# User-defined functions



## Introduction

User-defined functions are functions that you write, as opposed to the built-in functions in Pine Script™. They are useful to define calculations that you must do repetitevely, or that you want to isolate from your script's main section of calculations. Think of user-defined functions as a way to extend the capabilities of Pine Script™, when no built-in function will do what you need.

You can write your functions in two ways:

- In a single line, when they are simple, or
- On multiple lines

Functions can be located in two places:

- If a function is only used in one script, you can include it in the script where it is used. See our Style guide for recommendations on where to place functions in your script.
- You can create a Pine Script™ library to include your functions, which makes them reusable in other scripts without having to copy their code. Distinct requirements exist for library functions. They are explained in the page on libraries.

Whether they use one line or multiple lines, user-defined functions have the following characteristics:

- They cannot be embedded. All functions are defined in the script's global scope.
- They do not support recursion. It is **not allowed** for a function to call itself from within its own code.
- The type of the value returned by a function is determined automatically and depends on the type of arguments used in each particular function call.
- Each function call

## Single-line functions

Simple functions can often be written in one line. This is the formal definition of single-line functions:

```
<function_declaration>
    <identifier>(<parameter_list>) => <return_value>

<parameter_list>
    {<parameter_definition>{, <parameter_definition>}}

<parameter_definition>
    [<identifier> = <default_value>]

<return_value>
    <statement> | <expression> | <tuple>
```

Here is an example:

```
f(x, y) => x + y
```

After the function `f()` has been declared, it's possible to call it using different types of arguments:

```
a = f(open, close)
b = f(2, 2)
c = f(open, 2)
```

In the example above, the type of variable `a` is *series* because the arguments are both *series*. The type of variable `b` is *integer* because arguments are both *literal integers*. The type of variable `c` is *series* because the addition of a *series* and *literal integer* produces a *series* result.

## Multi-line functions

Pine Script™ also supports multi-line functions with the following syntax:

```
<identifier>(<parameter_list>) =>
    <local_block>

<identifier>(<list of parameters>) =>
    <variable declaration>
    ...
    <variable declaration or expression>
```

where:

```
<parameter_list>
    {<parameter_definition>{, <parameter_definition>}}

<parameter_definition>
    [<identifier> = <default_value>]
```

The body of a multi-line function consists of several statements. Each statement is placed on a separate line and must be preceded by 1 indentation (4 spaces or 1 tab). The indentation before the statement indicates that it is a part of the body of the function and not part of the script's global scope. After the function's code, the first statement without an indent indicates the body of the function has ended.

Either an expression or a declared variable should be the last statement of the function's body. The result of this expression (or variable) will be the result of the function's call. For example:

```
geom_average(x, y) =>
    a = x*x
    b = y*y
    math.sqrt(a + b)
```

The function `geom_average` has two arguments and creates two variables in the body: `a` and `b`. The last statement calls the function `math.sqrt` (an extraction of the square root). The `geom_average` call will return the value of the last expression: `(math.sqrt(a + b))`.

## Scopes in the script

Variables declared outside the body of a function or of other local blocks belong to the *global* scope. User-declared and built-in functions, as well as built-in variables also belong to the global scope.

Each function has its own *local* scope. All the variables declared within the function, as well as the function's arguments, belong to the scope of that function, meaning that it is impossible to reference them from outside — e.g., from the global scope or the local scope of another function.

On the other hand, since it is possible to refer to any variable or function declared in the global scope from the scope of a function (except for self-referencing recursive calls), one can say that the local scope is embedded into the global scope.

In Pine Script™, nested functions are not allowed, i.e., one cannot declare a function inside another one. All user functions are declared in the global scope. Local scopes cannot intersect with each other.

## Functions that return multiple results

In most cases a function returns only one result, but it is possible to return a list of results (a *tuple*-like result):

```
fun(x, y) =>
    a = x+y
    b = x-y
    [a, b]
```

Special syntax is required for calling such functions:

```
[res0, res1] = fun(open, close)
plot(res0)
plot(res1)
```

## Limitations

User-defined functions can use any of the Pine Script™ built-ins, except: barcolor(), fill(), hline(), indicator(), library(), plot(), plotbar(), plotcandle(), plotchar(), plotshape() and strategy().

Options v: v5