MicroBit Radio Controlled Rover

This exercise has been designed to enable you to work out what there is to be done. We are not giving you the answers, only some guidance. We would like you to take the lead and persevere in getting this to work.

The aim is to be able to control the rover by remote control from one MicroBit to another attached to the rover. The programming will be done using MicroPython in the Mu editor. You should not be doing this exercise if you have not done some programming with the MicroBit before.

The Kit

You will be provided with:
2 x MicroBit
1 x USB to Micro USB Lead
1 x Rover
1 x MicroBit Battery pack

The rover has been set up with an adapter so that the Microbit can be plugged into the rover on place of the normal Arduino board used in our work experiences.

The rover consists of two motors with three wires coming from each. By changing the direction of the electric current to the motor the wheels will rotate either forwards or backwards. The driver board on the rover has three wires for controlling the motor by the MicroBit. Two are for the direction control, forwards or backwards, the third is for enabling the motor. Enable is the actual power on/off to each motor. To make the motor rotate forwards requires the forward wire to be on and the enable too with the reverse being off.

Reverse also requires the enable to be on as well. If both the forward and reverse are both on or both off the motor will not turn. It will also not turn if the enable is switched off. The truth table below may help in understanding this.

Enable	**Forwards**	**Reverse**	**Result**
Off / 0 (zero) and	On or Off and	On or Off	**No movement**
On / 1 and	On / 1 and	On / 1	**No Movement**
On / 1 and	Off / 0 (zero) and	Off / 0 (zero)	**No Movement**
On / 1 and	On / 1 and	Off / 0 (zero)	**Forwards**
On / 1 and	Off / 0 (zero) and	On / 1	**Reverse**

To make the rover turn there are two ways. Turn one motor on only or turn one on forwards and the other backwards. One will produce a skidding turn the other a rotation. How and what you do is up to you.

The Wiring of the Rover and Microbit

The pin connections between the MicroBit and rover are as follows:

Pin Number on Microbit	Connection on Driver Board
pin2	Right Enable
pin13	Left Enable
pin1	Right Forward
pin14	Left Forward
pin8	Right Backward
pin16	Left Backward

Some Code Basics

Here are some code snippets to get you started.

You should already know about comments in Python code. These start with a # and are there for you to read, not the computer which ignores them.

Getting a Motor to Turn.

Remember to use the two tables above for all the combinations and pin numbers.

When running the code on the rover place the rover on the floor before switching on. This will prevent the rover driving suddenly off the desk and being damaged.

```
from microbit import *

# A basic drive forwards. Change the bracket values to change motion direction.

pin2.write_digital(1) # Right Enable on. 1(one)is on, 0 (zero) is off.
pin13.write_digital(1) # Left Enable on

pin1.write_digital(1) # Right forward on
pin8.write_digital(0) # Right backward off

pin14.write_digital(1) # Left forward on
pin16.write_digital(0) # Left backward off
```

Use Functions to Shorten and Make the Code Neater

The code below can make the code over all easier to read and better in the long run. A function can be set up for each forward, backwards, stop, left and right. Then a sequence of calls can be written to make each happen in turn. Functions are put in the code after the imports and before the calls to run them.

Give them a suitable name that means what they do. The name must be one word not starting with a number. Multiple word function names can be written like this, roverForward or rover_forward.

Ask for help if you need to on this. If you get it right it will make you a much better coder.

```
def myFunctionName():
      # Code is 4 spaces or one tab indented in from the left edge
      # Any code indented belongs to the function
      Code goes here
      More code goes here
      -
      -
      -
myFunctionName() # Call the function to make it run somewhere later in the code
```

A Delay or Pause

To make a piece of code run for a length of time or to wait until the next piece works you will need a pause. Useful for running a motor for a set amount of time only. A sleep is timed in milliseconds. There are 1000 to a second. This is different to normal Python where a sleep 1 is one second. So a pause for the MicroBit is written like this:

```
sleep(1000)      # A one second pause
sleep(500)       # A half second pause
```

Use Variables to Name Pins

Naming the pins using variables makes the code easier to read through and remember what it is that it does.

What does pin8 do? Naming it as right_backwards would be better. Name a variable like this:

```
right_backwards = pin8
```

Then use

```
right_backwards.digital_write(1)
```

as the command to turn that motor direction on.

Make A Start

So you have all the basics there to get the rover to move under your code. Try and write some code to get the rover to make all the basic moves needed, forwards, backwards, left, right and stop. You will need to use the sleep to pause the code in between moves. Once you are ready put the rover on the floor to drive.

Good luck!

And don't forget to save your code as you go along.

Adding Radio Control

The next stage is to be able to send messages from one MicroBit to the other. And when a message is received for the one on the rover to make the rover respond.

The radio is easy to use. It just requires it to be turned on and then for a message to be sent, something a bit like this. **This is only an example, do not type it!**

```
from microbit import *
import radio

# this next line is not needed if you are the only one using the rovers
radio.config(channel=7) # There are 99 radio channels

radio.on()
radio.send("F")
```

And the other MicroBit needs to be listening.

```
from microbit import *
import radio

radio.config(channel=7) # the same channel number as the remote
radio.on()

message = radio.receive()
display.show(message) # Display the message F
```

That **code above will not quite work** as the message will only be sent once when the Microbit is turned on and the other Microbit needs to be listening just at the right time. Which it won't be because it only listens once when it is turned on too. So you now need to get the units to be listening all the time and a message to be sent just when you need it to be.

Also the radio.config line is only needed if there is more than one of you using the radio at the same time. Just select different channel numbers between 1 and 99 making sure you use the same number on both of your MicroBits but not the same as another coder.
The next page explains it in more detail. Read on.

Amend the Rover Code

You will need a loop of code that is forever listening. Then if a message is received then do something. The MicroBit on the rover will need something added like this:

```
from microbit import *
import radio

radio.on()

while True: # while this is True do what follows. It is, so it does!
      # this is indented like before
      message = radio.receive()
      if message == “F”: # If the message received is the same as the letter F
            forwards() # call function forwards. This is double indented
      # Then loop back and listen again & again
```

This code checks all the time for a message and if one is received then it is assigned to the variable message. If this variable message is the same as (==) the letter F then do what you ask. In this case F for forwards. But remember to define (def) a function called forwards as well. Or instead of calling the function, write the lines of code needed to turn the motor on to make the rover go forwards. How you do it is up to you but we’d prefer it if you used functions.

You will require checks for all the movements of the rover within the while True loop.

The Remote Control

On the MicroBit for the remote control there are only two buttons. So to use them for left and right could be good. But how to command forwards and backwards. What the MicroBit does have is an accelerometer measuring how the MicroBit is tilted in four directions. This would make a good control mechanism.

Some code to start you off would be:

```
from microbit import *
import radio

radio.on()

while True: # as above we need to check the position all the time
      # check if MicroBit is held upwards
      if accelerometer.is_gesture("up"):
            # send an F as the message matching the rover code
            radio.send("F")
            # put an arrow south on the Led display
            display.show(Image.ARROW_S )

      # elif means else if as the next check if above is not true
      elif accelerometer.is_gesture ???????
            # add more checks here to check each down, left or right
            # for backwards, left & right

      # As a catch-all if none of the above are True do this
      else:
            # send an S for Stop if no direction is detected
            radio.send("S")
            # display something?
      display.clear()
      sleep(10 ) # a very short delay to keep the signal steady.
```

From the code above we have introduced an else if check. You'll need more code after the if to check for the other positions which are down, left and right. The other arrows are N (north), W (west) and of course E (east). Adding the images to the receiver MicroBit will let you know that the message has been received even if the motor does not move. A handy check to see that the messages are getting through.

So now try and get your rover remotely controlled.

Finally once you have a working remote control rover you may notice that it does not travel in a very straight line. This is because though the two motors are identical one is actually running in the reverse direction to the other. If you turn the rover over and look you might realise this. Motors do not work as well in reverse as forwards so therefore the rover slowly makes an arc instead of a straight line.
You can correct for this by slowing the speed of the faster motor. This following code will also help you to control the speed of the rover as well. Up to now the rover has been working at full speed.
The secret of speed control lies in the enable pin and the driver board.

When the enable is on it is on, when off it is off. If it were possible to turn the enable pin on and off quickly then it might look like the rover was moving slower. Try to imagine it this way.

If the enable pin is on for a second then off, then on again the motor will be running just 2 seconds in three. On, off, on! Because the motor has only run for two out of three seconds then it will only travel two thirds the distance.
But the problem is that the rover stopped and then started again, it didn't slow down. So now imagine that we stop the motor every half a second. On, off, on, off, on, off. It still has only traveled for two out of three seconds, the same distance but a little less jerky.

Now imagine we turn the motor on and off every tenth of a second. This is getting quite fast and the off is blending into the on. The rover will still only move for two seconds of time in three except now the times are much shorter between the off and on. The MicroBit can actually do this even faster, so fast you cannot see it stop at all. The name for this is Pulse Width Modulation or PWM. The pulse is the off and on. The width is the time off to on. The modulation is the changing of it all. A long explanation but we hope you understand.

So here is the code that controls the speed. Instead of:

```
pin2.write_digital(1) # The 1 is on remember
pin13.write_digital(1)
```

```
This would make one motor faster than the other.
```

```
pin2.write_analog(700) # The bracket number is between 0 ~ 1023
pin13.write_analog(1300) # The motor needs more than 1 to move though
```

By changing the values in the brackets it is possible to change the motor speed. And by making one slower than the other to control the straight line. It could also be used to control the speed of turns. The motors will not start to turn on very low numbers. You will have to experiment.