Microbit Rover Direct Control

In this exercise you will be creating all the code to maneuver a rover on a Microbit. This will be inserted into the connector on the rover and when switched on follow your commands. Or not, as the case may be!

Initial Connections and Code

Connecting a Microbit

Connect the Microbit with the short USB to Micro-USB connector to any available USB port on the computer. Wait a bit and a pop up message will appear. Cancel this. Press Flash and upload some blank code and a “Copied code onto Microbit” message should appear at the bottom right, as shown above. If not, wait a moment. Then clear any error messages that appear and the pop up box again. Then retry. The code will usually reload the second time.

First Code - Hello World

The classic first piece of code to write with any new language is to write out the message Hello world. This will test everything is working, as well as demonstrate some simple coding. Type this code into the editor and flash onto the Microbit. Do not copy and paste from these instructions. It doesn’t work!

```
from microbit import *
display.scroll("Hello, world!")
```

If successful, try changing the message to something else. If not successful did you forget the asterix * or misspell a word? The (“ “) are important too.

Connect the Rover Motors to the Code

Each of the two motors on the rover has three wires to control it. The first called the Enable is the on and off switch. If this wire is on the motor is able to do something. The other two control direction. Forwards or backwards. If one is switched on, that is the direction the rover will move in.

This table links the switching to the possible outcomes, a truth table. You will find this useful later.

Enable	**Forward**	**Reverse**	**Result**
OFF	OFF or ON	OFF or ON	**Stop**
ON	ON	OFF	**Forwards**
ON	OFF	ON	**Reverse**
ON	OFF	OFF	**Fast Stop**
ON	ON	ON	**Fast Stop**

Also here are the complete wiring connections from the Microbit to the rover.

Motor	**Rev Left**	**Fwd Left**	**Enable Left**	**Enable Right**	**Rev Right**	**Fwd Right**
Pin No.	16	14	13	2	8	1

That is all the basic information required. So now to the code.

Each connection will require a name. These are called variables in coding. Normally because they can be changed within the program running. Variable names should be a single word, not start with a number and be a meaningful name. Single letters or weird abbreviations are frowned upon. Anyone reading the code should be able to understand what the variable is for.

So forwards_right is good, fr not so.

Alter the code you already have to match this. The line with hash is a comment. A message to the human reader and is not code that does anything. A hash can be used to stop a line of code working to test something out.

```
from microbit import *

# define named variables for the motor pin connections
enable_left_motor = pin13
enable_right_motor = pin2

forwards_right = pin1
backwards_right = pin8

forwards_left = pin14
backwards_left = pin16
```

Flash the code onto the Microbit. Nothing should happen. There should not be any errors. Check your code against that above if there are.

First Movement

To get the rover to move requires the right connections to the motors to be switched on. The Microbit uses either a one (1) for on and a zero (0) for off.

This line will switch on the left motor. But not make it move. For that you require direction as well.

```
enable_left_motor.write_digital(1)
```

Without looking at the next page are you able to write the next five lines of code needed to make the rover move forward? Use the truth table above to work out which connections need to be on or off. You should have five additional lines similar to the one above with either a zero or one in the brackets.

Here are the completed lines. Flash your code onto the Microbit. Hopefully you will have no errors showing. If nothing happens, disconnect the Microbit and put it into the rover as you have been shown. The rover should move forwards.

```
enable_left_motor.write_digital(1)
enable_right_motor.write_digital(1)
forwards_right.write_digital(1)
backwards_right.write_digital(0)
forwards_left.write_digital(1)
backwards_left.write_digital(0)
```

Functions to Make the Code Easier to Read and Write

A function is a block of code that can be reused again and again. In our example to make the rover stop after going forward would require another six lines of code. And another six to make it go forward again. With functions all of that can be reduced to three lines.

A function is defined (def) with a name that is meaningful. Just like a variable. So to make the rover go forwards a function could be named forwards. It also requires two brackets afterwards and a colon.

```
def forwards():
    |
```

If you put this in the editor and then press enter you should see the cursor move to the indented position as indicated in the line above. The indent distance is one tab or 4 spaces. In Python any code that is indented will belong to the function line with the colon preceding it.

So now you can cut and paste your code to make the rover go forwards into the function, like this:

```
def forwards():
    enable_left_motor.write_digital(1)
    enable_right_motor.write_digital(1)
    forwards_right.write_digital(1)
    backwards_right.write_digital(0)
    forwards_left.write_digital(1)
    backwards_left.write_digital(0)
```

To make a function work it is called into action with just its name and two brackets.

```
forwards()
```

Add the call to your code, flash to the Microbit and make sure it is error free.

Time for Sleep

In a moment your code will also have a stop command. But if you ran your code with a forwards then a stop nothing would happen. The code will run so fast the rover will not have time to move before the stop is called. A sleep command will pause the code for a while during which the motors, that will have been turned on, move the rover forward. The sleep values are in milliseconds. So one second is 1000.

```
sleep(1000)
```

BUT before you add that to your code you need to create a stop function. Can you do that? **Place the stop function above the forward() call line**. All the functions need to be written above all the calls for them to work.

Then place the sleep command under the forward call followed by a stop call. Change the sleep call to 3 seconds to make the rover drive forwards for three seconds before stopping. Flash your code onto the Microbit and try it on the rover.

Our complete code is on the next page if you need it.

Complete code so far.

```
from microbit import *

# define named variables for the motor pin connections
enable_left_motor = pin13
enable_right_motor = pin2

forwards_right = pin1
backwards_right = pin8

forwards_left = pin14
backwards_left = pin16

# rover forwards function
def forwards():
    enable_left_motor.write_digital(1)
    enable_right_motor.write_digital(1)
    forwards_right.write_digital(1)
    backwards_right.write_digital(0)
    forwards_left.write_digital(1)
    backwards_left.write_digital(0)

# rover fast stop function
def stop():
    enable_left_motor.write_digital(1)
    enable_right_motor.write_digital(1)
    forwards_right.write_digital(0)
    backwards_right.write_digital(0)
    forwards_left.write_digital(0)
    backwards_left.write_digital(0)

forwards()
sleep(3000)
stop()
```

On Your Own

You now have all the basics needed to control the rover in any direction you require. Can you write functions to make it turn left and right or reverse? Then drive in a pattern designed by yourself?

There is an additional instruction on the next page that can make your code easier to write.

Additional Instruction - Slightly Advanced

The functions can be used with a sleep command within them. And the length of time of the sleep can be adjusted at every call of the function. The stop function below has been rewritten to allow a 2500 millisecond sleep.

```
def stop(delaytime):
    enable_left_motor.write_digital(1)
    enable_right_motor.write_digital(1)
    forwards_right.write_digital(0)
    backwards_right.write_digital(0)
    forwards_left.write_digital(0)
    backwards_left.write_digital(0)
    sleep(delaytime)

stop(2500)
```

The value of 2500 is sent to the function from the call. The value is then assigned to the variable delaytime. Rather like your variables for the wiring connections. When the code gets to the line sleep(delaytime) the value is substituted into brackets. Remember delaytime is just a variable name. Tortoise or hare would also work just as well but not be as meaningful!

Using this method all your calls for forward, left, right, stop and reverse can have the sleep within the call. Again saving on lines of code and making it quicker to write.

Try it!