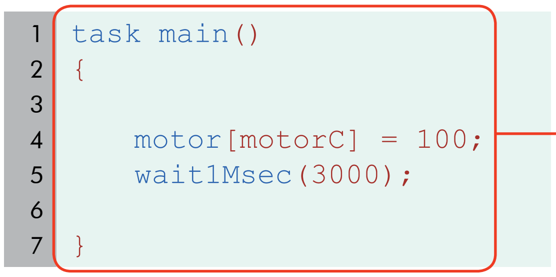
Programming in ROBOTC ROBOTC Rules

In this lesson, you will learn the basic rules for writing ROBOTC programs.

ROBOTC is a text-based programming language based on the standard C programming language.

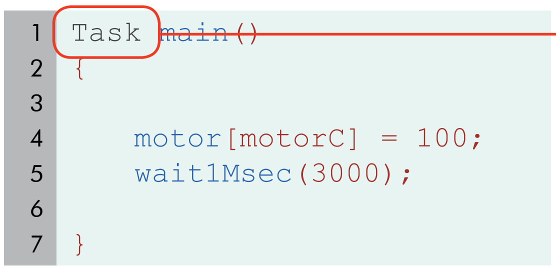
Commands to the robot are written as text on the screen, processed by the ROBOTC compiler into a machine language file, and then loaded onto the robot, where they can be run. Text written as part of a program is called "code".



```
task main()
{

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

}
```

Program Code
Text written as part of a program is called "code".

You type code just like normal text, but you must keep in mind that capitalization is important to the computer. Replacing a lowercase letter with a capital letter or a capital letter with lowercase, will cause the robot to become confused.

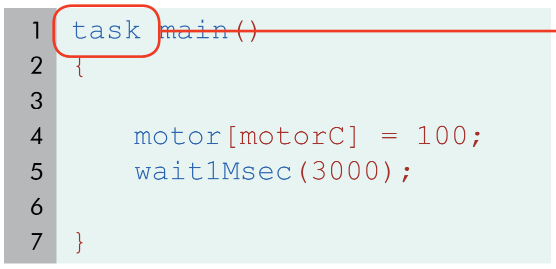


```
Task main()
{

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

}
```

Capitalization
Capitalization (paying attention to UPPERCASE vs. lowercase) is important in ROBOTC.

Capitalizing the 'T' in task causes ROBOTC to no longer recognize this command.

As you type, ROBOTC will try to help you out by coloring the words it recognizes. If a word appears in a different color, it means ROBOTC knows it as an important word in the programming language.



```
task main()
{

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

}
```

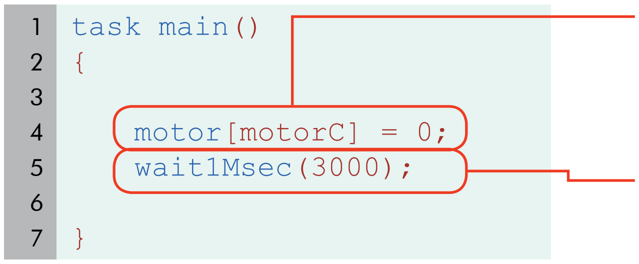
Code coloring
ROBOTC automatically colors key words that it recognizes.

Compare this correctly-capitalized "task" command with the incorrectly-capitalized version in the previous example. The correct one is recognized as a command and turns blue.

Programming in ROBOTC ROBOTC Rules (cont.)

And now, we will look at some of the important parts of the program code itself.

The most basic kind of statement in ROBOTC simply gives a command to the robot. The **`motor[motorC];`** statement in the sample program you downloaded is a simple command. It instructs the motor plugged into the Motor C port to turn on at 100% power.



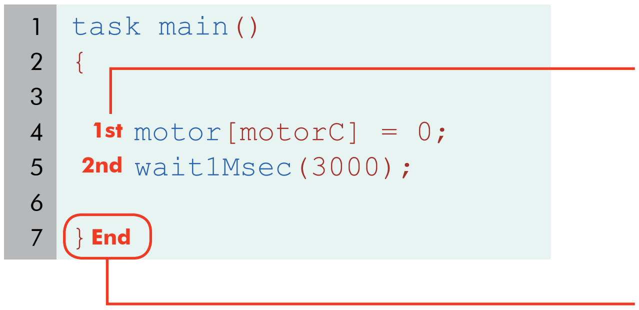
```
task main()
{

    motor[motorC] = 0;
    wait1Msec(3000);

}
```

Simple statement
A straightforward command to the robot.

This statement tells the robot to turn on the motor attached to motor port C at 100% power.

Simple statement (2)
This is also a simple statement. It tells the robot to wait for 3000 milliseconds (3 seconds).

Statements are run in order, as quickly as the robot is able to reach them. Running this program on the robot turns the motor on, then waits for 3000 milliseconds (3 seconds) with the motor still running, and then ends.



```
task main()
{

    motor[motorC] = 0;
    wait1Msec(3000);

}
```

Sequence
Statements run in English reading order (left-to-right, top-to-bottom). As soon as one command is complete, the next runs.

These two statements cause the motors to turn on *(1st command)*, and then the robot immediately begins a three second wait *(2nd command)* while the motors remain on.

End
When the program runs out of statements and reaches the } symbol in task main, all motors stop, and the program ends.

Programming in ROBOTC ROBOTC Rules (cont.)

How did ROBOTC know that these were two separate commands?
Was it because they appeared on two different lines?

No. Spaces and line breaks in ROBOTC are only used to separate words from each other in multi-word commands. Spaces, tabs, and lines don't affect the way a program is interpreted by the machine.

```
task main()
{

    motor[motorC] = 0;
    wait1Msec(3000);

}
```

Whitespace
Spaces, tabs, and line breaks are generally unimportant to ROBOTC and the robot.

They are sometimes needed to separate words in multi-word commands, but are otherwise ignored by the machine.

So why ARE they on separate lines? For the programmer. Programming languages are designed for humans and machines to communicate. Using spaces, tabs, and lines helps the human programmer to read the code more easily. Making good use of spacing in your program is a very good habit for your own sake.

```
task main(){motor[motorC
]=0;wait1Msec(3000);}
```

No Whitespace
To ROBOTC, this program is the same as the last one. To the human programmer, however, this is close to gibberish.

Whitespace is used to help programs be readable to humans.

But what about ROBOTC? How DID it know where one statement ended and the other began? It knew because of the semicolon at the end of each line. Every statement ends with a semicolon. It's like the period at the end of a sentence.

```
task main()
{

    motor[motorC] = 0;
    wait1Msec(3000);

}
```

Semicolons
Like periods in an English sentence, semicolons mark the end of every ROBOTC statement.

Checkpoint
Statements are commands to the robot. Each statement ends in a semicolon so that ROBOTC can identify it, but each is also usually written on its own line to make it easier for humans to read. Statements are run in "reading" order, left to right, top to bottom, and each statement is run as soon as the previous one is complete. When there are no more statements, the program will end.

Programming in ROBOTC ROBOTC Rules (cont.)

ROBOTC uses far more punctuation than English. Punctuation in programming languages is usually used to separate important areas of code from each other. Most ROBOTC punctuation comes in pairs.

Punctuation pairs, like the parentheses and square brackets in these two statements, are used to mark off special areas of code. Every punctuation pair consists of an **"opening"** punctuation mark and a **"closing"** punctuation mark. The punctuation pair designates the area **between them** as having special meaning to the command that they are part of.

```
task main()
{

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

}
```

Punctuation pair: Square brackets [ ]
The code written between the square brackets of the motor command indicate what motor the command should use.

```
task main()
{

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

}
```

Punctuation pair: Parentheses ( )
The code written between the parentheses of the wait1Msec command tell it how many milliseconds to wait.

Checkpoint
Paired punctuation marks are always used together, and surround specific important parts of a statement to set them apart.

Different commands make use of different punctuation. The motor command uses square brackets and the wait1Msec command uses parentheses. This is just the way the commands are set up, and you will have to remember to use the right punctuation with the right commands.

Programming in ROBOTC ROBOTC Rules (cont.)

Simple statements do the work in ROBOTC, but Control Structures do the thinking. These are pieces of code that control the flow of the program's commands, rather than issue direct orders to the robot.

Simple statements can only run one after another in order, but control statements allow the program to **choose the order that statements are run.** For instance, they may choose between two different groups of statements and only run one of them, or sometimes they might repeat a group of statements over and over.

One important structure is the **task main.** Every ROBOTC program includes a special section called task main. This control structure determines what code the robot will run as part of the main program.

```
task main()
{

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

}
```

Control structure: task main
The control structure "task main" directs the program to the main body of the code. When you press "Start" or "Run" on the robot, the program immediately goes to task main and runs its code.

The left and right curly braces { } belong to the task main structure. They surround the commands which will be run in the program.

```
while(SensorValue(touchSensor) == 0)
{
    motor[motorC] = 100;
    motor[motorB] = 100;
}
```

Control structure preview
Another control structure, the while loop, repeats the code between its curly braces { } as long as certain conditions are met.

Normally, statements run only once, but with a while loop, they can be told to repeat over and over for as long as you want!

Checkpoint
Control structures like task main decide which lines of code are run, and when. They control the "flow" of your program, and are vital to your robot's ability to make decisions and respond intelligently to its environment.

Programming in ROBOTC ROBOTC Rules (cont.)

Programming languages are meant to be readable by both humans and machines. Sometimes, the programmer needs to leave a note for human readers to help understand what the code is doing. For this, ROBOTC allows "comments" to be made.

Comments are text that the computer will ignore. A comment can therefore contain notes, messages, and symbols that may help a human, but would be meaningless to the computer. ROBOTC will simply skip over them. Comments appear in green in ROBOTC.

```
// Move motor C forward with 100% power

task main()
{

    /*
      Motor C forward with 100% power
      Do this for 3 seconds
    */

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

}
```

Comments: // Single line
Any section of text that follows a //double slash on a line, is considered a comment, although code to the left of the // is still considered normal.

Comments: /* Any length */
A comment can be created in ROBOTC using another type of paired punctuation, which starts with /* and ends with */

This type of comment can span multiple lines, so be sure to include both the opening and closing marks!

End of Section
What you have just seen are some of the primary features of the ROBOTC language. **Code** is entered as text, which builds **statements.** Statements are used to issue commands to the robots. **Control structures** decide which statements to run at what times. **Punctuation,** both single like **semicolons** and **paired like parentheses,** are used to set apart important parts of commands.

A number of features in ROBOTC code are designed to help the human, rather than the computer. **Comments** let programmers leave notes for themselves and others, and **whitespace** like tabs and spaces helps to keep your code organized and readable.