multidimensionais podem ser descritas como matrizes de matrizes. Por exemplo, uma matriz bidimensional pode ser imaginada como uma tabela bidimensional de um tipo de dados concreto uniforme.

jimmy		0	1	2	3	4
	0					
	1					
	2					

`jimmy` representa uma matriz bidimensional de 3 por 5 valores do tipo `int`. O maneira de se declarar essa matriz seria:

```
int jimmy[3][5];
```

e, por exemplo, a maneira de referenciar o segundo elemento verticalmente e o quarto horizontalmente em uma expressão seria:

```
billy[1][3] = 75;
```

jimmy		0	1	2	3	4
	0					

	1				75	
	2					

Matrizes multidimensionais não são limitadas a dois índices (duas dimensões). Elas podem conter quantos índices forem necessários, embora seja raro ter que representar mais que 3 dimensões. Apenas considere a quantidade de memória que uma matriz com muitos índices pode precisar. Por exemplo:

```
int century[100][365][24][60][60];
```

atribui um int para cada segundo contido em um século, que é mais que 3 bilhões de *ints*! Isso consumiria algo em torno de 12000 megabytes de memória RAM se pudéssemos declarar.

Matrizes multidimensionais não são nada mais que uma abstração, pois que podemos obter os mesmo resultados com uma matriz simples simplesmente colocando um fator entre os índices:

```
int jimmy[3][5]; // isso
int jimmy[15];   // é equivalente a isso
//com a única diferença que o compilador nos
//lembra a profundidade de cada dimensão imaginária
```

## Matrizes como parâmetros

Em algum momento podemos precisar passar uma matriz para uma função como parâmetro. Em MQL não é possível passar por valor um bloco de memória completo como parâmetro para uma função, mesmo que esteja ordenado como uma matriz, mas é permitido passar seu endereço. Isso tem quase o mesmo efeito prático e é uma operação muito mais rápida e eficiente. Para se admitir matrizes como parâmetros, a única coisa que precisamos fazer ao declarar a função é especificar no argumento o type base para a matriz, um identificador e um par de colchetes vazios []. Por exemplo, a seguinte função:

```
void MinhaFuncao(int valores[]);
```

**Nota Final :** Matrizes, tanto simples quanto multidimensionais, passadas como parâmetro de funções são uma fonte de erros comum para programadores com pouca experiência.

# Pré processadores na linguagem MQL4

---

## Que são pré-processadores?

Um pré-processador é um programa que recebe texto e efetua conversões léxicas nele. As conversões podem incluir substituição de macros, inclusão condicional e inclusão de outros arquivos. A linguagem de programação MQL4 possui um pré-processador que efetua as seguintes transformações:

- substitui trigrafos por equivalentes.
- concatena arquivos de código-fonte.
- substitui comentários por espaços em branco.
- reage a linhas iniciadas com um caractere de sustenido (#), efetuando substituição de macros, inclusão de arquivos, inclusão condicional e outras operações

O uso de pré-processadores tem vindo a ser cada vez menos comum à medida que as linguagens recentes fornecem características mais abstratas em vez de características orientadas lexicalmente. É certo que o abuso do pré-processador pode dar origem a código caótico. Em MQL4 há quatro diretrizes orientadoras dos pré-processadores:

- Diretiva `define`
- Diretiva `property`
- Diretiva `include`
- Diretiva `library`

---

## A diretiva Define

Define a diretriz orientadora usada para gerar uma constante. A constante é muito como a variável com a diferença de que, uma vez ajustado seu valor, não pode ser alterado mais uma vez e você não pode mudar seu valor em seu código como a variável

```
#define Minha_Constante 100
```

Como você pode observar no exemplo acima não há nenhum símbolo de atribuição (=) mas somente espaço entre o nome constante (Minha\_Constante) e seu valor (100). E você pode observar também que a linha não terminou com ponto e vírgula, mas terminou com um caractere de carriage-return (linha nova). O nome da constante obedece às mesmas regras que você tinha aprendido sobre escolher os nomes dos identificadores (SINTAXE da lição 2), porque no exemplo você não pode começar o nome constante com um número nem não excede 31 caracteres. O valor do índice pode ser qualquer tipo que você quiser. O compilador substituirá cada ocorrência do nome constante em seu código de fonte com o valor correspondente.



Assim você pode usar a constante acima em seu código como aquela:

```
Soma = Minha_Constante * 10;
```

## A diretiva property

Existem algumas constantes predefinidas chamadas "controle de compilação" incluída na língua MQL4, que você pode os ajustar em parâmetros em seu programa, algumas são específicas para indicadores. Abaixo as propriedades que você pode ajustar em seu programa

propriedade	tipo	descrição de uso
link	string	o link da pagina web da companhia ou pagina pessoa de quem desenvolveu o código
copyright	string	companhia ou nome pessoa de quem desenvolveu o código
stacksize	int	tamanho da pilha, é o tamanho de uma determinada reação de memória que o seu programa pode usar para trabalhos com as funções como por exemplo criar variáveis locais
library	void	define que o código que esta sendo escrito faz parte de uma biblioteca
indicator_chart_window	void	diz ao compilador que o traçado gráfico que o indicador fará devesse fazer parte da mesma janela onde estão sendo plotadas as cotações
indicator_separate_window	void	diz ao compilador que o traçado gráfico que o indicador fará devesse fazer parte de uma janela separada da janela onde estão sendo plotadas as cotações
indicator_buffers	int	numero de penas de traçados gráficos no indicador, um indicador pode ter no máximo 8 buffers de dados, ou seja 8 matrizes onde ele colocara seus cálculos, estas matrizes contem os dados que darão origem as curvas e símbolos traçados pelo indicador no gráfico
indicator_minimum	double	valor mínimo que será mostrado no gráfico para o indicador, numero abaixo deste não serão plotados pois ficam fora do gráfico
indicator_maximum	double	valor máximo que será mostrado no gráfico para o indicador, numero acima deste não serão plotados pois ficam fora do gráfico
indicator_colorN	color	define a cor da pena de desenho de cada um dos 8 buffers utilizados pelo indicador
indicator_widthN	int	define a espessura da linha da pena de desenho de cada um dos 8 buffers utilizados pelo indicador
indicator_styleN	int	define a estilo da linha as ser plotada pela pena de desenho de cada um dos 8 buffers

		utilizados pelo indicador
<code>indicator_levelN</code>	<code>int</code>	define a linha de nível do indicador, essas linha aparecem na horizontal na janela onde o indicador é plotado, com se fosse uma linha separadora de valores no gráfico
<code>indicator_levelcolor</code>	<code>color</code>	define a cor da linha de nível do indicador, essas linha aparecem na horizontal na janela onde o indicador é plotado, com se fosse uma linha separadora de valores no gráfico
<code>indicator_levelwidth</code>	<code>int</code>	define a espessura da linha de nível do indicador, essas linha aparecem na horizontal na janela onde o indicador é plotado, com se fosse uma linha separadora de valores no gráfico
<code>indicator_levelstyle</code>	<code>int</code>	define o estilo da linha de nível do indicador, essas linha aparecem na horizontal na janela onde o indicador é plotado, com se fosse uma linha separadora de valores no gráfico
<code>show_confirm</code>	<code>void</code>	define que deve aparecer a janela de confirmação de excussão do código quando ele começar a ser executado
<code>show_inputs</code>	<code>void</code>	mostra na janela terminal, tabulação experts, as configurações das variáveis externas que o código esta usando

## A diretiva include

Para programas escritos emMQL4, antes da compilação propriamente dita, há um processo que chamamos de pré-compilação. O papel desta diretiva é instruir o pré-processador (parte do compilador responsável pelo pré-processamento) para incluir, na hora da compilação, um arquivo especificado. Sua forma geral é:

```
#include "win32.h"
```

No exemplo acima você diz ao compilador para abrir o arquivo "win32.h" e ler todo seu conteúdo e copilá-lo no mesmo lugar em que encontrou a diretiva `include`. No exemplo que acima você incluiu o nome do arquivo entre aspas duplas, esse é o meio de você especificar ao compilador para usar o diretório padrão para buscar o arquivo (geralmente, <diretório do MetaTrader>\experts\include). Você pode incluir arquivos localismos em outros diretórios, porem a pesquisa sempre começa no diretório padrão de trabalho desta diretiva no MetaTrader. cosidere os seguintes diretorios no MetaTrader

```
C:\Arquivos de programas\MetaTrader 4
C:\Arquivos de programas\MetaTrader 4\experts
C:\Arquivos de programas\MetaTrader 4\experts\include
C:\Arquivos de programas\MetaTrader 4\experts\files
C:\Arquivos de programas\MetaTrader 4\experts\indicators
C:\Arquivos de programas\MetaTrader 4\experts\library
C:\Arquivos de programas\MetaTrader 4\logs
C:\Arquivos de programas\MetaTrader 4\Meu Diretorio
```

```
\\ na proxima linha voce diz que o arquivo esta em
\\C:\Arquivos de programas\MetaTrader 4\experts\include
#include "win32.h"

\\ na proxima linha voce diz que o arquivo esta em
\\C:\Arquivos de programas\MetaTrader 4\experts
#include "..\win32.h"

\\ na proxima linha voce diz que o arquivo esta em
\\C:\Arquivos de programas\MetaTrader 4\experts\files
#include "..\files\win32.h"
```

---

## A diretiva import

Algumas vezes, você necessita rotinas criadas em outros aplicativos que não seja o MetaTrader (por exemplo, alguma função específica do Windows) ou em outros programas compilados do próprio MetaTrader. Para isto você usa a diretiva `import`, com ela você diz ao compilador que com as definições que estão dentro do `import`, você quer chamar funções em outros aplicativos. Por exemplo:

```
#import "user32.dll"
    int MessageBoxA(int hWnd,
                    string lpText,
                    string lpCaption,
                    int uType);
    int MessageBoxExA(int hWnd,
                     string lpText,
                     string lpCaption,
                     int uType,
                     int wLanguageId);

#import "stdlib.ex4"
```

Na verdade, você simplesmente esta dizendo ao computador para chamar as funções que você define apos a diretiva em outro código executável.

No primeiro caso você quer chamar as funções `MessageBoxA` e `MessageBoxExA` na biblioteca do windows. Em arquivos DLL voce tem de definir como as funções serão chamadas pelo MT

No segundo caso você quer chamar as funções em executável do próprio MetaTrader chamado `stdlib.ex4`. Em arquivos `ex4` que são os arquivos executáveis dos códigos digitados no MetaEditor e compilados, as funções já estão prontas para uso, nada impede você de defini-las assim:

```
#import "stdlib.ex4"

string ErrorDescription(int error_code);
int    RGB(int red_value,int green_value,int blue_value);
bool   CompareDoubles(double number1,double number2);
string DoubleToStrMorePrecision(double number,int precision);
string IntegerToHexString(int integer_number);
```

## Como executar um programa criado com a linguagem MQL4

---

Bom, agora já temos as bases da linguagem MQL4. Chegamos ao ponto de realmente começarmos a programar e, com isso, deixar que o MetaTrader execute o trabalho pesado por nós. Porém, antes de começarmos realmente a programar, decidi revisar a maneira de ativar os códigos compilados (os executáveis) no MetaTrader.

Começaremos especificando, como funcionam os diretórios criados quando o MetaTrade é instalado, não todos, mas somente alguns, os que sejam mais importantes para nosso trabalho. Abaixo mostro a árvore de diretórios criados pelo MetaTrader, mas somente mostro os diretórios que mais no interessam

Diretório	Explicação
experts	Diretório base de códigos do MetaTrader, aqui estará o código e a versão executável do seu Consultor especialista, somente os consultores compilados (com extensão ex4) deste diretório ficam disponíveis para você utilizar no MetaTrader. (**)
files	fica dentro do diretório experts Todos os arquivos que você abrir com a função open, por definição, serão criados ou devem estar nesta pasta para que você possa gravá-los. (*)
include	fica dentro do diretório experts Diretório base para a diretiva <code>include</code> , aqui estarão os arquivos de inclusão. (*)
indicators	fica dentro do diretório experts Neste diretório você colocará e também estará o código executável de seus operadores, todos os indicadores que se encontrarem neste diretório e que estiverem compilados, podem ser utilizados nos gráficos do MetaTrader. (**)
libraries	fica dentro do diretório experts Diretório base para a diretiva <code>property library</code> . Todo código que você definir como uma biblioteca será colocado neste diretório. (*)
logs	fica dentro do diretório experts aqui é gravado pelo MetaTrade, para cada dia, todas as mensagens geradas pelos consultores, indicadores e scripts, os quais você pode acessar para estudar algum erro que ocorreu no seu código, vale a pena salientar que a saída da função <code>print</code> vai para este log também. O nome do arquivo é no seguinte formato AAAAMMDD.LOG onde AAAA - Ano que o arquivo corresponde MM - Mes que o arquivo corresponde DD - dia que o arquivo corresponde .LOG - a extensão do arquivo
scripts	fica dentro do diretório experts. Diretório onde serão gravados seus scripts. (**)
templates	fica dentro do diretório experts Quando você cria um consultor, indicador, script ou outro código, você pode especificar ele como uma base para novos códigos, neste diretório você deve colocar as instruções.
logs	Aqui você encontra os logs do MetaTrader, no que diz respeito a erros e aberturas e fechamentos de ordens. O nome do arquivo é no seguinte

	formato AAAAMMDD.LOG
	(*) você pode especificar outro diretório sendo que a base tem de ser este diretório
	(**) você pode colocar o código e o executável em outro diretório que não seja este, porem o mesmo não ficara disponível para ser utilizado pelo MetaTrade

Bom Agora separaremos os tipos de códigos que você mais comumente ira criar :

- Scripts
- Indicadores Peronalizados
- Consultores Especialistas

### Indicadores Personalizados

Nada mais é que um indicador técnico escrito independentemente, ou seja, ele não vem junto com o MetaTrade integrado ao terminal, você necessita compilá-lo para poder utilizar. Como indicadores internos, não podem negociar automaticamente e são apenas utilizado com funções de analise. Embora não possam realizar as negociações, os indicadores podem enviar sinais sonoros e alertas sobre o melhor período para uma negociação, cabendo ao usuário executar a ordem de negociação.

Como citado acima os Consultores são gravador no diretório  
`terminal_directory\experts\indicators.`

### Consultores Especialistas

Um Consultor Especialista (Tambem chamado de Expert, Expert Advisor ou simplesmente EA) nada mais é que um Trade System Automático (*Automated Trading System* - ATS) escrito na linguagem específica do MetaTrader, o *MetaQuotes Language* (MQL), além de notificar o investidor sobre pontos de compra e venda, de acordo com o método especificado, é capaz de executar automaticamente os trades, enviando as ordens diretamente à corretora. Automatizar os trades, você elimina a sua obrigação de acompanhar o mercado o tempo todo. tambem pode testar seu sistema sobre dados históricos, facilitando o desenvolvimento de estratégias de trading. Escrever um Expert Advisor é relativamente fácil, sendo apenas necessário que o investidor tenha conceitos básicos de programação (Neste portal eu lhe ensino a programar para MetaTrader).

A linguagem MQL inclui um grande número de variáveis usadas para controlar dados correntes e passados, operações aritméticas e lógicas, e comandos internos usados para controle das posições (abertura, cancelamento, fechamento de ordens etc). MQL possui uma linguagem similar ao *EasyLanguage*, desenvolvida pela *TradeStation Technologies*.

Dentre as vantagens dos Trade Systems Automáticos, podemos citar as principais:

- O investidor não precisa acompanhar o mercado o tempo todo, podendo realizar diversas operações *intraday* sem perder as boas oportunidades do mercado. Como o FOREX opera 24 horas, o investidor pode deixar o seu Trade System executando e ir dormir, almoçar ou sair com os amigos;
- Quem executa as ordens é o computador, não tendo o risco de qualquer emoção do investidor interferir nos trades. O computador realiza lucro e prejuízo (aciona STOP) sem dor e arrependimento, fazendo exatamente o que o Trade System determina. Vale lembrar que qualquer emoção que venha a influenciar os trades pode causar perdas significativas.

Um Consultor Especialista começa a funcionar a cada mudança na cota do símbolo ao qual esta ligado. Cada nova cota gera uma execução da função principal do Consultor (`start()`), logicamente, se ocorrer uma nova cota no par e o Consultor ainda não terminou de executar a função principal, o chamado para esta nova cota será perdido. O Consultor serve para informar sobre uma possibilidade para negociar e executar a referida negociação em um cliente, o que o faz emitir automaticamente ordens diretamente ao servidor sem a intervenção do usuário. Como a maioria de sistemas negociando. É importante salientar que você tem a possibilidade de testar seu histórico no Testador de Estratégias, utilizando dados históricos para avaliar seu consultor, porém, eu, particularmente não confio muito no testador de estratégia, já vi muitas divergências ocorrerem entre os testes e a conta demo ou real.

Como citado acima os Consultores são gravados no diretório

`terminal_directory\experts.`

---

## Scripts

É na verdade uma sequência de comandos pré determinados, o qual é criado para ser executado uma única vez. Uma vez terminada a execução os mesmos são retirados da memória, caso haja a necessidade de seu trabalho o mesmo deverá ser executado novamente pelo usuário. Você pode considerar um script como sendo um Consultor especialista que é executado somente uma vez. Alguns trabalhos comuns para scripts

- Colocar ordens de negociação com parâmetros padronizados (Volume, SP, TP, etc)
- Fechar todas as ordens abertas
- Modificar ordens abertas
- etc

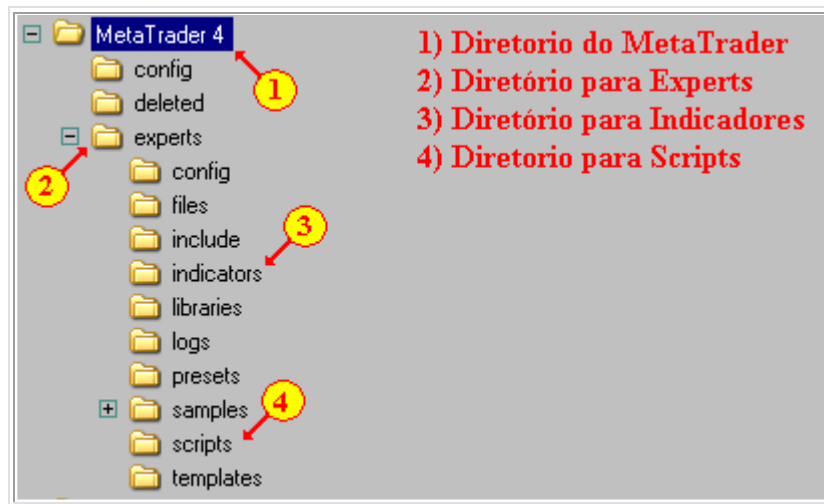
Como citado acima os Consultores são gravados no diretório

`terminal_directory\experts\scripts.`

---

## Como ativar seu código no MetaTrader

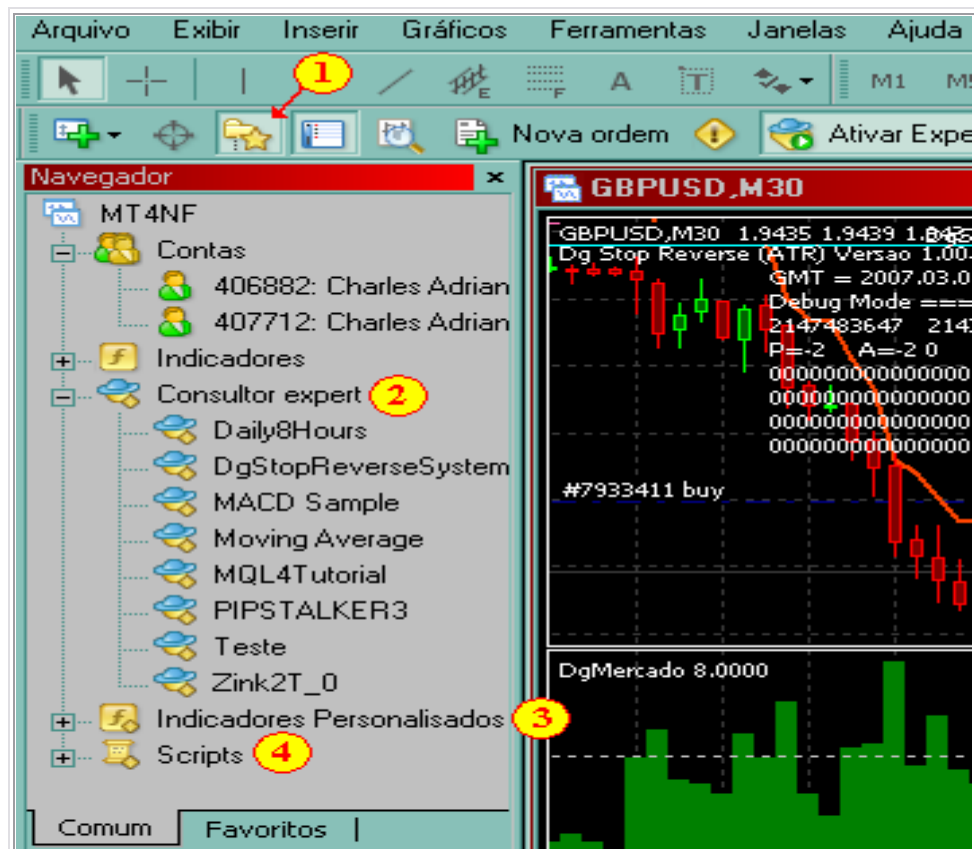
A) Certifique-se que o executável do script, indicador ou Consultor esteja no diretório apropriado, caso não esteja no diretório coloque-o no diretório correspondente, e caso não esteja compilado, compile-o



B) No MetaTrade, abra o gráfico em que você deseja que seu código seja executado

C) Abra a janela de navegação, e localize o tipo de executável que você quer utilizar





- 1) botão na barra de ferramenta para abrir a janela do Navegador
- 2) Localização dos Consultores
- 3) Localização dos indicadores
- 4) Localização dos Scripts

D) Simplesmente clique no nome do código que você deseja executar, sem soltar o botão do mouse arraste e solte em cima do gráfico ao qual este código deva ser ligado, caso seja um indicador ou consultor o mesmo apresentará a janela de propriedades onde você informa a configuração na qual o código deve ser executado (lembre-se aqui você configura todas as variáveis externas do seu código). No caso dos scripts, a janela de propriedades não será mostrada, pois o mesmo não possui esta facilidade.

E) Agora uma explicação resumida para conferir se seu consultor vai ser executado corretamente



- 1 - Expert que vou usar (note que o ícone esta mais colorido, indica que já esta compilado, os outros dois não)
- 2 - depois de clicar com o botão direito do mouse sobre o nome do experts aparece o menu e peço a opção para adicioná-lo ao chart
- 3 - Ai esta o nome do chart, se aparecer um "x" indica que o expert não esta sendo executado
- 4 - para ativar a excussão procure na barra de ferramentas este ícone e clique nele
- 5 - Carinha indica que o expert esta sendo executado, porem a carinha triste indica que ele não pode abrir posições
- 6 - para liberar o expert para abrir posições peça propriedades e selecione a marca "alow live trading" (note que a minha não esta selecionada, você tem de selecionar)
- 7 - apareceu a carinha feliz o expert ta funcionando...

# Criando indicadores com a linguagem MQL4

## PARTE I

---

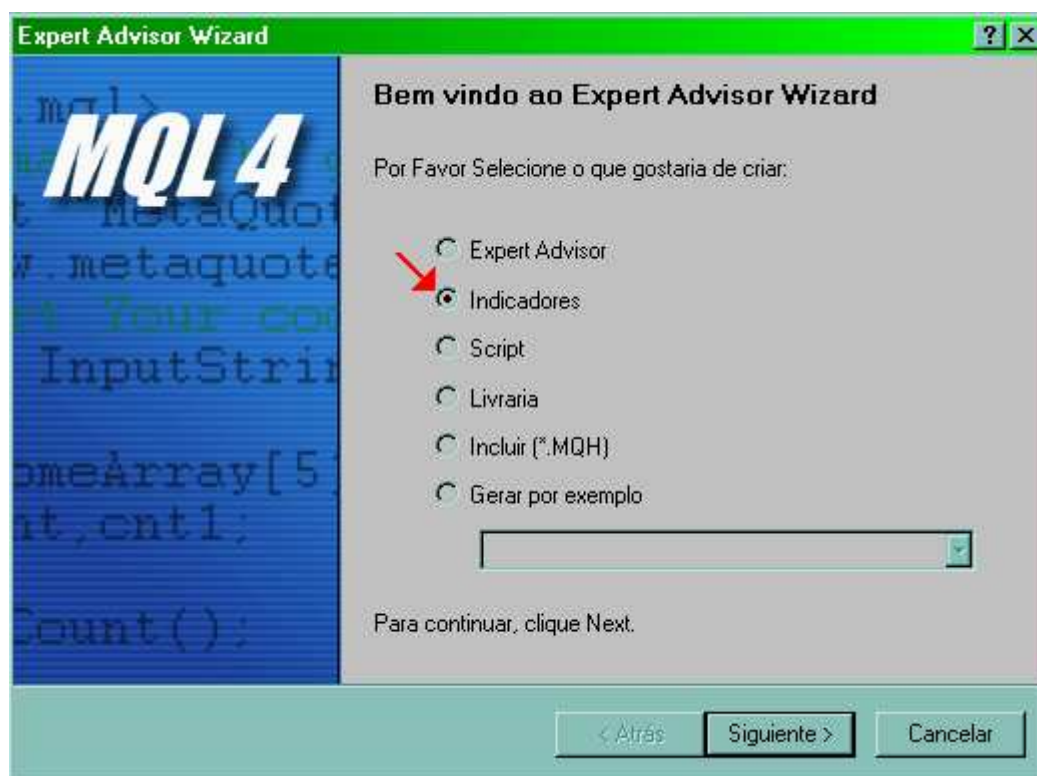
Bem vindo ao mundo prático de que você ira começar neste momento em MQL4; bem vindo a seu primeiro indicador em MQL4. Eu recomendo-o reler com muito cuidado as lições precedentes, antes de continuar com estas séries do cursos, o motivo é que nós os usaremos tudo que vimos ate agora nossas explanações e estudos dos indicadores e consultores especialistas feitos sob encomenda, os quais nós criaremos nesta série das lições. Hoje nós criaremos um indicador simples, que não signifique demasiado para nosso mundo, mas significa muito para nossa compreensão da programação em MQL4. Simplesmente realizaremos a subtração dos preços `high` e `low`; porem não se apresse, você saberá como tudo funciona mais adiante.

Então mãos a obra

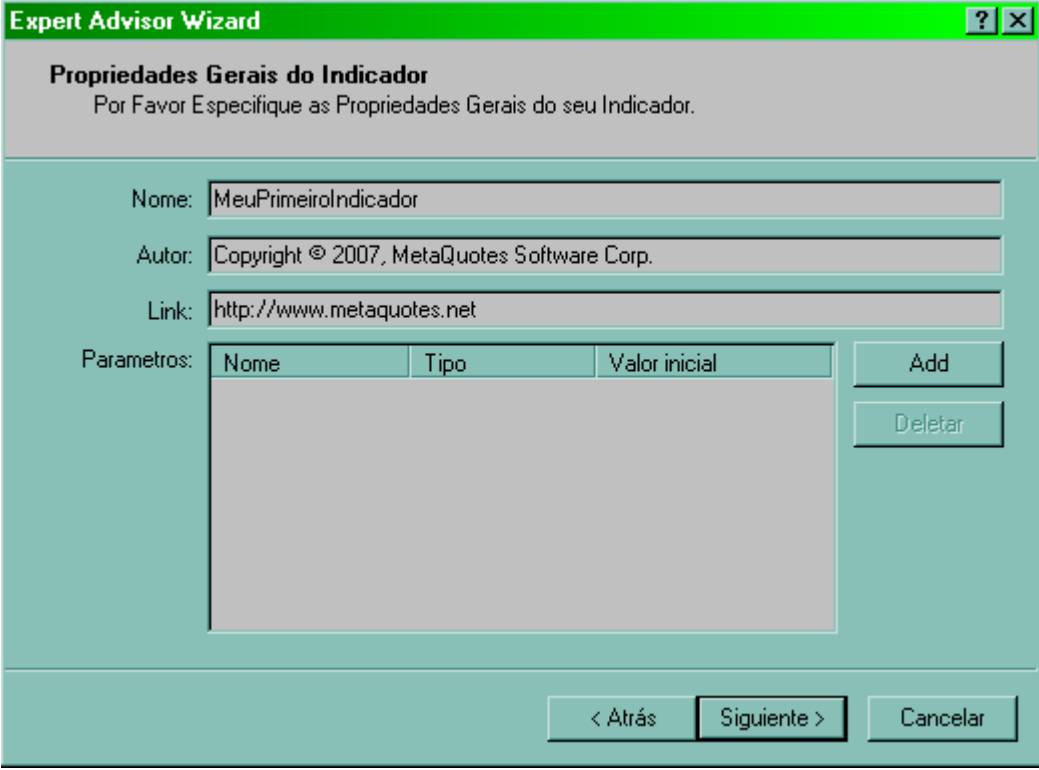
---

### O assistente do MetaEditor

Simplesmente, abra o MetaEditor e inicie um novo arquivo fonte. Aparecerá a janela do assistente do MetaEditor. Escolheremos que queremos montar o código de um indicador.



O Próximo passo é clicar no botão seguinte, e aparece a tela onde identificaremos o **nome** do nosso indicador, o **autor** ou nome do proprietário do código, bem como o **link** da página ou do e-mail do mesmo. Aqui também definiremos quais são as variáveis que o usuário pode modificar para configura o indicador (no nosso caso não temos nenhuma neste primeira versão)



The image shows a Windows-style dialog box titled "Expert Advisor Wizard". The main heading is "Propriedades Gerais do Indicador" (General Properties of the Indicator). Below the heading is a subtitle: "Por Favor Especifique as Propriedades Gerais do seu Indicador." (Please specify the general properties of your indicator). The dialog contains three text input fields: "Nome:" (Name) with the value "MeuPrimeiroIndicador", "Autor:" (Author) with the value "Copyright © 2007, MetaQuotes Software Corp.", and "Link:" (Link) with the value "http://www.metaquotes.net". Below these fields is a section labeled "Parametros:" (Parameters) which contains a table with three columns: "Nome" (Name), "Tipo" (Type), and "Valor inicial" (Initial value). The table is currently empty. To the right of the table are two buttons: "Add" and "Deletar" (Delete). At the bottom of the dialog are three buttons: "< Atrás" (Previous), "Siguiente >" (Next), and "Cancelar" (Cancel). The "Siguiente >" button is highlighted with a black border.

Nome	Tipo	Valor inicial
------	------	---------------

Clicamos no botão seguinte e finalmente aparece a ultima pagina do assistente, onde identificaremos as características de plotagem do nosso indicador, entre elas :

- Onde o indicador será desenhado
- A janela de desenho possui um limite mínimo ou Maximo
- Quantos indicadores e suas características teremos (máximo de 8 itens)

**Expert Advisor Wizard** [?] [X]


**Propriedades do Indicador Personalizado**  
Por Favor Especifique as Propriedades do Indicador Personalizados

☒ Indicator in separate window

☐ Minimo

☐ Maximo

Indexes:

#	Tipo	Cores	Simbolo
1	Line	 Red	

Add

Deletar

< Atrás Finalizar Cancelar

Feito isto, clicamos em finalizar e o assistente ira gerar o código inicial de nosso indicador, poupando-nos algum trabalho de digitação.

## O código gerado pelo assistente

```
//+-----+
-----+
//|
MeuPrimeiroIndicador.mq4 |
//|           Copyright © 2007, DooMGuarD Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2007, MetaQuotes Software
Corp."
#property link "http://dgforex.50webs.com"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 C'None'
//---- buffers
double ExtMapBuffer1[];

//+-----+
-----+
//| Custom indicator initialization function |
//+-----+
-----+
```

```

int init()
{
    //---- indicators
    SetIndexStyle(0,DRAW_LINE);
    SetIndexBuffer(0,ExtMapBuffer1);
    //----
    return(0);
}

//+-----+
//| Custom indicator deinitialization function |
//+-----+
int deinit()
{
    //----

    //----
    return(0);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int start()
{
    int counted_bars=IndicatorCounted();
    //----

    //----
    return(0);
}
//+-----+
//+-----+

```

Não sei quanto a você, mas quem inventou o assistente, merece um premio, pois em poucos passos geramos código suficiente para compilar um programa completo de indicador em MQL4 sem que o compilador retorne nenhum erros. Claro que o indicador compilado com este código não ira desenhar nada no gráfico ao qual for anexado (somente criará a janela onde ele deverá trabalhar), porem, você tem de concordar, nos poupou um grande trabalho na digitação. Agora vamos explicar as partes de do código, antes e começarmos a programar o nosso indicador.

---

## O cabeçalho do código gerado pelo assistente

```
//+-----+
-----+
// |
MeuPrimeiroIndicador.mq4 |
// |           Copyright © 2007, DooMGuaRD Sistemas
Especialistas... |
// |
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2007, DooMGuaRD Sistemas
Especialistas..."
#property link "http://dgforex.50webs.com"
```

Aqui não temos muito segredos, o que o assistente montou no código, nada mais é que as definições e comentários sobre quem escreveu o código. Note que o que o assessor nada mas fez que copiar as informações que você colocou na segunda janela do assistente (Autor e Link) como uma propriedade para o compilador.

Aqui ressaltamos uma particularidade da linguagem MQL4 : os comentários. Comentários são segmentos ou partes do seu código que não interessam ao compilador, portanto você pode escrever o que quiser neles que o compilador não irá se importar. Em MQL4 existem dois tipos de comentários

- Comentários de Linha : sempre que você encontrar duas barras juntas "//" significa que deste ponto em diante até o fim da linha será um comentário

```
//+-----+
-----+
// |
MeuPrimeiroIndicador.mq4 |
// |           Copyright © 2007, DooMGuaRD Sistemas
Especialistas... |
// |
http://dgforex.50webs.com |
//+-----+
-----+
```

- Comentários de várias linhas : sempre que você encontrar uma barra e um asterisco "/\*", significa que a partir deste ponto está iniciando um comentário, este comentário somente terá um fim ao encontrar a sequência de um asterisco e uma barra "\*/". Em outras palavras, tudo que estiver entre /\* e \*/ será considerado comentário

```

/*+-----+
-----+
|
|MeuPrimeiroIndicador.mq4 |
|           Copyright © 2007, DooMGuard Sistemas
|Especialistas... |
|
|http://dgforex.50webs.com |
|
+-----+
-----+*/

```

## As definições no código gerado pelo assistente

```

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 C'None'
//---- buffers
double ExtMapBuffer1[];

```

Bem nesta parte do código, foi criado, pelo assistente, a identificação que estamos criando um indicador. Isto se dá pelas definições das propriedades `#property`. Qual a função de cada uma? É o que veremos em seguida

`#property indicator_separate_window`: diz ao compilador para criar um indicador, onde as retas plotadas no gráfico, não farão parte da área do gráfico utilizada para colocar as cotações. O sistema vai criar uma janela separada para plotar seus valores. Se os valores do seu indicador, acompanhar os valores das cotações, você pode plotar ele na mesma área das cotações, para isso esta propriedade tem de ser `#property indicator_chart_window`. Na figura abaixo identificamos as 2 áreas no gráfico





onde :

- 1 - Chart Window
- 2 - Separate Window
- 3 - Separate Window

Nota: você pode colocar vários indicadores com seus valores plotados na janela principal do gráfico, mas não pode juntar 2 indicadores uma janela separada, ou seja cada vez que você usar , o sistema criará uma nova área de plotagem separada no gráfico.

`#property indicator_buffers`: aqui dizemos ao compilador quantos buffer (área de armazenamento de memória) que utilizaremos no nosso indicador, você pode considerar cada buffer como um conjunto de valores que irá gerar uma curva em um gráfico, para entender mais facilmente, porem o objetivo dos buffers vai além disso (mais adiante veremos o seu funcionamento). No nosso caso usaremos somente um buffer de memória. Você pode ter no máximo 8 buffers para cada indicador.

`#property indicator_color1`: para cada buffer do indicador podemos atribuir características que são assumidas como padrões, ou características iniciais. Uma destas características é a cor que usaremos para plotar o indicador. Aqui está definido que não utilizaremos nenhuma cor como padrão, o usuário terá que escolher uma na janela de propriedades do indicador.

Por fim, mas não menos importante, definimos uma variável, na qual colocaremos os valores calculados do nosso indicador, para que os mesmos sejam plotados no gráfico. Como a quantidade de itens de um indicador vai aumentando a medida que novas cotações, em relação ao tempo, vão aparecendo, esta matriz é criada sem nenhuma dimensão, o compilador acrescenta um código que a medida que necessitamos mais um item na matriz ao usar este item ele será automaticamente criado na memória se não existir.

## O função de inicialização do indicador

```
//+-----+
-----+
//| Custom indicator initialization function |
//+-----+
-----+
int init()
{
    //---- indicators
    SetIndexStyle(0,DRAW_LINE);
    SetIndexBuffer(0,ExtMapBuffer1);
    //----
    return(0);
}
```

Esta função é a primeira parte do nosso código executada quando você acrescentar seu indicador no gráfico. Ela será executada a cada vez que você mudar as propriedades do seu indicador. Aqui declaramos todas as características do indicador que desejamos. No código criado pelo assistente indicamos que o estilo do indicador será uma linha e que a variável `ExtMapBuffer1` será utilizada como buffer do operador

Não sei se você observou, mas quando declaramos as propriedade do indicador o índice que ele recebeu foi 1 (um)

```
#property indicator_color1 C'None'
```

Porém no código desta função o índice do indicaro é 0 (Zero)

```
SetIndexStyle(0,DRAW_LINE);
SetIndexBuffer(0,ExtMapBuffer1);
```

Isto se deve ao fato de que todas as variáveis de memória que são matrizes começam com o índice 0 (zero). Como a propriedade do indicador não é uma matriz mas sim definições separadas onde a diferença se da no numero ao final da propriedade os criadores da linguagem acharam por bem definir as propriedade do indicador começando em 1

---

## O função de termino do indicador

```
//+-----+
-----+
//| Custom indicator deinitialization function |
//+-----+
-----+
int deinit()
{
    //----

    //----
    return(0);
}
```

Nesta função você simplesmente elimina qualquer lixo que seu indicador possa ter deixado no gráfico, tipo, se voce desenhou algum símbolo que não foi criado como buffer, ou limpa um comentário, ou simplesmente grava alguma coisa em algum arquivo. Sua necessidade girara você nesta tarefa.

---

## O função de trabalho do indicador

```
//+-----+
-----+
//| Custom indicator iteration function |
//+-----+
-----+
int start()
{
    int counted_bars=IndicatorCounted();
    //----

    //----
    return(0);
}
//+-----+
-----+
```

Bem, nesta função é onde colocaremos o código real de calculo do indicador para cada intervalo de tempo no gráfico. Ela será chamada sempre que mudar alguma cotação do par que esta sendo representado no gráfico bem como em cada vez que uma barra necessitar do seu cálculo.

O simplesmente criou uma variável (counted\_bars) com o número de barras que não necessitam o calculo do valor do indicador. Em outra palavras ela contem o numero de barras onde o indicador já foi calculado e, em teoria, não necessitam ser recalculadas novamente

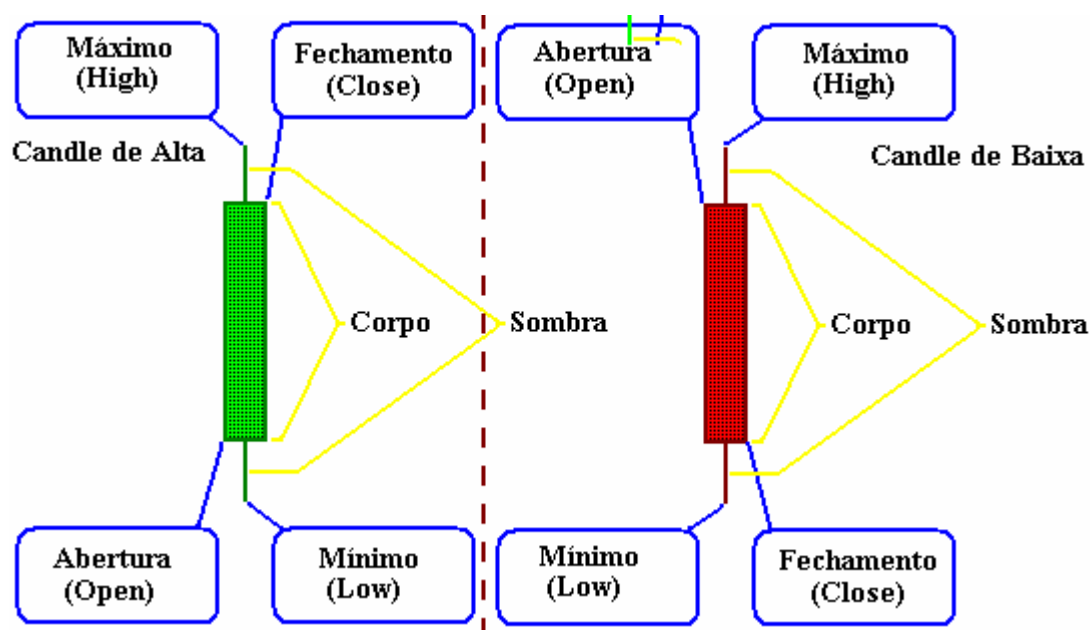
# Criando indicadores com a linguagem MQL4

## PARTE II

---

### Como funciona o nosso primeiro indicador

Bom nada mais justo que começar nosso primeiro indicador com a explicação do seu funcionamento. Considere o desenho de um candle (Vela)



Nosso indicador, nada mais faz que determinar a extensão de um candle. Um indicador deste tipo, pode revelar a você como os candles se comportam quanto ao tamanho, fato este que pode orientar você em seus stops.

O Seu funcionamento é simples :

Indicacador = (Preço\_High - Preço\_Low) / Variação\_Minima\_do\_Par

Por variação mínima do par se ente como sendo o valor da menor variação de preço que um par pode ter, por exemplo

EURUSR = 0.0001  
USDJPY = 0.01

---

## Definindo padrões no nosso indicador

Agora definiremos como nosso indicador irá trabalhar, bem, eu achei melhor desenhar nosso indicador como barras em vez de linhas, pois assim visualizamos de uma maneira melhor nosso gráfico. Vamos ao código (Veja o código adicional esta em **negrito** e este será sempre nossa metodologia ao adicionar algum código novo ou modificado)

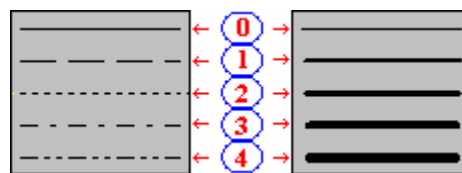
```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Green           //mudamos a cor para
verde
#property indicator_style  STYLE_SOLID //queremos uma linha
sólida
#property indicator_width1 3             //espessura da linha = 3

//---- buffers
double ExtMapBuffer1[];
```

### Definições de estilo de plotagem no gráfico

```
// estilo das linhas
DRAW_LINE      0 //Desenha linhas.
DRAW_SECTION   1 //Desenha segmentos de linhas.
DRAW_HISTOGRAM 2 //Desenha histogramas.
DRAW_ARROW     3 //Desenha arrows (símbolos).
DRAW_ZIGZAG    4 //Desenha segmentos de linhas entre os buffers
pares e impares dos indicadores.
DRAW_NONE      12 //Não desenha nada (usado como armazenamento
intermediário).
```

### Espessura (**width**) e formato de linhas



```
// Tipos de linhas para a espessura 0
STYLE_SOLID     0 //traço contínuo.
STYLE_DASH      1 //tracejado.
STYLE_DOT       2 //pontilhado.
STYLE_DASHDOT   3 //alternado pontos e traços.
STYLE_DASHDOTDOT 4 //alternado traços e dois pontos.
```

---

## Função de inicialização

Aqui definiremos os demais parâmetros do indicador, como dizer o tipo de linha, qual buffer utilizar, dar um nome que aparecerá no gráfico, etc, etc. Vamos ao código

```
//+-----+
-----+
//| Custom indicator initialization function |
//+-----+
-----+
int init()
{
    //---- indicators
    SetIndexStyle(0,DRAW_HISTOGRAM);    //queremos que seja um
histograma
    SetIndexBuffer(0,ExtMapBuffer1);

    IndicatorShortName("Meu Primeiro: ");    //damos um nome ao
indicador

    //----
    return(0);
}
```

---

## Função de termino

Não possuímos nenhuma espécie de trabalho para quando o indicador é retirado do gráfico, quando você fecha a janela do gráfico ou quando o MetaTrader é desligado. Logo essa parte do código continua a mesma.

---

## Função de trabalho

Primeiramente retiraremos a declaração de `counted_bars` pois, não usarei este nome de variável. Criaremos duas novas variáveis, uma para controlar o laço (loop) e outra para limitá-lo. No laço simplesmente calcularemos o valor de cada barra que foi requisitada.

```

//+-----+
-----+
//| Custom indicator iteration function |
//+-----+
-----+
int start()
{
    int Itens = Bars - IndicatorCounted(), // total de velas a
    calcular
        Item; // controle do laço
    //---- Calculo dos valores requisitados
    for(Item=Itens;Item>=0;Item--) {
        ExtMapBuffer1[Item] = (High[Item] - Low[Item])/Point;
    }

    //----
    return(0);
}
//+-----+
-----+

```

**Itens** : simplesmente contem a quantidade de candles que necessitam de recálculo. Aqui vale a pena lembrar algumas variáveis no MQL4 :

- **Bars**: é o número total de candles ou barras no gráfico
- **IndicatorCounted()**: é o numero total de candles ou barras que não se modificaram desde a ultima chamada da função trabalho do indicador, logo a conta nos da a quantidade de itens a serem recalculados. Vale a pena salientar que o índice 0 (zero) é o candle atual, ou o que ainda não se completou no tempo, ou ainda o candle mais a direita do gráfico.
- **High[Item]**: Valor do preço máximo da cotação do para para o referido candle
- **Low[Item]**: Valor do preço mínimo da cotação do para para o referido candle
- **Point**: é o valor da variação mínima da cotação do par, é muito usado para transformar diferenças entre cotações para pips

Então **(High[Item] - Low[Item])/Point** nos da a diferença em pips entre o valor máximo e o valor mínimo do candle o qual será armazenado no buffer do indicador () e que será utilizado pelo MetaTrade para gerar nosso gráfico de histograma.

## Como ficou nosso trabalho

Compile nosso indicador (se você copiou tudo certo não teremos erros de compilação) e acrescente no gráfico (eu acrescentei no EURUSD em TimeFrame de uma hora), teremos algo parecido com isso:



Note que o nome que você deu ao indicador aparecerá no gráfico, junto com o valor atual do mesmo.

---

### [Download do código Fonte](#)

#### [MeuPrimeiroIndicadorVersao1.MQ4](#)

Clicar com o botão direito do mouse e no menu escolher Salvar Como.



# Criando indicadores com a linguagem MQL4

## PARTE III

---

### Como podemos melhorar nosso indicador

Bem agora incrementaremos nosso indicador, acrescentaremos linhas de níveis e também uma media de valores dos candles. Nosso objetivo final será ficar com um indicador parecido como desenho abaixo. Para não perder o código de nosso primeiro indicador, vamos salva-lo com outro nome. Com o código do indicador com janela ativa no MetaEditos vá no menu em Arquivos -> Salvar Como e salve-o com o nome de "**MeuPrimeiroIndicadorPoderoso.mq4**". Consequentemente mude o cabeçalho do arquivo

```
//+-----+
-----+
//|
MeuPrimeiroIndicadorPoderoso.mq4 |
//|           Copyright © 2007, DooMGuarD Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2007, MetaQuotes Software
Corp."
#property link "http://www.metaquotes.net"
```

---

### Novas definições nos padrões no nosso indicador

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Green           //mudamos a cor para
verde
#property indicator_style1 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width1 3              //espessura da linha = 3

// definiremos linhas de níveis no gráfico
// Nota pode-se definir no maximo 8 valores
#property indicator_levelcolor Yellow
#property indicator_levelstyle STYLE_DOT
#property indicator_levelwidth 0
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
#property indicator_level4 100
```

Se você compilar esta simples mudança e colocar no gráfico terá esta seguinte saída:



Note que só apareceram 2 linhas pontilhadas, apesar de termos definido 4 linhas, é que as outras ficam fora do limite superior, se você quiser pode definir o valor mínimo e máximo para um indicador utilizando o seguinte código (o código a seguir não faz parte de nosso indicador, é somente ilustrativo).

```
#property indicator_minimum 0
#property indicator_maximum 150
```

Se você acrescentar estas definições no seu código terá uma saída como a de abaixo (note os limites verticais do grafico):



Agora sim apareceram as 4 linhas de níveis definidas, mas este código não fará parte do nosso indicador.

## Novas variáveis e definições no indicador

Como vamos trabalhar com uma média de valores, necessitamos que o usuário nos informe que tipo de configuração quer utilizar nestas médias. Adicionalmente devemos informar ao compilador que queremos um novo buffer para mais uma pena de desenho no indicador. Mãos a obra então

```
#property indicator_separate_window
#property indicator_buffers 2          //são 2 buffers agora
                                        //primeiro
buffer
#property indicator_color1 Green       //mudamos a cor para
verde
#property indicator_style1 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width1 3           //espessura da linha = 3
                                        //segundo
buffer
#property indicator_color2 Red         //mudamos a cor para
vermelho
#property indicator_style2 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width2 0           //espessura da linha = 0

// definiremos linhas de níveis no gráfico
// Nota pode-se definir no maximo 8 valores
```

```

#property indicator_levelcolor Yellow
#property indicator_levelstyle STYLE_DOT
#property indicator_levelwidth 0
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
#property indicator_level4 100

//---- input parameters
extern int    PeriodoMA = 25; // Número de períodos da média
extern int    TipoMA    = 1; // tipo de média a ser utilizada
extern string TipoMASos = "SMA=0, EMA=1, SMMA=2, LWMA=3"; //
tipos de média

//---- buffers
double ExtMapBuffer1[];
double ExtMapBuffer2[]; // buffer para a nova pena do
indicador
double ValoresParaMedia[500]; // Maior periodo para Media é de
500 valores

```

Os parâmetros de entrada "input parameters" geralmente são definidos no assistente o qual ficaria como na figura abaixo

**Expert Advisor Wizard**

**Propriedades Gerais do Indicador**  
Por Favor Especifique as Propriedades Gerais do seu Indicador.

Nome: MeuPrimeiroIndicadorPoderoso

Autor: Copyright © 2007, MetaQuotes Software Corp.

Link: <http://www.metaquotes.net>

Parametros:

Nome	Tipo	Valor inicial
PeriodoMA	int	14
TipoMA	int	1

Add

Deletar

< Atrás   Siguiente >   Cancelar

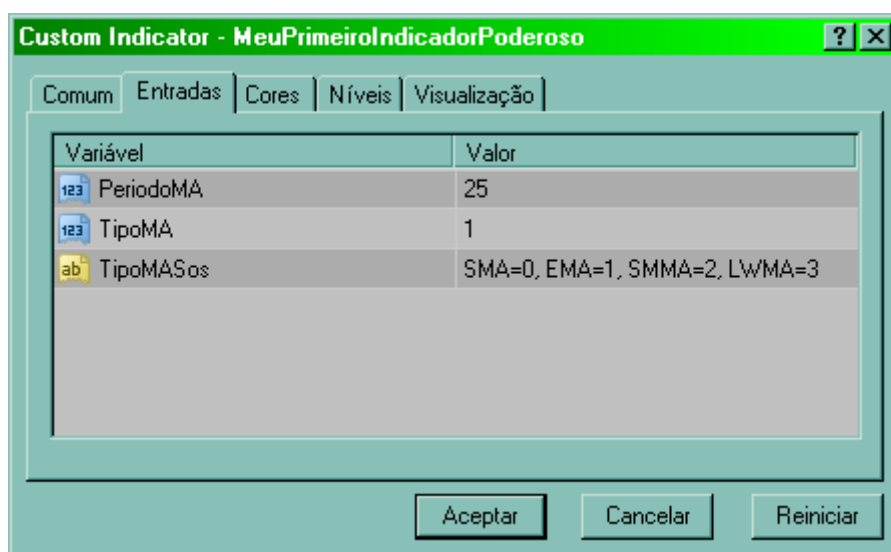
Para acrescentar uma parâmetro, simplesmente clique no botão "add" que aparece uma nova linha, com um duplo clique em cada uma das configurações você pode mudar o que ali aparecer escrito. Lógico que para eliminar um parâmetro você simplesmente seleciona o mesmo com um clique do mouse sobre o parâmetro e depois clica no botão "Deletar"

Só duas observações,

1) para utilizarmos o sistema de calculo de médias móveis, usaremos as mesmas definições do MQL4 para designar o tipo, a saber

```
// Tipos de linhas para a espessura 0
MODE_SMA 0 //Média simples
MODE_EMA 1 //Média exponencial
MODE_SMMA 2 //Amortizada.
MODE_LWMA 3 //Linear com atribuição de pesos.
```

2) **Tipomasos** na verdade não tem propósito real em nossos cálculos, só foi adicionado para mostrar ao usuário o que ele pode informar para a variável **Tipoma**. Ira produzir a seguinte tela de parâmetros (note que a terceira linha é uma ajuda importante ao usuário)



Com isso o usuário que utilizar o indicador vai saber que o **Tipoma=1** é para utilizar uma média tipo EMA.

---

## Modificações na função de inicialização

Agora diremos as características de nossa nova pena para o sistema de desenho do indicador

```
//+-----+
-----+
//| Custom indicator initialization function |
//+-----+
-----+
int init()
{
    //---- indicators
    SetIndexStyle(0,DRAW_HISTOGRAM);          //queremos que seja um
histograma
    SetIndexBuffer(0,ExtMapBuffer1);

    SetIndexStyle(1,DRAW_LINE);              //queremos que seja uma
linha
    SetIndexBuffer(1,ExtMapBuffer2);          //buffer a ser utilizado
    SetIndexDrawBegin(1,PeriodoMA);          //buffer a ser utilizado

    IndicatorShortName("Meu Primeiro Poderoso: "); //damos um nome
ao indicador

    // Testa se periodo valido
    if (PeriodoMA<1)    PeriodoMA = 1;
    if (PeriodoMA>500) PeriodoMA = 500;

    // Testa se periodo valido
    if ((TipoMA<0)|| (TipoMA<3)) TipoMA = 1; // Padrão é EMA

    //----
    return(0);
}
```

Somente uma observação, como temos um certo numero de períodos para calcular a média, se estivermos calculando sobre menos barras que o período necessário, então não teremos valores corretos da media e consequentemente do indicador, então diremos ao sistema de plotagem de indicadores que não queremos plotar essas barras iniciais do gráfico. Para você entender a função **SetIndexDrawBegin** veja a figura abaixo, pois uma imagem vale mais que mil palavras....



---

### Função de termino

Não possuímos nenhuma espécie de trabalho para quando o indicador é retirado do gráfico, quando você fecha a janela do gráfico ou quando o MetaTrader é desligado. Logo essa parte do código continua a mesma.

---

### Função de trabalho

Agora vamos acrescentar o calculo da média para que a mesma possa ser plotada no sistema

```

//+-----+
-----+
//| Custom indicator iteration function |
//+-----+
-----+
int start()
{
    int Itens = Bars - IndicatorCounted(), // total de velas a
    calcular
        Item; // controle do laço
    //---- Calculo dos valores requisitados
    for(Item=Itens;Item>=0;Item--) {
        // calcula o tamanho do candle em pips
        ExtMapBuffer1[Item] = (High[Item] - Low[Item])/Point;
        // Monta a matriz para calcular a média
        for(int i=0;i<PeriodoMA;i++)
            ValoresParaMedia[i] = (High[Item + i] - Low[Item +
i])/Point;
        // calcula a média e joga no buffer do indicador
        ExtMapBuffer2[Item] =
iMAOnArray(ValoresParaMedia,PeriodoMA,PeriodoMA,0,TipOMA,0);
    }

    //----
    return(0);
}
//+-----+
-----+

```

Bom, para calcular este segundo buffer do indicador, necessitamos criar uma matriz com os dados para o calculo da média. Você deve estar se perguntando, porque não usar uma matriz aberta, sem dimensões, a passagem de parâmetro para uma função em MQL4 deve ser uma região definida de memória, e uma matriz aberta não satisfaz esta condição. Outra pergunta que você deve estar fazendo, porque não pego simplesmente o valor de `ExtMapBuffer1` assim `ValoresParaMedia[i] = ExtMapBuffer1[Item]`, simples estamos caculando o valro de `ExtMapBuffer1` porem, os valores apos o valor atual que a matriz necessita ainda não foram calculados, é por este motivo que utilizamos a função `SetIndexDrawBegin` na função de inicialização, exatamente para evitar os valores do começo do gráfico.

---

## Como ficou nosso trabalho





Nota : Junto com o nome do indicador sempre aparece o valor atual dos buffers do indicador, ou seja, referente a última cota recebida do servidor.

### [Download do código Fonte](#)

#### [MeuPrimeiroIndicadorVersao2.MQ4](#)

Clicar com o botão direito do mouse e no menu escolher Salvar Como.

# Criando indicadores com a linguagem MQL4

## PARTE IV

### Dicas de Programação

---

Para facilitar sua vida na programação de indicadores em MQL4 tenha sempre em mente os seguintes itens

---

#### Onde desenhar meu indicador

- Em que parte do gráfico meu indicador vai aparecer, lembre-se que se os valores do seu indicador acompanharem as cotações, elas podem aparecer no gráfico principal, porém se os valores do indicador forem distantes das cotações o que pode acontecer é que a plotagem dos valores fique fora dos limites dos gráficos

```
#property indicator_chart_window // na janela principal
// ou
#property indicator_separate_window // em uma janela abaixo
```

---

#### Necessito limitar os valores do indicador

Se você deseja ou sabe que o seu indicador oscilará entre determinados valores então deixe estes valores previamente programados evitando que o usuário tenha que criar seus limites a cada vez que anexar o indicador a um gráfico

```
#property indicator_minimum 0
#property indicator_maximum 150
```

---

#### Quanto buffers utilizar

Você, em via de regra, quando programa um indicador quer que ele mostre alguma coisa no gráfico, e provavelmente em muitas vezes necessitará utilizar mais de um tipo de indicação, por exemplo, você pode querer colocar uma linha e sinais quando o indicador detectar um sinal de compra ou de venda. Então você deve definir quantas penas de desenho seu indicador vai ter.

```
#property indicator_buffers n // onde n pode variar de 1 a 8
```

Aqui temos uma limitação de no máximo 8 penas, e você não pode usar mais. O que você usou na definição acima, diz ao compilador quantas penas você quer utilizar, porém você pode utilizar buffers sem a necessidade de desenhar valores no gráfico. Qual a vantagem de utilizar um buffer e não plotar nada, simples, quando

you trace a buffer you work with an open matrix, so, when starting a new candle, the buffer of index 0 was placed at index 1 so that index 0 is available for this new candle, or in other words, for each new candle the values are manipulated in memory so that the index zero of the buffer is always the last candle. So if you want auxiliary values, referring to previous calculations, you place them in a buffer that will not be drawn, so you do not need to create a code to move these values in memory to leave everything as you need, and also do not need to execute the calculation for all candles each time you calculate the indicator using these auxiliary data. For more buffers than the defined ones see the code below

```
property indicator_buffers 1 // onde n pode variar de 1 a 8

...

double ExtMapBuffer1[];
double ExtMapBuffer2[];

...

int init()
{
    //---- indicators
    IndicatorBuffers(2); //aumentar o numero de buffers

    SetIndexStyle(0,DRAW_HISTOGRAM); //este será plotado
    SetIndexBuffer(0,ExtMapBuffer1);

    SetIndexBuffer(1,ExtMapBuffer2); //este não será plotado

    return(0);
}
```

---

## Definindo padrões para os buffers

Posso definir vários padrões, os que aparecem na janela de propriedades dos indicadores quando o usuário anexa o indicador no gráfico, deixe o máximo de valores previamente definidos e que sejam os valores ótimos para cada uma das variáveis, ou pelo menos os mais usados

```
buffer //primeiro
#property indicator_color1 Green //mudamos a cor para
verde
#property indicator_style1 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width1 3 //espessura da linha = 3

// definições dos buffers 2 a 7

//oitavo e ultimo buffer
#property indicator_color8 Red //mudamos a cor para
vermelho
#property indicator_style8 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width8 0 //espessura da linha = 0
```

---

## Linhas de níveis

Se você tem em seu indicador valores que são mais ajustados a determinadas situações, como por exemplo, entre dois valores calculados, indicam um nervosismo no mercado ou algo assim, então melhor mostrar essa região ao usuário.

```
#property indicator_levelcolor Yellow
#property indicator_levelstyle STYLE_DOT
#property indicator_levelwidth 0
#property indicator_level1 25
// definições dos níveis de 2 a 7
#property indicator_level8 100
```

---

## Se eu necessitar mais de 8 buffers

Bem eu disse que você pode colocar no Máximo 8 buffers para as penas de plotagem no gráfico para seu indicador. Porém você pode lançar mão de alguns artifícios para colocar símbolos, como, setas, letras gregas ou outros no seu gráfico, como por exemplo, para indicador áreas de entrada e saída do mercado. Para isso lance mão da função ObjectCreate, abaixo uma simples função para desenhar setas no gráfico

```
void MostraSeta(string Nome, double Tempo, double Valor, bool
```

```

Sobe)
{
    ObjectDelete(Nome);
    ObjectCreate(Nome, OBJ_ARROW, 0, Tempo, Valor);
    if (Sobe)
        ObjectSet(Nome, OBJPROP_ARROWCODE, 241);
    else
        ObjectSet(Nome, OBJPROP_ARROWCODE, 242);
    ObjectSet(Nome, OBJPROP_COLOR, Blue);

    return (0);
}

// usando

int start()
{
    int  Itens = Bars - IndicatorCounted(), // total de velas a
    calcular
        Item; // controle do laço
    bool SinalBuy = False; // Sinal para comprar
    bool SinalSell = False; // Sinal para vender
    string NomeSinal; // nome do objeto

    //---- Calculo dos valores requisitados
    for(Item=Itens;Item>=0;Item--) {
        // calcula a curva a plotar e se tem sinal
        //...

        //Cria um nome para o objeto
        NomeSinal = StringConcatenate("Sinal",Time[Item]);
        // se tem sinal buy coloca a seta 1 pip abaixo do preço
        mínimo
        if (SinalBuy)
            MostraSeta(NomeSinal ,Time[Item],Low[Item]-Point,True);
        // se tem sinal buy coloca a seta 1 pip acima do preço
        Máximo
        if (SinalSell)
            MostraSeta(NomeSinal ,Time[Item],High[Item]-Point,False);
    }

    //----
    return(0);
}

```

---

## Desenhando barras (segmentos de histogramas)

Um dos tipos de indicadores em que todos tem mais dificuldade de montar, são os que utilizam segmentos de barras para serem desenhados, sim, pois o histograma em resumo deve começar seu desenho do 0 (zero), então aqui vai a dica de como desenhar estes tipos de gráficos. Segmentos de Histogramas so podem ser desenhados na janela principal ([indicator\\_chart\\_window](#)), portanto não tente usá-lo em janelas separadas

```

//+-----+
-----+
//|
MeuPrimeiroIndicadorPoderoso.mq4 |
//|           Copyright © 2007, DooMGuard Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2007, MetaQuotes Software
Corp."
#property link "http://www.metaquotes.net"

#property indicator_chart_window
#property indicator_buffers 2

#property indicator_color1 Red           //mudamos a cor para
verde
#property indicator_style1 STYLE_SOLID //queremos uma linha
sólida
#property indicator_width1 3             //espessura da linha = 3

#property indicator_color2 Green        //mudamos a cor para
verde
#property indicator_style3 STYLE_SOLID //queremos uma linha
sólida

double ExtMapBuffer1[];
double ExtMapBuffer2[];

int init()
{
    SetIndexStyle(0,DRAW_HISTOGRAM);
    SetIndexBuffer(0,ExtMapBuffer1);
    SetIndexStyle(1,DRAW_HISTOGRAM);
    SetIndexBuffer(1,ExtMapBuffer2);
    return(0);
}

int deinit()
{
    return(0);
}

int start()
{
    int Itens = Bars - IndicatorCounted(),
        Item;
    for(Item=Itens;Item>=0;Item--) {
        ExtMapBuffer2[Item] = Low[Item];
        ExtMapBuffer1[Item] = High[Item];
    }
    return(0);
}

```

Este programa gerara esta saída no gráfico



Note que as cotações coloquei como gráfico de linhas (roxa), então as barras verdes são plotadas pelo indicador. Se trocar a linha que identifica onde o indicador será desenhado, você terá a seguinte saída:

```
#property indicator_separate_window
```

Compile, retire o indicador antigo e anexe o novo no gráfico e você ira obter:





## Criando Consultores Especialistas em linguagem MQL4 - Parte I (Experts Advisors)

---

Bem agora que já sabemos como criar indicadores, se faz necessário, fazê-los trabalhar para nós. Ou seja, o indicador simplesmente mostra para você alguma informação importante no mercado. Porém, não entra e sai do mercado segundo sua própria teoria. É para isto que existem os **Consultores Especialistas**. Como vimos anteriormente, os Consultores Especialistas, nada mais são que automatizações dos métodos e regras que utilizamos para determinar o melhor momento de entrar e sair do mercado. Neste ponto se abre-se um novo mundo para todos nós, no que diz respeito a programação em MQL4.

---

### Qual o Método que utilizaremos?

Estive pensando, qual seria o melhor exemplo de um Trade System simples. Muitos apareceram na minha mente, porém resolvi colocar um sistema baseado em Médias Móveis, para ser mais preciso utilizaremos o cruzamentos de duas médias, uma de poucos períodos (a que chamaremos de média curta) e outra de mais períodos (a qual chamaremos de média lenta), por considerá-lo de fácil entendimento, e também porque podemos incrementar o mesmo explorando todo nosso sistema de programação em MQL4.

Nesta lição simplesmente criaremos o Código base para o Consultor, que será gerado pelo assistente e o comentaremos. Porém se faz necessário explicar como identificaremos as medias a serem utilizadas para informar ao assistente na hora de sua criação, aqui me limitarei a colocar em poucas palavras o método e na próxima lição o comentarei com mais detalhes.

O Consultor terá em sua interação com o usuário duas variáveis externas que chamaremos respectivamente de

- **Média Longa** : número de amostragem para a médias que utiliza o maior número de barras. Nossa variável identificadora no consultor será **MALPerido** (**Média Móvel Longa Período**)
- **Média Curta** : número de amostragem para a médias que utiliza o menor número de barras. Nossa variável identificadora no consultor será **MACPerido** (**Média Móvel Curta Período**)

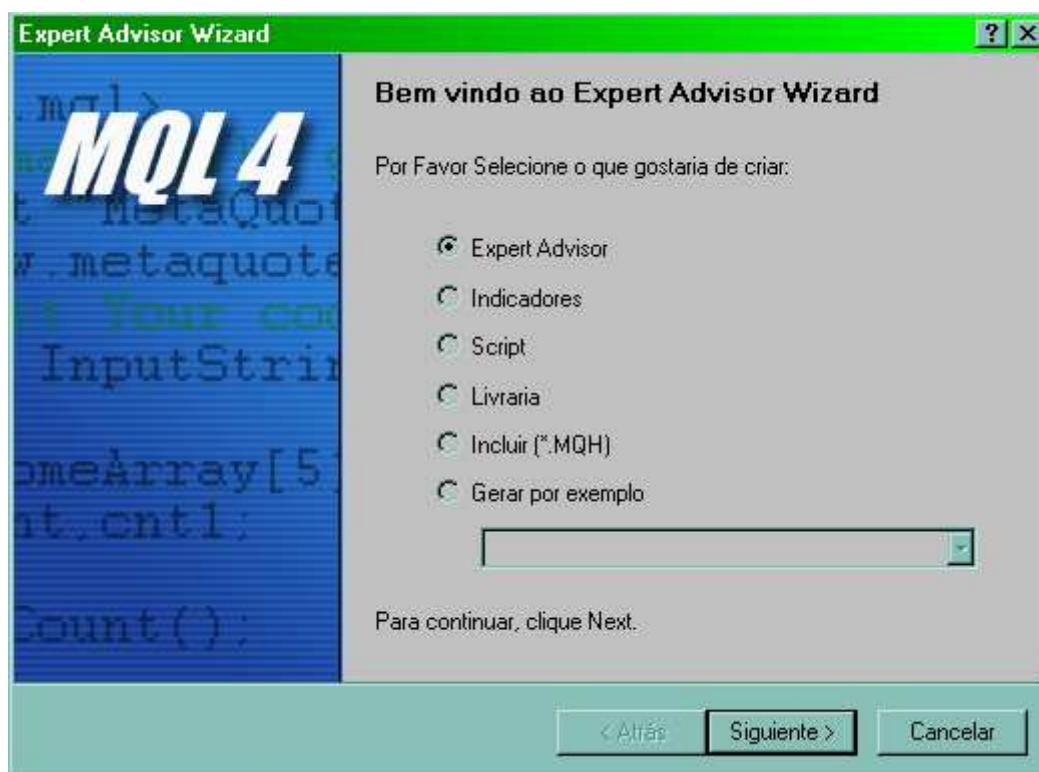


Neste primeiro momento, com a informação apresentada conseguiremos dar início aos nossos trabalhos. Lembre-se, nosso objetivo não é produzir a versão mais sofisticada do indicador, mas sim a mais simples, e ir incrementando o mesmo, para entendermos o processo de desenvolvimento de um Consultor Especialista.

---

### O assistente do MetaEditor

Simplesmente, abra o MetaEditor e inicie um novo arquivo fonte. Aparecerá a janela do assistente do MetaEditor. Escolheremos que queremos montar o código de um Expert Advisor (Consultor Especialista).



O Próximo passo é clicar no botão Seguinte, e aparece a tela onde identificaremos o **nome** do nosso indicador, o **autor** ou nome do proprietário do código, bem como o **link** da página ou do e-mail do mesmo. Aqui também definiremos quais são as variáveis que o usuário pode modificar para configura o Consultor. Como já explicado termos 2 variáveis :

- **MALPerido** (**M**edia **M**óvel **L**enta **P**eríodo): o nosso período Lento é uma variável inteira, pois o indicador de médias móveis que utilizaremos não consegue mensurar meio período ou partes de um período (quando falo período estou falando no tempo que é utilizado para montar cada barra). Usaremos como média lenta o valor 21 períodos, ou as ultimas 21 barras do gráfico no qual o consultor será anexado.

- **MACPerido** (**M**édia **M**óvel **C**urta **P**eríodo): o nosso período Rápido também é uma variável inteira. Usaremos como média curta o valor de 10 períodos, ou as ultimas 10 barras do gráfico no qual o consultor será anexado.

Nota: estes valores serão os valores padrões para o nosso consultor, ou seja, quando o usuário anexar o consultor ao gráfico estes valores aparecem, sendo que se o usuário desejar, poderá modificá-lo.

**Expert Advisor Wizard** [?] [X]

**Propriedades Gerais do Expert Advisor**  
Especifique as Propriedades Gerais do Expert Advisor.

Nome:

Autor:

Link:

Parametros:

Nome	Tipo	Valor inicial
MMLPeriodo	int	21
MMRPeriodo	int	10

Add

Deletar

< Atrás Finalizar Cancelar

Feito isto, clicamos em finalizar e o assistente ira gerar o código inicial de nosso consultor, poupando-nos algum trabalho de digitação.

### O código gerado pelo assistente

```
//+-----+
//+-----+
//|
MeuPrimeiroConsultorVersao1.mq4 |
//|           Copyright © 2006, DoomGuard Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
//+-----+
#property copyright "Copyright © 2006, DoomGuard Sistemas
Especialistas..."
#property link "http://dgforex.50webs.com"

//---- input parameters
extern int MMLPeriodo=21;
extern int MMCPPeriodo=10;

//+-----+
//+-----+
//| expert initialization
function
//+-----+
//+-----+
```

```

int init()
{
    //----

    //----
    return(0);
}

//+-----+
-----+
//| expert deinitialization
function                                     |
//+-----+
-----+
int deinit()
{
    //----

    //----
    return(0);
}

//+-----+
-----+
//| expert start
function                                     |
//+-----+
-----+
int start()
{
    //----

    //----
    return(0);
}
//+-----+
-----+

```

## O cabeçalho do código gerado pelo assistente

```

//+-----+
-----+
//|
MeuPrimeiroConsultorVersao1.mq4 |
//|          Copyright © 2007, DooMGuaRD Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2007, DooMGuaRD Sistemas
Especialistas..."
#property link "http://dgforex.50webs.com"

```

Aqui, novamente, não temos muito segredos, o que o assistente montou no código, nada mais é que as definições e comentários sobre quem escreveu o código.

Note que o que o assessor nada mas fez que copiar uma parte das informações que você colocou na segunda janela do assistente (Autor e Link) como uma propriedade para o compilador, e também montou o código com as funções principais do nosso Consultor Especialista.

---

### As definições no código gerado pelo assistente

```
//---- input parameters
extern int MMLPeriodo=21;
extern int MMCPeriodo=10;
```

O assistente criou as variáveis que informamos na segunda janela. A principio isso não parece ser grande coisa mas ele evitou alguma digitação. Como esta variáveis já estão inicializadas com valores, estes valores serão assumidos como padrões ao se anexar o Consultor Especialista ao gráfico.

---

### O função de inicialização do Consultor

```
//+-----+-----+
-----+
//| expert initialization
function                                     |
//+-----+-----+
-----+
int init()
{
    //----

    //----
    return(0);
}
```

Bem, o assistente, não tem a mínima idéia de como deve funcionar nosso consultor, então, logicamente ele não pode fazer mais nada que criar este corpo de função de inicialização, pois como ele não conhece o método ele não pode prever o que tem de ajustar antes de rodar a primeira vez o código da função principal.

---

## O função de termino do Consultor

```
//+-----+
-----+
//| expert deinitialization
function                                     |
//+-----+
-----+
int deinit()
{
    //----

    //----
    return(0);
}
```

Se o assistente não sabe o que vamos fazer, saberá muito menos o que tem que tirar ou encerrar quando o consultor é retirado da memória. Portanto ele somente cria o corpo da função. Mais uma vez aqui o assistente não pode montar um código maior que esse, pois não sabe o que tem a fazer ao encerrar o Consultor Especialista.

---

## O função de trabalho do Consultor

```
//+-----+
-----+
//| expert start
function                                     |
//+-----+
-----+
int start()
{
    //----

    //----
    return(0);
}
//+-----+
-----+
```

Sem maiores comentários, nós dominamos a idéia do Consultor Especialista e não o assistente, logicamente ele não pode mais montar o código de entrada e saída da função principal.

Bom agora é oficial, sinto que você esta desapontado com o assistente do Consultor Especialista. Sim, porque você deve estar pensando:

Um consultor realiza um trabalho mais complicado e mais importante que um indicador, e o assistente sabe menos sobre ele que sobre o indicador....

Lembre-se o assistente foi concebido para seguir passos de montagem inicial de um código e não para pensar no que você pode querer fazer....

Para isso, usaremos a próxima lição

# Criando Consultores Especialistas em linguagem MQL4 - Parte II (Experts Advisors)

---

## Entendendo nosso Trade System

Como já deve ser de seu conhecimento (se ainda não for, primeiro estude analise técnica para depois se aventurar em programação de Consultores Especialistas), existem indicadores chamados seguidores de tendências e as médias móveis pertencem a esta classe. Esses indicadores possuem uma inércia natural, ou seja, não foram projetados para apontar reversões rapidamente. A primeira informação importante fornecida por uma média móvel é sua inclinação. Uma média móvel ascendente mostra um mercado comprador, enquanto que uma média descendente indica um mercado vendedor. Posicione-se de acordo com o que a média indica, pois ela tende a refletir de maneira adequada o comportamento dos investidores.

Por ser um indicador seguidor de tendência existe um momento no qual não devemos usar as médias móveis.

Que momento é esse?

O mercado seguidamente se coloca em acumulação, movendo-se lateralmente entre limites de preço. Nesses movimentos, a aplicação das médias só é possível em períodos muito curtos e mesmo assim existe uma chance considerável de obtenção de sinais errados, pois não há uma tendência definida a ser seguida.

Quanto maior o período mais suave é o comportamento da média e mais imune a ruídos e movimentos curtos ela estará. No entanto, se for grande demais pode responder de maneira muito lenta à mudanças significativas no mercado

Quanto menor o período de maneira mais próxima a média seguirá os preços. Contudo, se o período for pequeno demais a média estará excessivamente exposta às variações de preços, perdendo sua utilidade como seguidora de tendências.

O cruzamento entre duas médias, uma longa e uma curta, tenta sintetizar o melhor destes dois mundos (longo e Curto). No nosso primeiro consultor usaremos a média longa seja de 21 barras e a curta de 10. Quando a média de 10 superar a de 21 cruzando para cima, tem-se um sinal de compra, de maneira semelhante, cruzando para baixo um sinal de venda é caracterizado. Nesta variação, é importante que ambas estejam inclinadas na mesma direção.

---



## Regras para o nosso primeiro Trade System automatizado

Nosso consultor é um sistema seguidor de tendência com inversão de mercado, ele tenta detectar quando o mercado definiu uma tendência e posiciona um ordem neste sentido, e ao mesmo tempo que essa tendência é aceita por ele, todas as ordens que foram abertas para a tendência anterior são fechadas. A dita tendência é detectada com o uso de médias móveis.

**1 - Sinal de Compra** : quando a média curta cruzar de baixo para cima a média longa.

**2 - Sinal de Venda** : quando a média curta cruzar de cima para baixo a média longa.

**3 - Modo de saída** : Fechar a ordem atual e inverter posição no mercado ao ocorrer um sinal de mudança de tendencia, ou cruzamento.

**4 - Ordens Abertas** : Somente uma ordem aberta.

**5 - Barras de Trabalho** : Barras 1 e 2, ou seja, não usaremos a ultima barra, ou a barra mais a direita do gráfico, pelo simples fato de que a mesma ainda não tem seu período de calculo dos preços finalizado.

Lembre-se, que neste ponto não desenvolveremos um sistema que tente ser o mais eficiente possível, mas sim um sistema que siga as regras propostas para negociações.



Aproveito o desenho para identificar a numeração das barras na matriz de dados do MetaTrader, para os valores de indicadores e cotações. Onde a numero zero (0), é

a barra mais a direita, e a numeração das barras cresce no sentido da direita para a esquerda do gráfico.

---

### Verificando nossas condições iniciais

Bem, para que nosso consultor trabalhe bem, primeiramente devemos nos assegurar que o usuário, informou a média Longa com um período maior que a média Curta, caso ele tenha informado invertido, vamos arrumar esta situação invertendo os valores das variáveis e se o usuário informou esses valores iguais assumiremos a nossa configuração padrão.

Para tanto criaremos duas constantes onde colocaremos os valores das médias padrões, caso, sintamos a necessidade de mudar estes padrões simplesmente mudamos o valor da constante e o programa assumira estes novos valores em todos os lugares do código.

Também definiremos uma string, na qual colocaremos o comentário que aparecerá em cada ordem aberta. Porque criar uma string e não usar a string diretamente, isso é uma questão de praticidade, pois, se necessitar mudar o valor muda-se na definição e automaticamente todo o código já estará arrumado com este novo valor.

Agora chegou a hora de chamar a atenção para uma variável que toda a ordem criada a partir de um consultor deve ter, o **Numero Mágico**. É um numero inteiro, que identifica que determinara ordem foi criada por determinado consultor, desta maneira, se você tem vários consultores rodando no seu MetaTrader, você pode ficar monitorando e usar, somente as ordens que seu consultor criou, sem se preocupar com as outras. A princípio, não calcularemos esse número mágico, mas sim criaremos uma variável com ele para facilitar nosso trabalho inicial. Note que se você colocar estes consultor em dois gráficos, somente um deles ira trabalhar, pois ambos compartilham o mesmo número mágico, logo, como nossa regra é só uma ordem aberta, se tivermos mais de um consultor sendo executado, só um terá ordens abertas. Em uma das lições futuras iremos resolver este problema. No momento, só necessitamos um número mágico.

```
#define ComentarioDoConsultor "Meu EA V1"

#define MediaLonga 21 // Período padrão para a média longa
#define MediaCurta 10 // Período padrão para a média Curta

//---- input parameters
extern int MMLPeriodo = MediaLonga; // Período da media Longa
extern int MMCPeriodo = MediaCurta; // Período da media Curta

int NumeroMagico = 1001; // Nosso número mágico da versão 1
```

Nossa função de inicialização vai prever qualquer falha nos valores do período a ser utilizado, então deve ficar assim

```

int init()
{
    //Testa se limites estão invertidos
    if (MMLPeriodo<MMCPeriodo) {
        // Período do média Longa é menor que a Curta então inverte
os valores
        int Auxiliar = MMLPeriodo; // Guarda valor original de
Período Longo
        MMLPeriodo = MMCPeriodo; // novo valor para Período Longo
        MMCPeriodo = Auxiliar; // novo valor para Período Curto

    }
    // se períodos iguais então assume o padrão
    // pois se as médias forem iguais não existira cruzamento
    if (MMLPeriodo==MMCPeriodo){
        MMLPeriodo = MediaLonga; // novo valor para Período Rápido
        MMCPeriodo = MediaCurta; // novo valor para Período Rápido
    }

    return(0);
}

```

---

## Monitorando o Mercado em busca dos sinais

Bem, não nos resta mais nada além de ficar controlando o mercado para ver se existe alguma condição que respeite as regras que criamos para nosso Consultor Especialista. Simplesmente iremos calcular o valor das médias nas duas barras especificadas nas regras, tanto para a Média Curta e para a Média Longa, e testar se houve o cruzamento das mesmas. Para testar o cruzamento é simples, verificamos se na barra mais a esquerda qual média está acima, se na próxima barra inverter a posição houve um cruzamento.

Caso detectarmos o cruzamento, fecharemos a ordem aberta (se existir), e abriremos a nova ordem na direção em que o mercado irá caminhar (Teoricamente), tudo isso determinado pelos cálculos do nosso consultor. Tudo que discutiremos no resto desta lição deverá fazer parte da função `start()`.

Então, vamos ao trabalho. Criaremos as variáveis de memória que necessitaremos no desenrolar dos cálculos e as inicializaremos de acordo com as necessidades. Quero abrir aqui um parêntese, no que se refere a meu método de programação pessoal. É prática comum quando utilizar o valor calculado por um indicador, usar diretamente a função que calcula o indicador no lugar do código onde o valor deve ser utilizado, eu, por outro lado, prefiro criar uma variável e jogar o valor do cálculo do indicador nela, e somente depois utilizar esse valor a partir da variável mesmo que somente seja usado uma só vez, isso possibilita um maior controle visual do código, além do que se você necessitar este valor mais de uma vez tem o custo da chamada da função de cálculo a cada vez que necessitar deste valor.

Com base nesta última observação, vamos criar as variáveis e inicializá-las com as chamadas de funções necessárias para os valores que queremos.

```

    int Total          = OrdersTotal(), // Numero Total de ordens
no MetaTrader
        NumOrdensBuy  = 0,              // Numero de ordens buy
neste consultor
        NumOrdensSell = 0,              // Numero de ordens sell
neste consultor
        i;                             // utilizado nos loops

    // Calculo das medias para o período lento
    double MML1 = iMA(NULL,0,MMLPeriodo,0,MODE_EMA,PRICE_CLOSE,1);
// barra 1
    double MML2 = iMA(NULL,0,MMLPeriodo,0,MODE_EMA,PRICE_CLOSE,2);
// barra 2
    // Calculo das medias para o período Curto
    double MMC1 = iMA(NULL,0,MMCPeriodo,0,MODE_EMA,PRICE_CLOSE,1);
// barra 1
    double MMC2 = iMA(NULL,0,MMCPeriodo,0,MODE_EMA,PRICE_CLOSE,2);
// barra 2

    // Determina se tem algum cruzamento
    // se Curta abaixo da lenta na barra 2 e
    // Curta acima da lenta na barra 1 entao sinal buy
    bool SinalBuy = (MMC2<MML2) && (MMC1>MML1); // Cruzou para
cima?
    // se Curta acima da lenta na barra 2 e
    // Curta abaixo da lenta na barra 1 entao sinal sell
    bool SinalSell = (MMC2>MML2) && (MMC1<MML1); // Cruzou para
baixo?

```

onde necessito comentar algumas desta variáveis

- MML1 : media Longa calculada na barra 1 (linha laranja no gráfico)
- MML2 : media Longa calculada na barra 2 (linha laranja no gráfico)
- MMC1 : media Curta calculada na barra 1 (linha amarela no gráfico)
- MMC2 : media Curta calculada na barra 2 (linha amarela no gráfico)

Note, que nesta versão do Consultor, não deixamos o usuário mudar o tipo de média e nem o preço a ser utilizado para o cálculo da média, simplesmente queremos usar Média Exponencial e o preço de fechamento de cada barra.

Para existir um sinal Buy a linha amarela (MMC) tem de estar abaixo da linha laranja (MML) na barra 2 e acima na barra 1. Com isso determinamos o cruzamento de cima para cima da linha amarela pela linha laranja.

Para existir um sinal Sell a linha amarela (MMC) tem de estar acima da linha laranja (MML) na barra 2 e abaixo na barra 1. Com isso determinamos o cruzamento de cima para baixo da linha amarela pela linha laranja.

Agora contaremos quantas ordens de cada tipo que nos interessam temos abertas, isso porque, se ja temos uma ordem aberta dependendo do sinal temos de fechá-la ou impedir que abra outra ordem igual. Para isso lançaremos mão de um laço for que percorre todas as ordens abertas no MetaTrader, e logicamente algumas funções auxiliares para determinar se a ordem que estamos trabalhando no momento foi criada pelo nosso Consultor Especialista e se faz parte do Par de moedas do Gráfico onde o Consultor Especialista esta sendo executado.

Dentro do laço, a primeira providencia é seleccionar a ordem que estaremos trabalhando, ao requisitarmos uma ordem no MetaTrader, não necessariamente pode ser a ordem que queremos, por exemplo, a ordem pode ter sido aberta manualmente ou por outro Consultor Especialista, e neste caso não nos interessa esta ordem em particular. Para isso verificamos algumas informações sobre a ordem para saber se ela se enquadra na ordem que nosso Consultor Especialista quer utilizar. Aqui lançaremos mão de dois artifícios, uma perguntamos se a ordem é do par cujo o gráfico o nosso Consultor especialista esta anexado e a outra é que se o número mágico da ordem é o mesmo que o nosso Consultor Especialista definiu como o dele. Note bem, nada garante que 2 consultores especialistas tenham um numero mágico diferente, porem, isso é difícil de acontecer. Bem, vamos contar então

```
// Conta as ordens abertas por este sistema
for(i=0;i<Total;i++) { // passa por todas as ordens abertas
    // seleciono a ordem da lista de ordens
    // pela localização da mesma na lista
    OrderSelect(i,SELECT_BY_POS,MODE_TRADES);
    // se a ordem pertence a este par e tem este número mágico
    if ((OrderSymbol()==Symbol()) &&          // é deste par de
moedas
        (OrderMagicNumber()==NumeroMagico)) { // e é deste
Consultor
    if (OrderType()==OP_BUY) NumOrdensBuy++;
    if (OrderType()==OP_SELL) NumOrdensSell++;
    }
}
```

Agora que determinamos a existência dos sinais e já sabemos se temos ou não ordens abertas e quais os seus tipos, somente nos resta fazer cumprir as demais regras do nosso Consultor Especialista, e a primeira delas é determinar se existe alguma ordem do tipo diferente ao sinal e fechar a posição aberta (se existir), pois segundo nossa regra numero 4 devemos ter somente uma ordem aberta e se existe um sinal para abrir uma ordem diferente da já aberta não podemos deixá-la continuar aberta, independente se tenha ou não lucro. Um dos códigos possíveis e mais indicado para isso é esse:

```

// verifica se tem de fechar alguma ordem e a fecha
for(i=0;i<Total;i++) { // passa por todas as ordens abertas
// seleciono a ordem da lista de ordens
// pela localização da mesma na lista
OrderSelect(i,SELECT_BY_POS,MODE_TRADES);
// se a ordem pertence a este par e tem este número mágico
if ((OrderSymbol()==Symbol()) && // é deste par de
moedas
        (OrderMagicNumber()==NumeroMagico)) { // e é deste
Consultor
        // se tem ordem aberta e for contraria ao sinal obtido
então fecha
        if ((SinalBuy &&(NumOrdensSell>0)) || // Sina Buy com
Ordem Sell Aberta
                (SinalSell&&(NumOrdensBuy>0 ))) // ou Sina Sell com
Ordem Buy Aberta
                // entao fechar essa ordem
                OrderClose(OrderTicket(), OrderLots(),
OrderClosePrice(), 0, Red);
        }
}

```

Com certeza você me perguntará, não podíamos fechar já quando contamos as ordens, ou seja, antes de incrementar determina os o tipo e se ela deve ser fechada, pois assim economizaríamos um laço e seguramente o programa seria executado mais rapidamente. Sim, existe essa possibilidade, mas agora queremos separar bem as etapas para fixar as partes de trabalho do nosso consultor. Isso você poderá fazer como tarefa de casa, se assim o desejar, depois de terminarmos esta primeira versão do nosso consultor.

Bom só nos resta agora determinar, se devemos abrir alguma ordem. Para isso perguntamos se existe um sinal e se não existe alguma ordem aberta na direção que o sinal esta indicando. Caso exista o sinal e não exista ordem nesta direção colocamos uma ordem como pede o sinal. Note que aqui identificaremos nossa ordem com o numero mágico do nosso consultor. Para isso usamos a função:

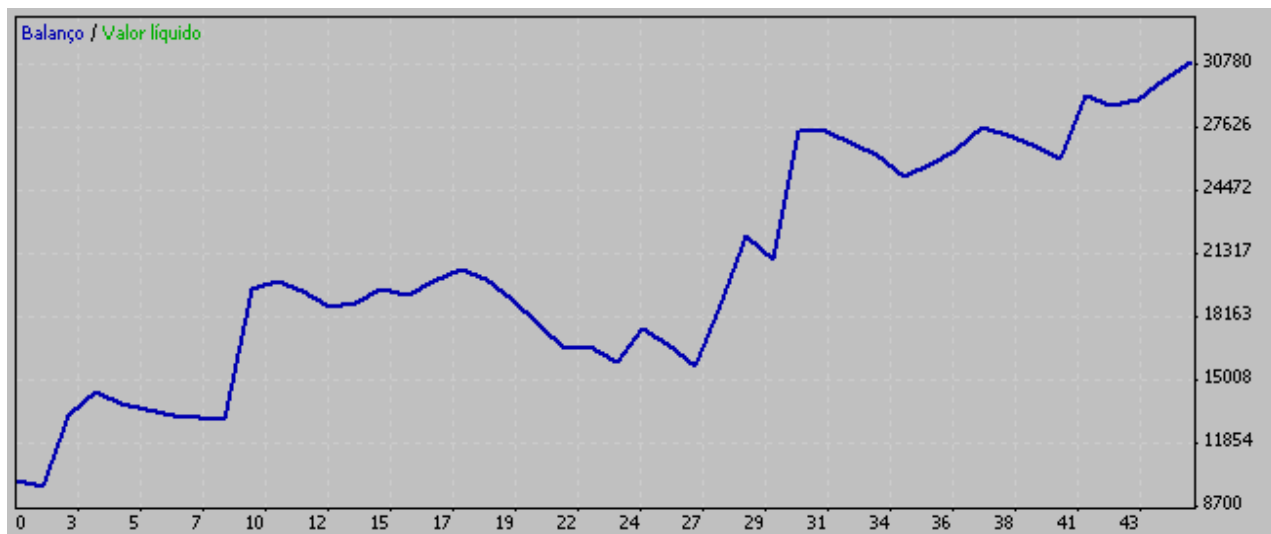
```

// verifica se abre uma posição longa
// deve ter sinal buy e nenhuma ordem buy aberta
if ((SinalBuy) && (NumOrdensBuy==0))
    OrderSend(Symbol(), OP_BUY, 1, Ask, 0,
        0,
        0,
        ComentarioDoConsultor,NumeroMagico, 0, Green);

// verifica se abre uma posição Curta
// deve ter sinal sell e nenhuma ordem sell aberta
if (SinalSell && (NumOrdensSell==0))
    OrderSend(Symbol(), OP_SELL, 1, Bid, 0,
        0,
        0,
        ComentarioDoConsultor, NumeroMagico, 0, Green);

```

Com isso terminamos nossa primeira versão do nosso Consultor Especialista, Abaixo a evolução obtida no strategy tester no gráfico GBPJPY no TimeFrame de 1 hora e no período de Janeiro a Março de 2007, com um saldo inicial de USD 10.000,00.



Nada mau para nosso primeiro teste. Porem, com algumas coisas que acrescentaremos no nosso EA, no próximo capítulo, você verá que o rendimento dele, vai cair, pois as vezes ao tentar nos proteger perdemos oportunidades de maiores lucros, esta claro, estou falando do Take Profit, Stop Loss e Trailing Stop. Quero abrir mais um parêntese neste nosso mundo de testes de Consultores Especialistas, existem alguma coisas que não se pode ter muita confiança, e em minha opinião o Strategy Tester é uma destas coisas. Pois com o tempo você notara que dependendo o método usar o vai ter um resultado diferente e esse resultado é diferente de uma conta real ou conta demo.

---

[Download do código Fonte](#)

[MeuPrimeiroConsultorVersao1.MQ4](#)

Clicar com o botão direito do mouse e no menu escolher Salvar Como.

# Criando Consultores Especialistas em linguagem MQL4 - Parte III (Experts Advisors)

---

## Problemas com o nosso primeiro consultor

Com certeza, nosso consultor ira ser executado e irá trabalhar segundo as regras propostas, ele abrira e fechara ordens de acordo com o método que nós propomos. Porem, propositadamente, o código foi montado, de maneira que podem ocorrer alguns inconvenientes, como por exemplo: se o servidor não conseguiu fechar ou abrir nossa ordem, como faço para avisar o usuário que ocorreu este tipo de problema? Este problema pode ocorrer pois ao abrir a ordem especificamos a variação máxima do preço que informamos, ou se preferir o Slip Page. Se o preço variar mais que o Slip Page esperado a ordem não será aberta, isso pode acontecer em horário de noticias onde o preço varia muito num minúsculo espaço de tempo, ou o servidor recebe muitas ordens simultaneamente, só citando um exemplo. Alem do mais, podemos ter outros problemas como por exemplo, o numero de barras no gráfico ser insuficiente para calcular as médias.

---

## Previendo o numero de barras disponíveis para trabalho

Bom, em via de regra, teremos muitas barras no gráfico para executar nosso Consultor, porem, é interessante não deixar o sistema rodar com um numero de barras menor que o numero de barras que necessitamos para o calculo de nossas médias, então no nosso código, vamos prever isso na função de principal, antes de fazer qualquer cálculo:

```
// verifica se o número de barras é maior que o mínimo necessário
if (Bars<=MediaLonga) {
    Print("O número de barras no gráfico não pode ser menor que
",MediaLonga," Barras");
    return(0);
}
```

## Se uma ordem a ser aberta ou fechada gerar um erro

Bem, até agora supomos que todos os comandos executados no nosso consultor, sejam executados corretamente. Bem, tenho novidades para você, estamos na terra, onde nem tudo ocorre como se fosse no paraíso. Diversas circunstancias podem fazer com que ocorra algum erro no envio de alguma requisição de nosso computador, ao servidor de nossa corretora. E você, claro, quer que o usuário seja informado de alguma anormalidade neste sentido. Para esta, e outras situações, onde nem tudo



ocorre como esperado, o MQL4 tem uma função especial, a **GetLastError**, Esta função retorna o numero do ultimo erro ocorrido no processamento, o qual foi detectado pelo MetaTrader. Bem ele retorna um número (cuja definição está no arquivo de inclusão `stderror.mqh`), mas, e aí, o que significa este número. Bem, para auxiliar melhor você, existe um arquivo de inclusão (`stdlib.mqh`) onde uma função (**ErrorDescription**) retorna a descrição do erro para você poder mostrar ao usuário. Para lançar mão destes recursos prontos, iremos incluir em nosso código estes arquivos

```
//+-----+
-----+
//|
MeuPrimeiroConsultor.mq4 |
//|      Copyright © 2006, DooMGuaRD Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2006, DooMGuaRD Sistemas
Especialistas..."
#property link "http://dgforex.50webs.com"

#include "include\stderror.mqh" // códigos dos erros
#include "include\stdlib.mqh"  // descritor dos erros
```

Agora poderemos utilizá-los em nosso sistema. Primeiramente quando tentamos fechar uma ordem

```
// verifica se tem de fechar alguma ordem e a fecha
for(i=0;i<Total;i++) { // passa por todas as ordens abertas
// se a ordem pertence a este par e tem este número mágico
OrderSelect(i,SELECT_BY_POS,MODE_TRADES);
if ((OrderSymbol()==Symbol()) &&
(OrderMagicNumber()==NumeroMagico)) {
// se tem ordem aberta e for contraria ao sinal obtido
então fecha
if ((SinalBuy&&(NumOrdensSell>0))
|| (SinalSell&&(NumOrdensBuy>0)))
if (!OrderClose(OrderTicket(), OrderLots(),
OrderClosePrice(), 0, Red))
// verifica se houve algum erro na requisição de
modificação
Print(" Erro número :
",ErrorDescription(GetLastError()));
}
}
}
```

Agora quando, tentamos abrir uma ordem. Aqui quero abrir uma observação, como somente enviamos uma ordem de cada vez para o servidor, não existe a necessidade de colocar em dois lugares a mensagem de erro, por isso resolvi colocar no fim do código de requisição das ordens. Bem, a função **OrderSend** retorna -1 se a não conseguir completar a requisição da ordem para o servidor, utilizaremos esta

informação para ver quando existe alguma ordem que não se completou. Para tanto criaremos uma nova variável dentro da função Start(). Vejamos o código

```
int Total          = OrdersTotal(), // Numero Total de ordens
no MetaTrader
    NumOrdensBuy   = 0,              // Numero de ordens buy
neste consultor
    NumOrdensSell = 0,              // Numero de ordens sell
neste consultor
    Ticket         = 0,              // Numero de ordem
requisitada ao servidor
    i;                             // utilizado nos loops
```

```
// verifica se abre uma posição longo
// deve ter sinal buy e nenhuma ordem buy aberta
if ((SinalBuy) && (NumOrdensBuy==0))
    Ticket = OrderSend(Symbol(), OP_BUY, 1, Ask, 0, 0,
0,"Primeiro Consultor",
    NumeroMagico, 0, Green);

// verifica se abre uma posição curta
// deve ter sinal sell e nenhuma ordem sell aberta
if (SinalSell && (NumOrdensSell==0))
    Ticket = OrderSend(Symbol(), OP_SELL, 1, Bid, 0, 0,
0,"Primeiro Consultor",
    NumeroMagico, 0, Green);

// verifica se houve algum erro na requisição da ordem para o
servidor
if (Ticket<0)
    Print(" Erro número : ",ErrorDescription(GetLastError()));
```

Nota, as funções de compra e venda serão modificadas mais adiante, para a inclusão de outra proteções, no código acima apenas illustrei as modificações para acrescenta a mensagem de erro, caso ela ocorra.

---

## Vamos estabelecer objetivos de perdas e ganhos

Bem, embora nosso consultor, tenha o método de reversão para fechar ordens, seria interessante deixar o usuário decidir sobre alguns aspectos de objetivos a atingir. Este objetivos seriam a Perda Máxima (Stop Loss ou SL) que o consultor pode ter numa posição e também o lucro esperado (Take Profit ou TP). Lembre-se que ao definir proteções de Lucro Máximo podemos esta limitando nossos ganhos, porem, podemos ganhar al algumas posições, visto que o mercado tem de reverter a tendência para haver um novo sinal, mas isso, é uma coisa que tem de ser avaliado quanto o real sucesso de limitar os ganhos. Vamos definir as variáveis externas para essas proteções

```

#define MediaLonga 21
#define MediaRapida 10

//---- input parameters
extern int TakeProfit = 300; // objetivo de 300 pips (zero
desabilita)
extern int StopLoss = 75; // perda máxima de 75 pips (zero
desabilita)
extern int MMLPeriodo = MediaLonga; // Muda para usar a
constante Lenta
extern int MMRPeriodo = MediaRapida; // Muda para usar a
constante Rápida

int NumeroMagico = 543621; // Nosso número mágico

```

Agora vamos testar estas duas novas variáveis. Cada corretora tem um valor mínimo para colocar stops, então, simplesmente vamos ver se o valor for menor ou igual ao stop mínimo da corretora desabilitaremos esta facilidade no consultor. Para descobrir o stop mínimo da corretora, faremos uso da função **MarketInfo** na função **init()**

```

int StopMinimo = MarketInfo(Symbol(),MODE_STOPLEVEL);
//Verifica se o Take Profit é valido
if (TakeProfit <= StopMinimo ) TakeProfit = 0;
//Verifica se o Stop Loss é valido
if (StopLoss <= StopMinimo ) StopLoss = 0;

```

Agora que já temos certeza que os valores vão ser aceitos pelo servidor então programaremos estes stops, Como observação, programamos estes valores de SL e TP em pips, para transformá-los em valores monetários da moeda devemos multiplicá-los pelo valor do ponto naquele par, isso se faz mediante o uso da variável pré-definida **Point**, por exemplo : **Point** no EURUSD vale 0.0001, já no GBPJPY vale 0.01. **Point** é a menor variação que um par pode ter, em outras palavras é o valor de um pip para uma unidade de troca do par.

```

// verifica se abre uma posição longa
// deve ter sinal buy e nenhuma ordem buy aberta
if ((SinalBuy) && (NumOrdensBuy==0)) {
    TipoOrd = OP_BUY; // Tipo da ordem a abrir
    Preco = Ask; // preço a utilizar na abertura
    // Calcula lucro Máximo esperado
    if (TakeProfit>0)
        LocalTP = Preco + (TakeProfit * Point);
    // Calcula perda Máxima aceita
    if (StopLoss>0)
        LocalSL = Preco - (StopLoss * Point);
}
// verifica se abre uma posição Curta
// deve ter sinal sell e nenhuma ordem sell aberta
if (SinalSell && (NumOrdensSell==0)) {
    TipoOrd = OP_SELL;
    Preco = Bid;
    // Calcula lucro Máximo esperado
    if (TakeProfit>0)
        LocalTP = Preco - (TakeProfit * Point);
    // Calcula perda Máxima aceita
    if (StopLoss>0)
        LocalSL = Preco + (StopLoss * Point);
}
if (TipoOrd!=-1) {
    // Abre a ordem de acordo com as especificações
    Ticket = OrderSend(Symbol(), TipoOrd, 1, Preco, 0,
                        LocalSL, // Programa perda maxima
                        LocalTP, // Programa lucro máximo
                        ComentarioDoConsultor,
                        NumeroMagico, 0, Green);
    // verifica se houve algum erro na requisição da ordem para
    o servidor
    if (Ticket<0)
        Print(" Erro número : ",ErrorDescription(GetLastError()));
}

```

## Protegendo nossos lucros

Alem de estipular um lucro máximo e uma perda máxima, o usuário do nosso consultor pode querer também proteger seu lucro, ou seja, se a posição começar a ter um lucro mínimo, vamos estipulando um novo valor para o nosso stoploss, par que se fechar por stoploss, tenhamos algum lucro. A essa operação chamamos Trailing Stop. Então, o trailing stop, simples mete verifica se houve um novo valor máximo do lucro em uma ordem ele, a partir deste lucro máximo calcula um novo stop loss. Para isso faremos a mesma coisa que fizemos anteriormente, criaremos uma variável e testaremos se a mesma esta dentro dos limites de stop da corretora

```

extern int TrailingStop = 15; // protege nosso lucro a cada 15
pips de ganhos

```

E o teste se o valor é aceitável

```
//Verifica se o Trailing Stop é valido
if (TrailingStop<=StopMinimo) TrailingStop = 0;
```

Bem, agora que nos certificamos que a variável contém um valor aceito pelo servidor de nossa corretora, vamos proteger nosso lucro. Bem, para isso acrescentaremos o código de proteção onde contamos as ordens controladas pelo nosso consultor. Nele, verificaremos se o lucro é o que queremos para proteger nossa ordem. Necessitamos mais algumas variáveis de trabalho

```
int Total          = OrdersTotal(), // Numero Total de ordens
no MetaTrader
    NumOrdensBuy   = 0,              // Numero de ordens buy
neste consultor
    NumOrdensSell  = 0,              // Numero de ordens sell
neste consultor
    Ticket         = 0,              // Numero de ordem
requisitada ao servidor
    Lucro          = 0,              // Lucro da ordem atual
    Atual          = 0,              // Diferença atual entre SL
e o preço
    TipoOrd        = -1,             // Tipo de ordem a a abri -
1 = nenhuma
    i;                             // utilizado nos loops
double NovoSL      = 0, // Novo valor para o SL
    LocalTP        = 0, // TP calculado para a ordem
    LocalSL        = 0, // SL calculado para a ordem
    Preco;           // preço para abrir a ordem
```

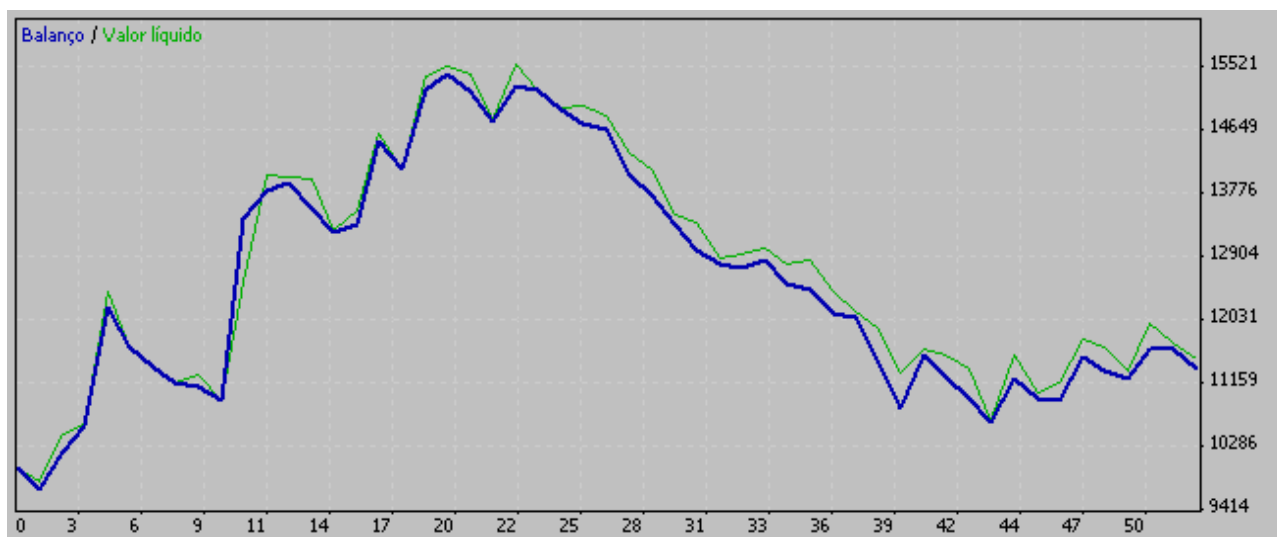
```
// Conta as ordens abertas por este sistema
for(i=0;i<Total;i++) { // passa por todas as ordens abertas
    // seleciono a ordem da lista de ordens
    // pela localização da mesma na lista
    OrderSelect(i,SELECT_BY_POS,MODE_TRADES);
    // se a ordem pertence a este par e tem este número mágico
    if ((OrderSymbol()==Symbol()) &&
(OrderMagicNumber()==NumeroMagico)) {
        // Calcula o lucro da ordem em pips
        Lucro = OrderProfit() / Point;
        // Zera variável para novo SL
        NovoSL = 0;
        if (OrderType()==OP_BUY) {
            NumOrdensBuy++; // Conta se for Buy
            //se tem trailing entao calcula novo SL
            if (TrailingStop>0) {
                // Calcula quantos pips do preço atual esta o SL
                // lembre-se na ordem buy o SL esta sempre abaixo do
preço
                Atual = (OrderClosePrice()-OrderStopLoss())/Point;
                // so usa o trailing se estiver com lucro maior que o
proprio trailing
                // e se o SL esta abaixo do Trailing esperado
                if ((Lucro>TrailingStop) && (Atual>TrailingStop))
                    NovoSL = OrderClosePrice() - (TrailingStop*Point);
            }
        }
        if (OrderType()==OP_SELL) {
            NumOrdensSell++; // Conta se for Sell
            //se tem trailing entao calcula novo SL
```

```

        if (TrailingStop>0) {
            // Calcula quantos pips do preço atual esta o SL
            // lembre-se na ordem sell o SL esta sempre acima do
preço
            Atual = (OrderStopLoss()-OrderClosePrice())/Point;
            // so usa o trailing se estiver com lucro maior que o
proprio trailing
            // e se o SL esta abaixo do Trailing esperado
            if ((Lucro>TrailingStop) && (Atual>TrailingStop))
                NovoSL = OrderClosePrice() + (TrailingStop*Point);
        }
    }
    // Se tiver um novo SL entao programa ele na ordem
    if (NovoSL>0) {
        // tenta modificar a ordem
        bool Ok = OrderModify(OrderTicket(), OrderOpenPrice(),
NovoSL,
        OrderTakeProfit(), 0,Red);
        // verifica se houve algum erro na requisição de
modificação
        if (!Ok) Print(" Erro número :
",ErrorDescription(GetLastError()));
    }
}
}

```

Com isso terminamos nossa segunda versão do nosso Consultor Especialista, Abaixo a evolução obtida no strategy tester no gráfico GBPJPY no TimeFrame de 1 hora e no período de Janeiro a Março de 2007, com um saldo inicial de USD 10.000,00.



Bem, você lembra que falei no fim da lição passada, as proteções podem nos fazer perder algumas chances de lucros maiores, bem, acabamos de comprovar, não mudamos nada no método, apenas colocamos as proteções, e nosso lucro diminuiu consideravelmente. Claro que você deve realizar mais testes e ver qual a configuração ideal para trabalhar com as proteções que colocamos. Em todos os

casos minha dica, é melhor garantir um lucro menor, que apostar numa perda maior...

Se quer comprovar isso, simplesmente informe zero para as variáveis de proteção e rode o teste, Vera que o resultado obtido será mo mesmo da versão 1 do nosso Consultor Especialista. Agora cabe a você determinar os melhores valores de proteção, ou se você quer utilizá-los realmente...

---

### Comentários adicionas sobre estar proteções

Finalmente, você pode prever muitas coisas que podem dar errada em seu consultor, depende apenas de sua imaginação, evitar ou, se não for possível, mostrar ao usuário o erro ocorrido.

---

### Download do código Fonte

#### **MeuPrimeiroConsultorVersao2.MQ4**

Clicar com o botão direito do mouse e no menu escolher Salvar Como.

## Criando Consultores Especialistas em linguagem MQL4 - Parte IV (Experts Advisors)

### Como aumentar o grau de confiabilidade do nosso consultor

Bom criamos, uma consultor que obedece as regras, colocamos algumas proteções para possíveis erros, agora só nos resta melhorar o nosso consultor usando mais ferramentas de análise para que ele possa funcionar melhor. Para tentar ter maior índice de acertos nos sinais, acrescentaremos uma média muito lenta, para que sempre que houver um sinal, se o preço de abertura da vela atual (o de índice zero) estiver abaixo desta média ou acima ele valide um sinal sell ou buy, de acordo com a seguinte regra :

**Sinal de Compra** : quando a média curta cruzar de cima para cima a média longa e o preço de abertura da ultima vela for maior que a média de referencia.

**Sinal de Venda** : quando a média curta cruzar de cima para baixo a média longa e o preço de abertura da ultima vela for menor que a média de referencia.

Nota : nosso objetivo é aumentar o acerto de sinais verdadeiros, porem, não garanto que este seja o melhor método para isso, porem me pareceu mais simples de explicar e utilizar no nosso Consultor. Você Vê que ao longo de um gráfico, perdemos alguns sinais verdadeiros, porem filtramos muitos sinais falsos, então, melhor ganhar menos que perder mais....





A terceira média é um media de muitos períodos (muito mais longa que a longa), onde sempre tenta mostrar a definição de uma tendência a longo prazo, ora, se resolvermos confiar nesta tendência, podemos dizer ao nosso sistema para somente considerar sinais que estão a favor da tendência. No gráfico acima veja que houve dois sinais um de compra (sinalizado) e algum tempo depois outro de venda. Note que os sinais estão acima da terceira media, como a terceira media da uma clara visão de tendência altista, então todos os sinais de venda acima desta media podem ser desconsiderados, pois o mercado baixou, mas pela terceira média ele tende a subir novamente. Com isso queremos eliminar falsos sinais (ou pelo menos a maioria deles). Como a compra que estava aberta não foi fechada, o sistema também ignora o próximo sinal (que seria uma compra) pois ele foi desenhado para ter somente uma ordem aberta por vez, e como já é uma ordem de compra, um sinal de compra não muda nada...

---

### Programando a terceira Media

Na verdade não necessitamos muito trabalho aqui, porem aproveitando, vamos dar mais opções ao usuário de modificar os parâmetros do Consultor Especialista, deixando-o também escolher o tipo de media e o preço a utilizar para o cálculo da média. Para isso mudaremos também os parâmetros de entrada

```
#define ComentarioDoConsultor "Meu EA V3"

#define MediaBase 100 // Período padrão para a média Base
#define MediaLonga 21 // Período padrão para a média longa
#define MediaCurta 10 // Período padrão para a média Curta

//---- input parameters
extern int TakeProfit = 300; // objetivo de 20 pips (zero desabilita)
extern int StopLoss = 75; // perda máxima de 25 pips (zero desabilita)
extern int TrailingStop = 50; // protege nosso lucro a cada 15 pips de ganhos
extern string MMPer = "--- Período das médias";
extern int MMBPeriodo = MediaBase; // Período da media Base
extern int MMLPeriodo = MediaLonga; // Período da media Longa
extern int MMCPeriodo = MediaCurta; // Período da media Curta
extern string MMMet = "--- Metodo das medias";
extern string MMMetSOS = "0=SMA 1=EMA 2=SMMA 3=LEMA";
extern int MMBMetodo = MODE_EMA ; // Metodo para a Média Base
extern int MMLMetodo = MODE_EMA ; // Metodo para a Média Longa
extern int MMCMetodo = MODE_EMA ; // Metodo para a Média Curta
extern string MMPre = "--- Preço das medias";
extern string MMPreSOS = "0=Close 1=Open 2=High 3=Low 4=Median 5=Typical 6=Weighted";
extern int MMBPreco = PRICE_CLOSE; // Preço para a Media Base
extern int MMLPreco = PRICE_CLOSE; // Preço para a Media Longa
extern int MMCPreco = PRICE_CLOSE; // Preço para a Media Curta
```

Note que agora deixamos mais controle do nosso Consultor Especialista ao usuário. Note também, algumas variáveis externas em forma de string, as quais nada mais são para uma separação dos parâmetros de entrada e os valores possíveis em

algumas variáveis, tais como o tipo de média e o preço a utilizar. Agora vamos determinar se o usuário entrou com estes novos valores corretamente, as linhas abaixo devem estar na função `init()`

```
// Verificações para a média base, tem de ser maior que a longa
// se for menor ou igual desabilita o uso da terceira média
// Lembre-se que zero desabilita essa verificação
if (MMBPeriodo<=MMLPeriodo)
MMBPeriodo = 0;

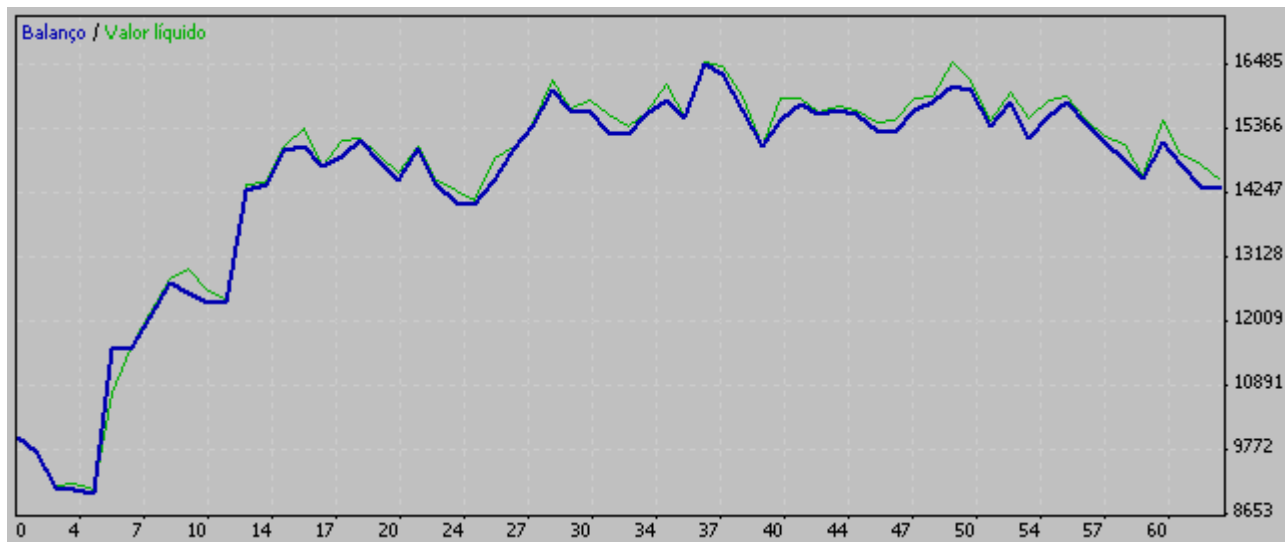
// Verificando os tipos de média em cada período
// se estiver fora dos limites assume método EMA
if ((MMBMetodo<0) || (MMBMetodo>3)) MMBMetodo = MODE_EMA;
if ((MMLMetodo<0) || (MMLMetodo>3)) MMLMetodo = MODE_EMA;
if ((MMCMetodo<0) || (MMCMetodo>3)) MMCMetodo = MODE_EMA;

// Verificando os preços utilizados em cada período são
// válidos
// se estiver fora dos limites assume Preço Close
if ((MMBPreco<0) || (MMBPreco>6)) MMBPreco = PRICE_CLOSE;
if ((MMLPreco<0) || (MMLPreco>6)) MMLPreco = PRICE_CLOSE;
if ((MMCPreco<0) || (MMCPreco>6)) MMCPreco = PRICE_CLOSE;
```

Pronto, nada mais nos resta que verificar se temos que utilizar a terceira média para a determinação de nossos sinais, isto se faz na função `start()` com o seguinte código

```
// se informou média base diferente de zero então usa a
// terceira média
if (MMBPeriodo>0) {
    // Verifica Média Base para validar entrada na barra atual
    double MMB0 = iMA(NULL,0,MMBPeriodo,0,MMBMetodo,MMBPreco,1);
    // barra 0
    // verifica se a terceira média autoriza os sinais
    SinalBuy = SinalBuy && (Close[0]>MMB0);
    SinalSell = SinalBuy && (Close[0]<MMB0);
}
```

Abaixo a evolução obtida no strategy tester no gráfico GBPJPY no TimeFrame de 1 hora e no período de Janeiro a Março de 2007, com um saldo inicial de USD 10.000,00.



É aqui vemos o que a adição de uma simples média de controle de tendência a longo prazo faz com nosso Consultor Especialista. Aumentamos o lucro tentando evitar sinais falsos no sistema. Diminuímos o numero de negociações porém aumentamos o lucro.

Bem, qualquer melhoria que você queira fazer no método do Consultor Especialista, agora é de sua responsabilidade e risco. O que faremos no próximo passo é falar um pouco de gerenciamento de dinheiro.

## Gerenciamento de dinheiro

Programaremos dois tipos de gerenciamento de dinheiro no nosso sistema, um no qual utilizaremos uma quantidade fixa no volume da ordem e outra que usa um percentual do dinheiro disponível, de maneira que se aumentamos nosso saldo nosso volume aplicado numa ordem aumenta e se perdemos em alguma ordem na próxima aplicamos menos. O tipo de gerenciamento que você quer utilizar depende de sua imaginação...

Primeiro criaremos as variáveis nas quais o usuário escolhe o tipo e o valor do gerenciamento a ser utilizado

```
extern string MMRisco = "--- Calculo do Volume";
// zero uma das variáveis abaixo desabilita a mesma
// porem se as duas estiverem zeradas usa o mínimo
// se as duas tiverem ativa usa Lotes
extern double Percentual = 4; // usa 10% da Margem Livre no
calculado do volume
extern double Lotes = 0; // usa lotes fixos no volume
```

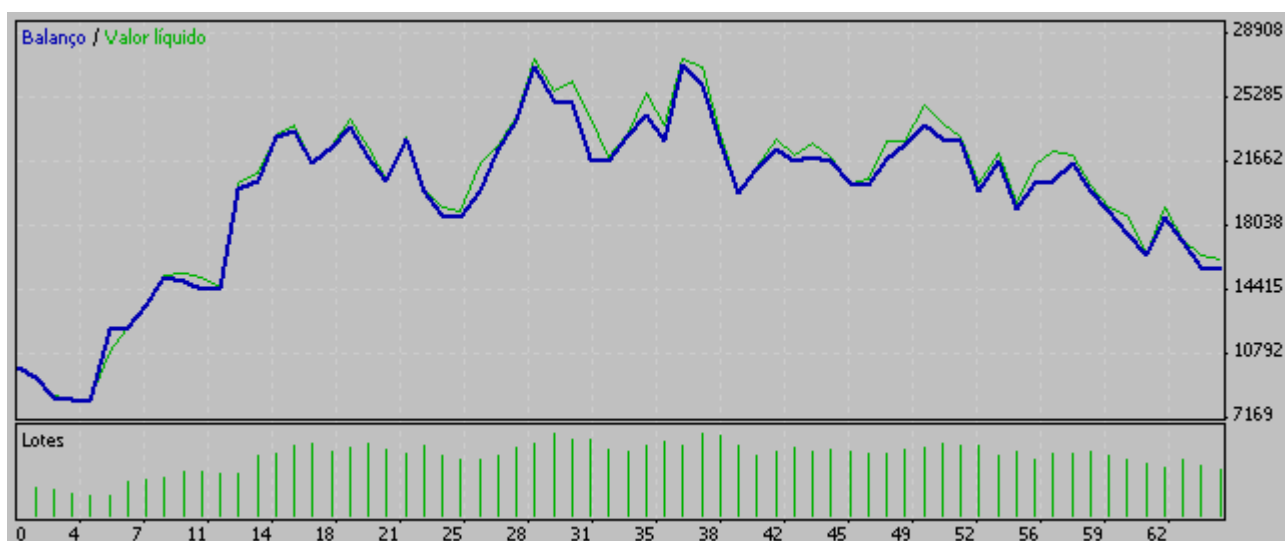
Depois acrescentaremos esse calculo do volume e o ajuste do valor do lote de acordo com o tamanho do lote, a variação mínima que o lote pode ter bem como os máximos e mínimos lotes possíveis de serem utilizados

```

// Calculo o volume da ordem, de acordo com a Margem livre
double LMax = MarketInfo(Symbol(),MODE_MAXLOT), // Lote
Maximo
        LMin = MarketInfo(Symbol(),MODE_MINLOT), // Lote
Minimo
        Lote = LMin, // Valor do
Lote
        MinL = MarketInfo(Symbol(),MODE_LOTSTEP); // Variação
do Lote
// usa percentual da margem
if (Percentual>0) {
    Lote = AccountFreeMargin(); // dinheiro disponível
    Lote = Lote * (Percentual/100); // aplico o percentual
    Lote = Lote * AccountLeverage(); // aplica a alavancagem
    // transforma em lotes
    Lote = Lote ; MarketInfo(Symbol(),MODE_LOTSIZE);
}
// usa lotes
if (Lotes>0)
    Lote = Lotes;
// verifico quantas variações vabem neste lote
Lote = NormalizeDouble(Lote / MinL,0)*MinL;
// verifica limites da corretora
if (Lote<LMin) Lote = LMin;
if (Lote>LMax) Lote = LMax;
// Abre a ordem de acordo com as especificações
Ticket = OrderSend(Symbol(), TipoOrd, Lote, Preco, 0,
                    LocalSL, // Programa perda maxima
                    LocalTP, // Programa lucro máximo
                    ComentarioDoConsultor,
                    NumeroMagico, 0, Green);

```

Como você pode ver no gráfico abaixo, simplesmente, mudando a estratégia do calculo do volume a ser aplicado aumentamos nosso rendimento no período, pois ao utilizar um percentual do dinheiro disponível se ganhamos mais arriscamos mais se perdemos arriscamos menos...



## Comentarios adicionais sobre Consultores Especialistas

Nosso objetivo com este curso, não é montar um infalível sistema de negociações o qual garanta que você sempre saia ganhando. Porem, nosso objetivo é mostrar como fazer um simples consultor, tentar pretejá-lo de alguns erros e tentar aumentar o grau de certeza das posições abertas. Esta evolução com certeza consegui temos mostrar a você. Porem, você, notará que poderá fazer muitas melhorias no sistema que montarmos ate agora, e isto ficará a cargo de sua imaginação, pois já entreguei a você as armas necessárias para conseguir seus objetivos. Apenas aconselho a revisar a apêndice de funções da linguagem MQL4 para se familiarizar com o que você pode contar para suas idéias.

---

## Download do código Fonte

### **MeuPrimeiroConsultorVersao3.MQ4**

Clicar com o botão direito do mouse e no menu escolher Salvar Como.

# Criando Scripts em linguagem MQL4

## Parte I

---

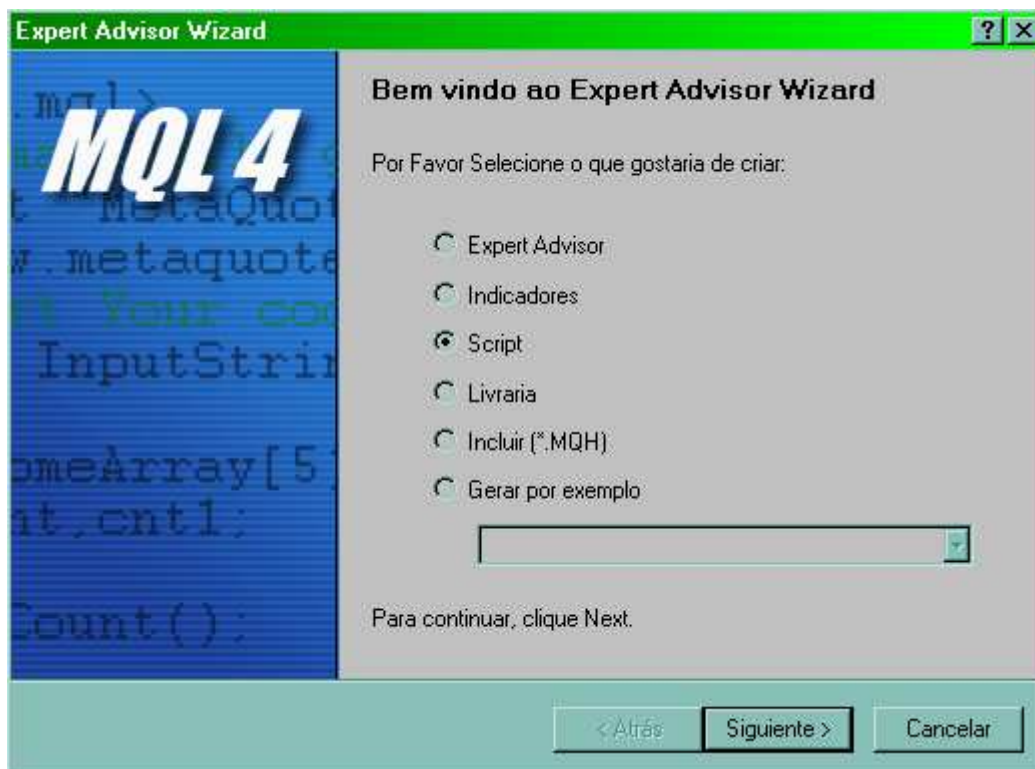
Começaremos agora a ultima parte referente ao aprendizado de programação em MQL4. A programação de Scripts. Como já dito anteriormente o script se executa uma única vez e quando termina seu trabalho, ele é automaticamente removido do sistema. Porém ensinarei a vocês como deixarem um script monitorando seu sistema o tempo todo.

---

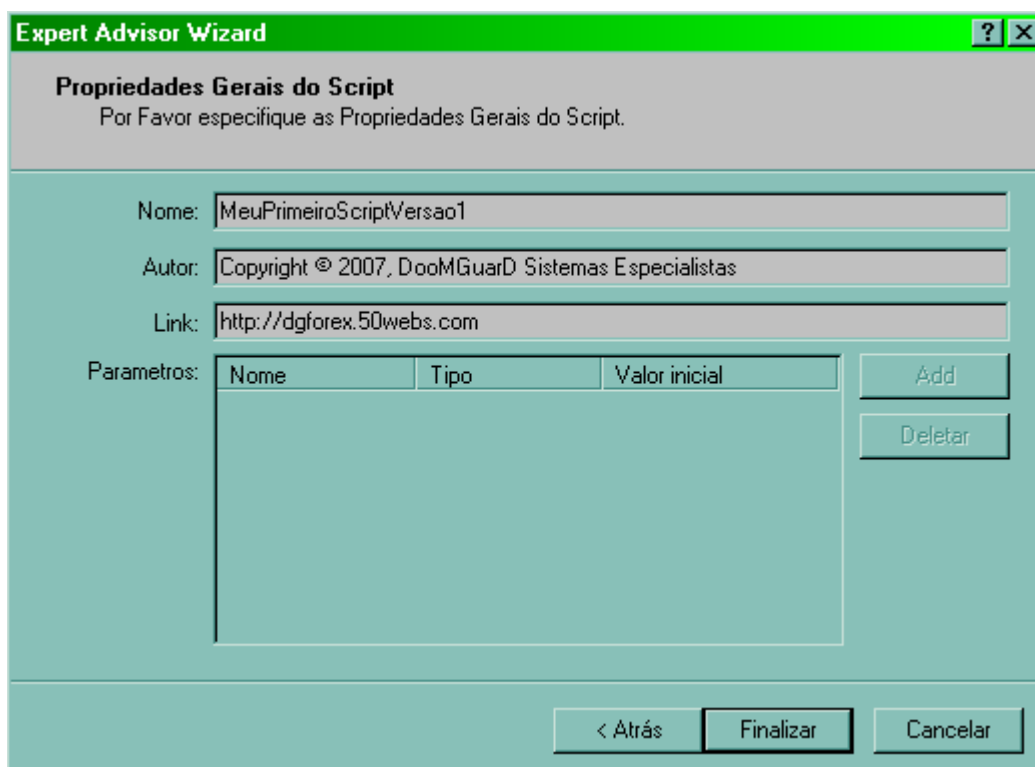
### Q que fará nosso primeiro Script

Depois de meditar um pouco resolvi criar um simples script para colocar ordens Stop. Estas ordens são interessantes para serem utilizadas próximas as notícias que possuem um grande impacto no mercado, como por exemplo o Non-Farm Payroll (NFP). O Método é simples, alguns segundos ou minutos antes da notícia pegar o preço atual e programar 2 ordens, uma BuyStop e outra Sell Stop. A ordem Stop funciona assim, você determina um suporte ou uma resistência e espera que este suporte ou resistência sejam rompidos. No caso de suporte você coloca uma ordem SellStop e ao atingir esse suporte ela se transforma numa ordem Sell, no caso de uma resistência você coloca uma ordem BuyStop ao atingir o suporte a ordem BuyStop se transforma numa ordem Buy.

Novamente Utilizaremos o assistente do MetaEditor para Criar nosso Código inicial. Obviamente escolheremos Script nas opções.



Informamos os demais dados e pronto nosso script esta com o corpo criado



```
//+-----+
-----+
//|
MeuPrimeiroScriptVersao1.mq4 |
//|           Copyright © 2006, DooMGuaRD Sistemas
Especialistas... |
//|
http://dgforex.50webs.com |
//+-----+
-----+
#property copyright "Copyright © 2006, DooMGuaRD Sistemas
Especialistas..."
#property link "http://dgforex.50webs.com"

//+-----+
-----+
//| script program start
function                                     |
//+-----+
-----+
int start()
{
    //----

    //----
    return(0);
}
//+-----+
-----+
```

Como você notou, não temos função de inicialização e nem de finalização, isso porque o script será executado somente uma vez, no momento em que for anexado ao gráfico, e ao terminar seu trabalho ele será encerrado.

---

### Criando os parametros de Trabalho

Necessitamos quatro paramentos básicos para nosso script, porem o usuário não tem como mudar estes parâmetros se não tiver o código fonte do script, visto que, o script não apresenta janela de configuração ao ser colocado no gráfico.

- Distancia de colocação da ordem em pips
- Volume da ordem
- Take Profit
- Stop Loss

Embora o script não necessite, pois usaremos somente uma função, por questão de organização criarei variáveis globais para acomodar os valores das variáveis que necessitamos



```

int      Distancia   = 20; // onde colocar a ordem
int      TakeProfit  = 40; // Lucro esperado
int      StopLoss    = 20; // perda máxima
double   Lotes       = 1;  // lotes das ordens

int NumeroMagico = 2001;

#define ComentarioDoScript "Meu Script V1"

```

agora o código da função principal, mas lembre-se, não estou adicionando nenhuma proteção ou verificação ao sistema, deixarei isso como lição de casa para você.

```

int start()
{
    // Parametros para BuyStop
    double PrecoBuy = Ask + (Distancia * Point);
    double TPBuy    = PrecoBuy + (TakeProfit * Point);
    double SLBuy    = PrecoBuy - (StopLoss * Point);
    // Parametros para SellStop
    double PrecoSell = Bid - (Distancia * Point);
    double TPSell    = PrecoSell - (TakeProfit * Point);
    double SLSell    = PrecoSell + (StopLoss * Point);

    // Coloca Ordem BuyStop
    OrderSend(Symbol(), OP_BUYSTOP, Lotes, PrecoBuy, 3,
              SLBuy, // Programa perda máxima
              TPBuy, // Programa lucro máximo
              ComentarioDoScript,
              NumeroMagico, 0, Green);

    // Coloca Ordem SellStop
    // Coloca Ordem BuyStop
    OrderSend(Symbol(), OP_SELLSTOP, Lotes, PrecoSell, 3,
              SLSell, // Programa perda máxima
              TPSell, // Programa lucro máximo
              ComentarioDoScript,
              NumeroMagico, 0, Green);

    return(0);
}

```

Simples, porem, facilita muito a vida de quem opera com notícias.

Ao adicionar este script no gráfico você verá que ele pega o preço atual, e coloca 2 ordens stops, de acordo com as regras

- Uma ordem BuyStop 20 pips acima do preço
- Uma ordem SellStop 20 pips abaixo do preço

Lógico cada uma com seus respectivos volumes e stops de proteção. Sempre que você desejar modificar os valores terá de compilar o script novamente, visto que o script não possui janela de propriedades como os Indicadores e Consultores especialistas

---

[Download do código Fonte](#)

[\*\*MeuPrimeiroScriptVersão1.MQ4\*\*](#)

Clicar com o botão direito do mouse na linha acima e no menu escolher Salvar Como.

# Criando Scripts em linguagem MQL4

## Parte II

---

### O que fará nosso segundo script

Bem, na lição anterior você viu o script colocar ordens stops. Nesta nova versão acrescentaremos um código que fará que o script espere ate um determinado horário para colocar as ordens e que não seja finalizado enquanto isso não ocorrer. Tomaremos como exemplo o evento que ocorre toda primeira sexta-feira do mês as 3:30pm GMT+0, o Non-Farm Payroll (no Brasil o em horário normal seriam as 12:30pm) . O que faremos é colocar para rodar o script perto do horário e ele se encarregara de colocar o script 30 segundo antes da noticia, e esperara que uma das ordens stop se torne uma ordem não stop, ou seja que ela se concretize e ai eliminará a outra ordem, com isso termina seu trabalho e se descarrega do gráfico.

---

### Idea base de como deixar o script rodando

Bem, aqui, simplesmente deixaremos o sistema ficar esperando a que o horário programado seja ultrapassado, ai o script funcionara normalmente como no exemplo anterior. Para isso necessitaremos guardar o horário que queremos em uma variável. para tanto usarmos uma variável inteira cujo conteúdo será definido como um numero com o seguinte formato HHMMSS onde HH é a hora, MM são os minutos e SS os segundos em que a ordem devera ser colocada.

```
int    Distancia  = 20;      // onde colocar a ordem
int    TakeProfit = 40;      // Lucro esperado
int    StopLoss   = 20;      // perda máxima
double Lotes      = 1;       // lotes das ordens
int    AbrimEm    = 122915;  // abra as ordens as 12:29:15
int    LimitaAbrir = 122945; // porem não abra depois das
12:29:45

int NumeroMagico = 2002;

#define ComentarioDoScript "Meu Script V2"
```

agora vamos simplesmente programar para ele rodar e esperar o horário especificado para o trabalho

```

int start()
{
    // Loop para o controle do horario de trabalho
    int Horario
    bool sair = False;

    while (!Sair) {
        Horario = (TimeHour(TimeLocal()) * 10000) + // Hora HH0000
                  (TimeMinute(TimeLocal()) * 100) + // Minuto MM00
                  TimeSeconds(TimeLocal()); // segundos SS
        // verifica se pode sair do loop e continuar o trabalho
        Sair = Horario >= AbrimEm;
        // adormece o programa por 5 segundos para nao sobrecarregar
o
        // processador
        if (!Sair) sleep(5000);
    }
    // verifica se ja passou o limite final para abrir a ordem
    if (Horario>LimitaAbrir)
        return(0); // se sim entao cai fora sem fazer nada

    // Parametros para BuyStop
    double PrecoBuy = Ask + (Distancia * Point);
    double TPBuy    = PrecoBuy + (TakeProfit * Point);
    double SLBuy    = PrecoBuy - (StopLoss * Point);
    // Parametros para SellStop
    double PrecoSell = Bid - (Distancia * Point);
    double TPSell    = PrecoSell - (TakeProfit * Point);
    double SLSell    = PrecoSell + (StopLoss * Point);

    // Coloca Ordem BuyStop
    OrderSend(Symbol(), OP_BUYSTOP, Lotes, PrecoBuy, 3,
              SLBuy, // Programa perda máxima
              TPBuy, // Programa lucro máximo
              ComentarioDoScript,
              NumeroMagico, 0, Green);

    // Coloca Ordem SellStop
    // Coloca Ordem BuyStop
    OrderSend(Symbol(), OP_SELLSTOP, Lotes, PrecoSell, 3,
              SLSell, // Programa perda máxima
              TPSell, // Programa lucro máximo
              ComentarioDoScript,
              NumeroMagico, 0, Green);

    return(0);
}

```

so chamo atenção para a função `sleep(5000)` ela faz com que o nosso programa fique inativo por 5 segundos, com isso nao ficamos ocupando o processo e liberamos o MetaTrader para outros scripts e indicadores.

Qualquer melhoria que você tenha, fica como tarefa de casa para melhorar suas aptidões de programação de scripts.

---

## Comentarios adicionais sobre Scriptss

Bem, outra coisa que você pode fazer é programar dias e horários para o script executar suas ordens, e neste, caso não deixar que o mesmo se encerre no gráfico. Porém você tem de anexar o script ao gráfico cada vez que iniciar o MetaTrader pois o mesmo não guarda o script como parte integrante do gráfico ao sair do sistema. Este tipo de script fica como "tarefa de casa" para você, como auxílio veja a parte do curso intitulada trabalhando com arquivos, onde você aprenderá a ler e escrever arquivos para salvar e restaurar o contexto de seus códigos, o que preservará o conteúdo caso haja queda de energia ou desligamento acidental do computador

---

## Download do código Fonte

### **MeuPrimeiroScriptVersão2.MQ4**

Clicar com o botão direito do mouse e no menu escolher Salvar Como.

---

## E agora o que temos pela frente

Com esta lição encerramos praticamente o curso de MQL4. Porém, como em outros cursos, isso deixa um vazio muito grande no que diz respeito as dicas de programação e orientações em vários aspectos da programação, para isso nas próximas lições colocarei dicas de trabalhos com cada grupe específico de funções do MQL4, as quais foram projetadas pelos desenvolvedores da linguagem, com isso quero aumentar seu grau de compreensão e de aproveitamento na programação de sistemas com MQL4

# Trabalhando com Arquivos na linguagem MQL4

---

## O que são arquivos e quais os tipos de arquivos

Denomina-se por arquivo todo e qualquer local em que são custodiados documentos, formados a partir da atividade de uma organização. Os documentos, também chamados de informação arquivística, são a geração espontânea em uma empresa ou qualquer localidade. O arquivo constitui um repositório de informações que pode provar alguma coisa, é fonte de evidências de atividades e eventos ocorridos ou regras a serem seguidas. Isso o diferencia da biblioteca, que é um repositório de informações que tem por função instruir o usuário acerca de algum assunto. Existem três tipos de arquivos, assim conceituado segundo tabela de temporalidade, são eles: o arquivo permanente, o arquivo intermediário e o arquivo corrente.

Ja em termos de informática ampliamos um pouco mais este conceito, onde, arquivo como sendo um espaço no disco rígido de um computador, onde os dados são guardados. O arquivo é um agrupamento de registros que seguem uma regra estrutural, e que contém informações (dados) sobre uma área específica. Estes arquivos podem conter informações de qualquer tipo de dados que se possa encontrar em um computador: textos, imagens, vídeos, programas, etc. Geralmente o tipo de informação encontrada dentro de um arquivo pode ser prevista observando-se os últimos caracteres do seu nome, após o último ponto (por exemplo, txt para arquivos de texto sem formatação). Esse conjunto de caracteres é chamado de extensão do arquivo. Como os arquivos em um computador são muitos (só o sistema operacional costuma ter centenas deles), esses arquivos são armazenados em diretórios (também conhecidos como pastas).

Em MQL4 podemos ter 2 tipos de arquivos

Texto: arquivo onde toda a informação são gravadas em forma de um texto sem formatação

Binários : neste tipo os dados são gravados de acordo como são armazenados nos bytes de memória, o que torna um arquivo de leitura mais rápido, porem que o torna difícil de entender se você tentar abrir e ler diretamente num editor de texto tipo notepad

---

## Regras básicas sobre arquivos

Seja o que for que voce tenha em mente para fazer com o trabalho de arquivos, você, não poderá fazer em outros diretórios que não sejam os relacionados abaixo

- `Diretorio_Do_MetaTrader/HISTORY/<current broker>` : para acessar os dados históricos

- Diretorio\_Do\_MetaTrader/TESTER/FILES : para ser utilizado no strategy tester
- Diretorio\_Do\_MetaTrader/EXPERTS/FILES : para os demais casos

Qualquer outro diretório esta terminantemente proibido de trabalhar, e o mesmo não seja acessado pelo programa que voce montar

---

## Como abrir e Fechar um Arquivo

Bem, o primeiro passo para trabalhar com um arquivo é, abrir o mesmo. Para isso o MQL4 possui uma função específica chamada `FileOpen`. Depois que voce fez o que queria, ou seja, leu e/ou gravou os dados, você deve fechar o arquivo e para isso usa-se a função `FileClose`. Exemplo de utilização destas funções

```
int handle;
handle=FileOpen("MeuArquivo.csv",FILE_CSV|FILE_READ,',' );
if(handle>0) {
    // trabalhe com o arquivo
    FileClose(handle);
}
```

`FileOpen` possui os seguintes parâmetros

```
int FileOpen( string filename, int mode, int delimiter=',')

onde
- filename : Nome do arquivo a ser aberto
- Modo      : modo de abertura do arquivo, aqui devemos
               especificar uma combinação que diz o tipo de
               arquivo que queremos abrir e também como iremos
               trabalhar com o arquivo.
               FILE_BIN    : define que o tipo de arquivo será
                           binário
               FILE_CSV    : define que o tipo de arquivo será
                           Texto sem formato
               FILE_WRITE  : define o arquivo será somente para ser
                           escrito, se o arquivo existir ele sera zerado
                           e se não existir será criado.
               FILE_READ   : define que o arquivo será somente
                           leitura, se não existir não poderá ser aberto
- delimiter: quando você grava varias informações em uma linha,
               este parâmetro especifica como serão separadas as
               informações, o padrão é o ponto e vírgula,
               sinceramente, eu nunca mudei esta informação

Regras básicas
- FILE_BIN e FILE_CSV não podem ser utilizados juntos, ou usa um
  ou usa outro, porem um deles deve ser utilizado
- FILE_WRITE e FILE_READ não podem ser utilizados juntos, ou usa
  um ou usa outro, porem um deles deve ser utilizado

a função retorna um numero que o Handle de manipulação do
arquivo, sempre que você quiser se referenciar a
este arquivo use esse numero
```

FileClose somente um parâmetro

```
void FileClose( int handle)
```

onde

- Handle : é o numero do arquivo devolvido pela função FileOpen.

Agora, o número máximo de arquivos que poder estar abertos ao mesmo tempo é 32. Se por algum motivo a função FileOpen não conseguir abrir o arquivo ele retornara -1 e o erro poderá ser verificado com a função GetLastError().

Os handles de um arquivo aberto em um módulo não podem ser passados a outros módulos, por modulo entende-se como um arquivo compilado tipo ex4 ou uma library.

Quando você abrir um arquivo no strategy tester (back test) lembre-se que ele será aberto no **Diretorio\_Do\_MetaTrader/TESTER/FILES** e quando abrir anexado em uma conta demo ou conta real ele estar sendo aberto no **Diretorio\_Do\_MetaTrader/EXPERTS/FILES**

---

## Como gravar dados em um arquivo

Bom, agora ja sabemos abrir e fechar os arquivos, começaremos a aprender como gravar os dados que queremos salvar no arquivo. Isso nos possibilita gravar as variáveis chaves que o nosso sistema usa, e que poderão ser recuperadas, caso haja uma queda de energia por exemplo, o que evita termos de prever um código para executar todos os códigos novamente. Em MQL4 temos diversas funções para trabalhar com a gravação de dados em arquivos, você deve escolher as que mais lhe agradam.

FileWrite:

```
int FileWrite( int handle, ...)
```

exemplo

FileWriteArray:

```
int FileWriteArray( int handle, object array[], int start, int count)
```

exemplo



FileWriteDouble:

```
int FileWriteDouble( int handle, double value, int  
size=DOUBLE_VALUE)
```

exemplo

FileWriteInteger:

```
int FileWriteInteger( int handle, int value, int  
size=LONG_VALUE)
```

exemplo

FileWriteString:

```
int FileWriteString( int handle, string value, int length)
```

exemplo

---

[Como ler \(recuperar\) dados de um arquivo](#)

---

[Comentarios e dicas adicionais sobre o trabalho com arquivos](#)

---