Functions

A function is a group of statements that are run as a single unit when the function is called from another location, such as task main(). Commonly, each function will represent a **specific behavior** in the program.

Functions offer a number of distinct advantages over basic step-by-step coding.

- They save time and space by allowing common behaviors to be written as functions, and then run together as a single statement (rather than re-typing all the individual commands).
- Separating behaviors into different functions allows your code to follow your planning more easily (one function per behavior or even sub-behavior).
- Through the use of parameters, multiple related (but not identical) tasks can be handled with a single, intuitive function.

Using Functions

Functions must be created and then run separately. A function is created by "declaring" it, and run by "calling" it.

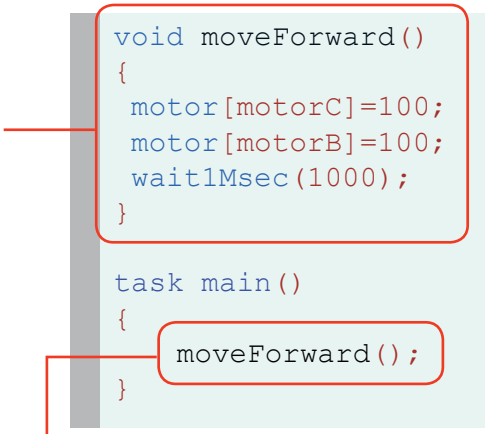
1. Declare Your Function

Declare the function by using the word "void", followed by the name you wish to give to the function. It's helpful to give the function a name that reflects the behavior it will perform.

Within the function's {curly braces}, write the commands exactly as you would normally. When the function is called, it will run the lines between its braces in order, just like task main does with the code between its own braces.

2. Call Your Function

Once your declare your function, it acts like a new command in the language of ROBOTC. To run the function, simply "call" it by name – remember that its name includes the parentheses – followed by a semicolon.



```
void moveForward()
{
 motor[motorC]=100;
 motor[motorB]=100;
 wait1Msec(1000);
}

task main()
{
 moveForward();
}
```

Advanced Functions

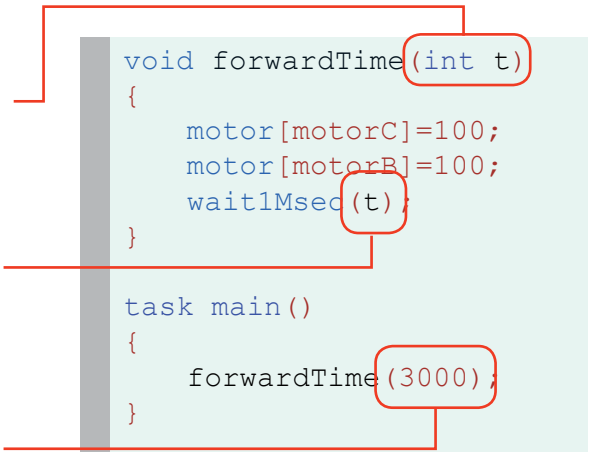
Parameters

Parameters are a way of **passing information into a function**, allowing the function to run its commands differently, depending on the values it is given. It may help to think of the parameters as **placeholders** – all parameters must be filled in with real values when the function is called, so in the places where a parameter appears, it will simply be replaced by its given value.

1. Declare parameter
A parameter is declared in the same way that a variable is (type then name) **inside the parentheses** following the function name.

2. Use parameter
The parameter value behaves like a "placeholder". Whatever value is provided for the parameter when the function is called will appear here.

3. Call function with parameter
When the function is called, a value must be provided within the parentheses to take the place of the parameter inside the function.

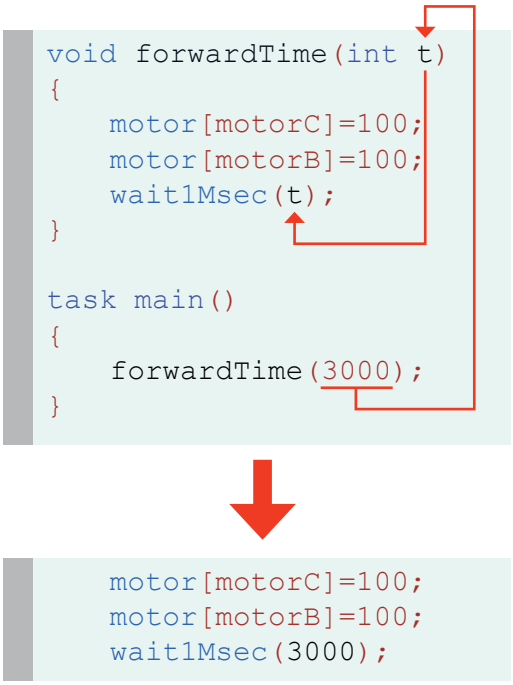


```
void forwardTime(int t)
{
    motor[motorC]=100;
    motor[motorB]=100;
    wait1Msec(t);
}

task main()
{
    forwardTime(3000);
}
```

Substitution
The arrows in the illustration to the right show the general "path" of the value from the place where it is provided in the function call, to where its value is substituted into the function.

The function will run as if the code read as it does in the bottom box.



```
void forwardTime(int t)
{
    motor[motorC]=100;
    motor[motorB]=100;
    wait1Msec(t);
}

task main()
{
    forwardTime(3000);
}
```

```
    motor[motorC]=100;
    motor[motorB]=100;
    wait1Msec(3000);
```

Advanced Functions

Return Values

Not all functions are declared "void". Sometimes, you may wish to capture a mathematical computation in a function, for instance, or perform some other task that requires you to **get information back out of the function** at the end. The function will "return" a value, causing it to behave as if the **function call itself** were a value in the line that called it.

1. Declare return type
Because the function will give a value back at the end, it must be declared with a type other than void, indicating what type of value it will give.

2. Return value
The function must "return" a value. In this case, it is returning the result of a computation, the square of the parameter.

3. Function call becomes a value
The function call itself becomes a value to the part of the program that calls it. Here, it is acting as an integer value for the wait1Msec command.

```
int squareOf(int t)
{
    int sq;
    sq = t * t;
    return sq;
}

task main()
{
    motor[motorC]=100;
    motor[motorB]=100;
    wait1Msec(squareOf(100));
}
```

Substitution
The arrows in the illustration to the right show the general "path" of the value as it is returned and goes back into the main function.

The parameter 100 is used (as t in the function) to calculate the value, but is not shown in the arrows.

The function will run as if the code read as it does in the bottom box.

```
int squareOf(int t)
{
    int sq;
    sq = t * t;
    return sq;
}

task main()
{
    motor[motorC]=100;
    motor[motorB]=100;
    wait1Msec(squareOf(100));
}
```

```
    wait1Msec(10000);
```