MQL4 COURSE

By Coders' guru
www.forex-tsd.com

-7-

Functions

Welcome to the world of MQL4 Functions.
The functions in any language take two phases:
Learning them which sometimes a boring thing.
Using them which always a **lifeboat.**

I want to tell you my traditional sentence:
I hope you enjoyed the previous lessons, which you can download them from here:

1- Lesson 1 - Welcome to the MQL4 course.
   http://www.forex-tsd.com/attachment.php?attachmentid=372
2- Lesson 2 – SYNTAX.
   http://www.forex-tsd.com/attachment.php?attachmentid=399
3- Lesson 3 - MQL4 Data types.
   http://www.forex-tsd.com/attachment.php?attachmentid=469
4- Lesson 4 - MQL4 Operations & Expressions.
   http://www.forex-tsd.com/attachment.php?attachmentid=481
5- Lesson 5- Loops & Decisions (Part1).
   http://www.forex-tsd.com/attachment.php?attachmentid=504
6- Lesson 6 - Loops & Decisions (Part2).
   http://www.forex-tsd.com/attachment.php?attachmentid=547

Let's start the seventh lesson.

What's the meaning of functions?

The function is very like the sausage **machine**, you input the **meat** and the **spices** and it outs the **sausage**.
The meat and the spices are the **function parameters**; the sausage is the **function return** value. The machine itself is the **function body**.
There's only one difference between the functions and your sausage machine, some of the functions will return nothing (nothing in MQL4 called **void**).

Let's take some examples:

```
double                          // type of the sausage – return value
my_func (double a, double b, double c) // function name and parameters list (meat & spices)
  {
   return (a*b + c);                // sausage outs - returned value
  }
```

As you see above, the function starts with the **type** of the returned value "**double**" followed by the **function name** which followed by parentheses.
Inside the parentheses you put the meat and spices, sorry, you put the **parameters** of the function.
Here we have put three parameters **double a, double b, double c**.
Then the function body starts and ends with braces. In our example the function body will produce the operation **(a*b + c)**.
The **return** keyword is responsible about returning the final result.

Return keyword:

The return keyword terminate the function (like the **break** keyword does in the loop), and it gives the control to the **function caller** (we will know it soon).
The return keyword can include an expression inside its parentheses like the above example **return (a*b + c);** and this means to terminate the function and return the result of the expression.
And it can be without expression and its only job in this case is to terminate the function.

Notice: Not all the functions use the return keyword, especially if there's no return value. Like the next example:

```
void                            // void mean there's no sausage – returned value.
my_func (string s) // function name and parameters list (meat & spices)
  {
     Print(s);
  }
```

The function above will not return value, but it will print the parameter **s** you provided.
When the function has no return value you use "**void**" as the funciotn returns type.
These kinds of functions in some programming language called "**Methods**", but MQL4 calling them functions.

Function call:

We know very well now what the function is (I hope)? How to use the functions in your MQL4?
There's an extra steps after writing your function to use the function in you program.
This step is **calling it** (using it).

Assume you have a function which collects the summation of two integers.
This is the function:

```
int collect (int first_number, int second_number)
  {
     return(first_number+ second_number);
  }
```

You know how the previous function works, but you want to use it.

You use it like this:

```
int a = 10;
int b = 15;
int sum = collect(a,b);
Print (sum);
```

The example above will print **25** (is it a magic). But how did it know?
The magic line is **int sum = collect(a,b);** here you declared a variable (**sum**) to hold the function return value and gave the function its two parameters (**a,b**).
You basically **called the function.**
MQL4 when see your **function name**, it will take you parameters and go to the function and it will return –soon- with the result and place them in same line.
It's very like copying all the lines of the function instead of the place you called the function in, easy right?

Nesting functions inside function:

You can nest function (or more) inside the body of another function. That's because the caller line is treated like any normal statement (it's actually a statement).

For example:

We will use the collect function described above inside another new function which its job is printing the result of the collection:

```
void print_collection (int first_number, int second_number)
  {
     int sum = collect(first_number, second_number);
     Print(sum);
```

```
}
```

Here we called the **collect** function inside the **print_collection** function body and printed the result. **void** means there's no return vale (do you still remember?).

MQL4 Special functions init(), deinit() and start():

In MQL4, every program begins with the function **"init()"** (initialize) and it occurs when you attach your program(Expert advisor or Custom indicator) to the MetaTrader charts or in the case you change the financial symbol or the chart periodicity. And its job is initializing the main variables of your program (you will know about the variables initialization in the next lesson).

When your program finishes its job or you close the chart window or change the financial symbol or the chart periodicity or shutdown MetaTrader terminal, the function **"deinit()"** (de-initialize) will occur.

The third function (which is the most important one) **"start()"** will occur every time new quotations are received , you spend 90 of your programming life inside this function.

We will know a lot about these functions in our **real world lessons** when we write our own Expert advisor and Custom Indictor**.**

I hope you enjoyed the lesson.
I welcome very much the questions and the suggestions.

See you
Coders' Guru
25-10-2005