DT Mini Challenge

Networking with micro:bit

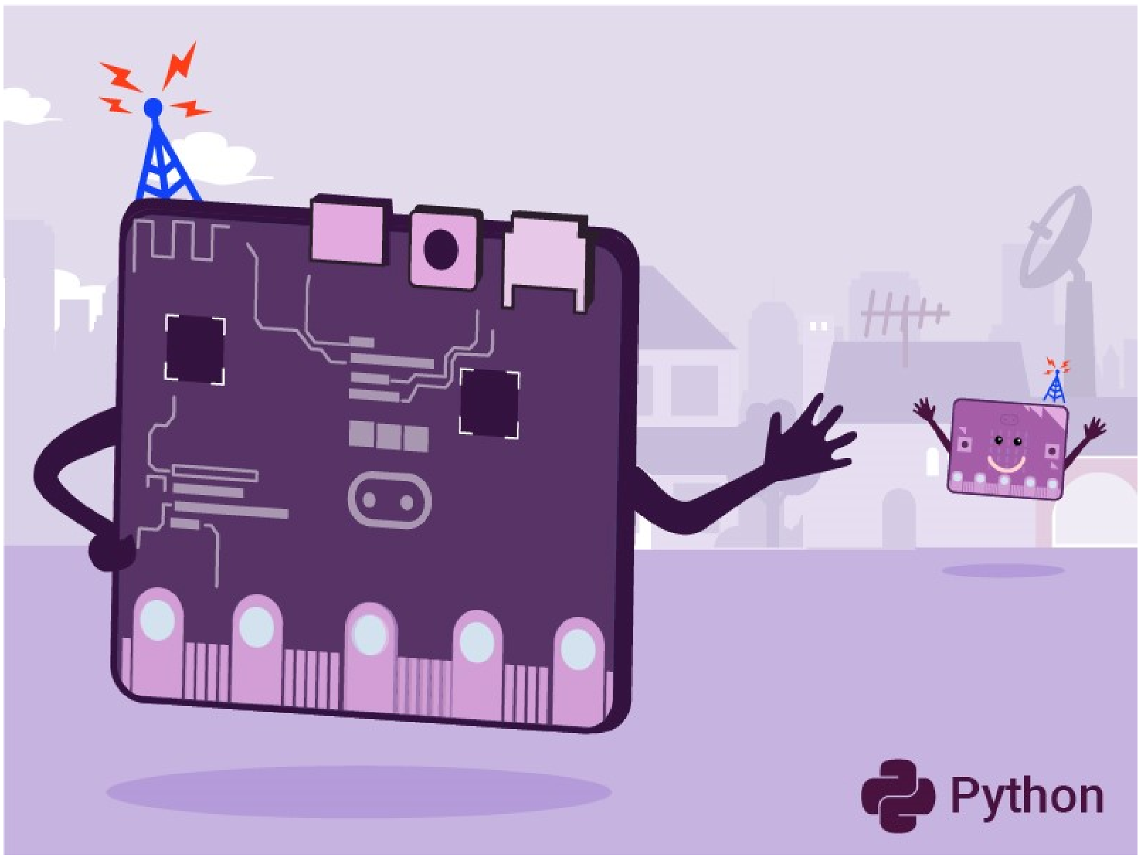
1

RADIO SEND AND RECEIVE

1.1. Welcome

1.1.1. Radio communication

This course is all about using the micro:bit's radio to make micro:bits message each other!



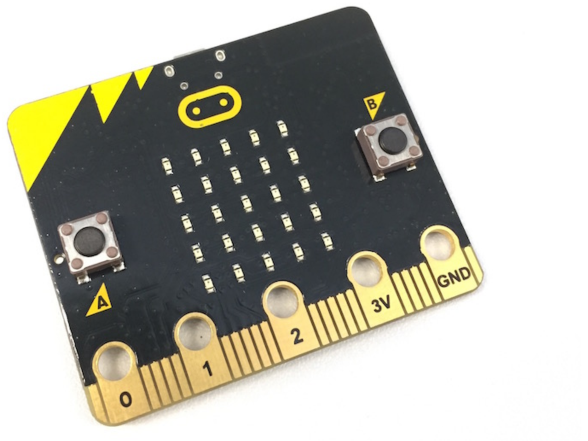
We'll be writing code for both the sender and receiver, so it's best to work in pairs.

Do you and a friend both have micro:bits?

Radio is way more fun in real life! If you have some micro:bits, loading code onto them is the most fun way to send and receive messages!

1.1.2. Quick review

Before we use the radio, we need to learn how to program the micro:bit.



The BBC micro:bit (https://www.microbit.org/) is a tiny computer that runs the Python (https://microbit-micropython.readthedocs.io) programming language.

The micro:bit has:

- **a 5 x 5 display of LEDs** (light emitting diodes)
- **two buttons** (A and B)
- **an accelerometer** (to know which way is up)
- **a magnetometer** (like a compass)
- **a temperature sensor**
- **Bluetooth Radio** (to talk to other micro:bits and phones)
- **pins** (gold pads along the bottom) to connect to other devices like screens, motors, buttons, lights, robots and more!

If you don't have a real micro:bit...

You can still do this course. It includes a full micro:bit simulator, so you'll be able to do almost everything you'd do on a real micro:bit!

1.1.3. Scrolling messages

First, let's learn how to display messages.

We can scroll our own message on the micro:bit display using the `display.scroll` *method* (like a command).

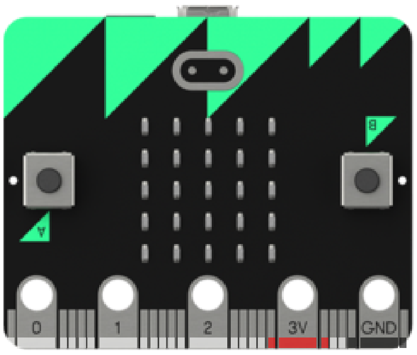
`display.scroll` takes a *string* as an argument, which is text between quotation marks (`' '`). This program scrolls `Hello, world!` but you can change it to write anything you want.

Press the ▶ button to run your code on the virtual micro:bit at the bottom of the screen. Try changing the message!

```
from microbit import *

display.scroll('Hello, world!')
```

Press start

Press the ▶ button to start running the program.



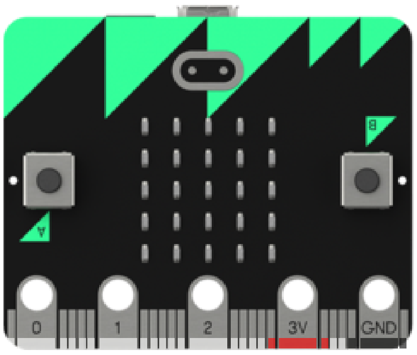
1.1.4. What happens when things go wrong

When you talk, you need to follow certain rules to be understood, called the *grammar* or *syntax* of a language.

Like English, Python has its own *syntax*. However, unlike people, computers can't understand bad grammar at all!

Press ▶ to run the following example using write instead of display.scroll and see what happens. **Then click ■ to stop it running**:

```
from microbit import *

write('Hello, world!')
```



The BBC micro:bit scrolls the error on the display, which is hard to read. We print the error message in another box for you.

Don't panic!

Errors happen all the time, but don't worry, you can learn to fix them. **Try to read and understand the error message**. This message tells us that the problem is on line 3, and has something to do with 'write'.

Errors on a real micro:bit

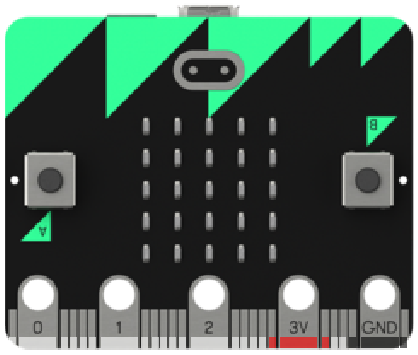
To see the printed error message for a real micro:bit, you'll need to use the serial console over USB (https://support.microbit.org/support/solutions/articles/19000022103-outputing-serial-data-from-the-micro-bit-to-a-computer).

1.1.5. Help! I have a Syntax Error

A `SyntaxError` just means that you haven't followed the grammar or *syntax* of the programming language, so the computer can't understand you!

Let's practise fixing a `SyntaxError` together. Run this program:

```
from microbit import *

display.scroll('micro:bit'
```



Where's the error?

The error message tries to help by printing which line it thinks the error is on. In this example, it says `line 4`.

If it doesn't seem like there's an error where the error message says, **try looking on the previous line**.

Hover me for answer

1.1.6. Loop-the-loop

So far we've made very short programs.

In our first program, the micro:bit said `'Hello, world!'` once, but then it stopped! What if we want the program to keep running forever?

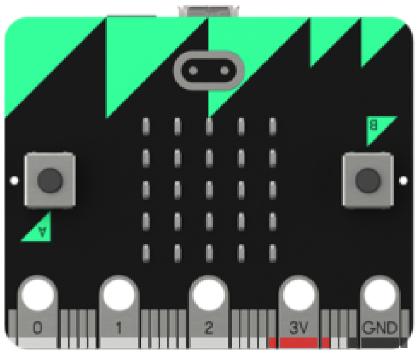
We can use a `while` *loop* to run some code again (and again)!

For example, this program will keep running until you stop it:

```
from microbit import *

while True:
  display.scroll('Hello!')
```

You have to click stop!

Don't forget to use the ■ button to stop the program, otherwise it will keep running forever.



1.1.7. Sleep

That program repeats itself very quickly!

We can use a sleep function `sleep()` inside the *loop* to make the program pause between messages.

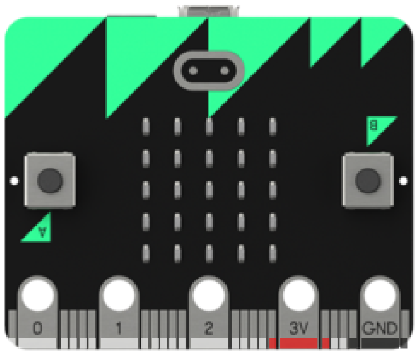
For example, this program will pause for 2 seconds between saying `'Hi!'`:

Try changing the number in `sleep()` to make it pause for longer or shorter.

```
from microbit import *

while True:
  display.scroll('Hi!')
  sleep(2000)
```

You have to click stop!

Don't forget to use the ■ button to stop the program, otherwise it will keep running forever.



1.1.8. Downloading

If you have a micro:bit you can see your program in real life!

1. Click the Download button. You will get a `.hex` file.
2. Plug your micro:bit into your computer using the USB cable.
3. Your micro:bit will show up in your list of files in your directory
4. Drag the `.hex` from the downloads folder onto the micro:bit.
5. Watch the yellow light on the micro:bit flash for a few seconds.
6. See your program on the micro:bit!

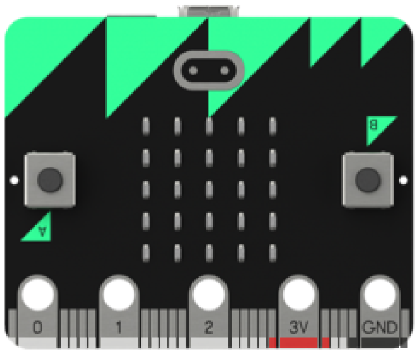
We have more detailed instructions with pictures (https://medium.com/p/b89fbbac2552) on our blog.

1.1.9. Congratulations

Congratulations! You know the basics of micro:bit.

We learned about:

- scrolling `strings` on the micro:bit
- repeating instructions with `while` loops
- making the micro:bit wait with the `sleep()` function



Now lets learn how to use the radio.

1.2. Radio send

1.2.1. How radio works

Radio waves transmit music, conversations, pictures and data invisibly through the air, often over very large distances.

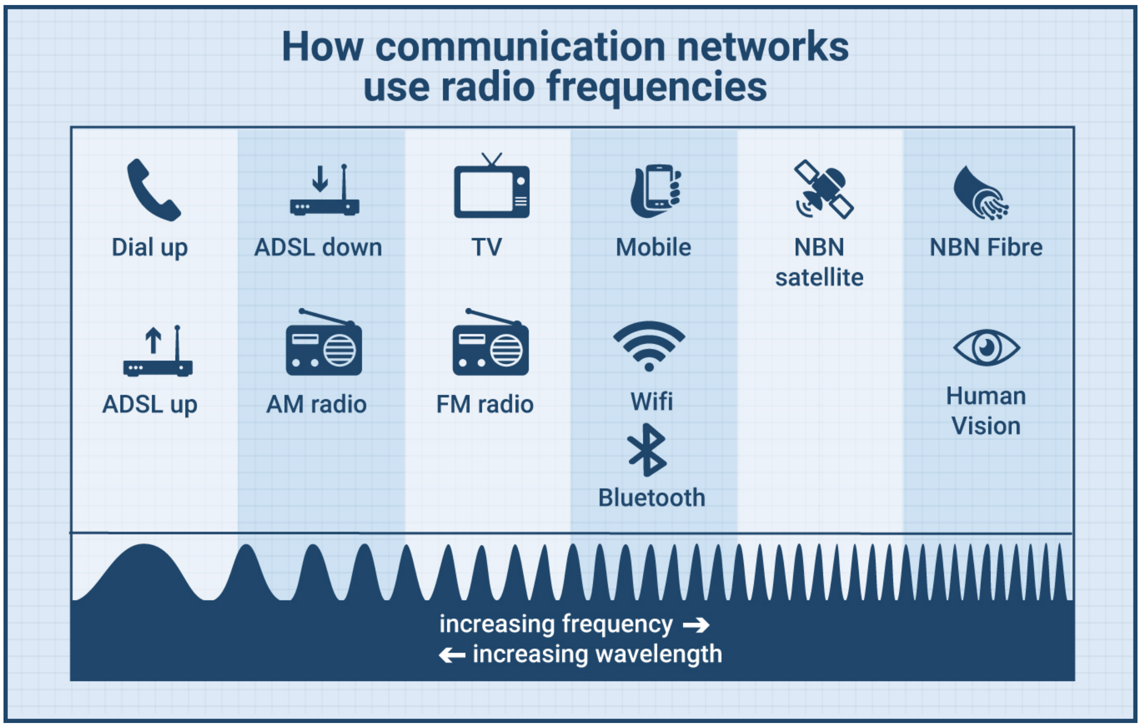
You're surrounded by technology that uses radio waves: your mobile phone, Wi-Fi networks, television and radio broadcasts, GPS, and even microwave ovens!

Radio communication needs two devices:

- the *transmitter* takes a message, encodes it, and sends it over radio waves
- the *receiver* receives the radio waves and decodes the message

A device, like a mobile phone, that does both is called a *transceiver*.

And that's how we send messages through the air, without wires!



We use lots of different frequencies to relay information

1.2.2. Turn on the radio

The BBC micro:bit has a built-in radio that can both transmit and receive messages with other micro:bits.

To use it, first we need to import the `radio` module:

```
import radio
```

Next, we need to turn the radio on:

```
import radio

radio.on()
```

The radio is off by default because it uses power and memory that you might need for other things.

Your micro:bit has a range of up to ~30 metres.

1.2.3. Sending a message

Now instead of scrolling the message across our micro:bit, we can send it to a different microbit!

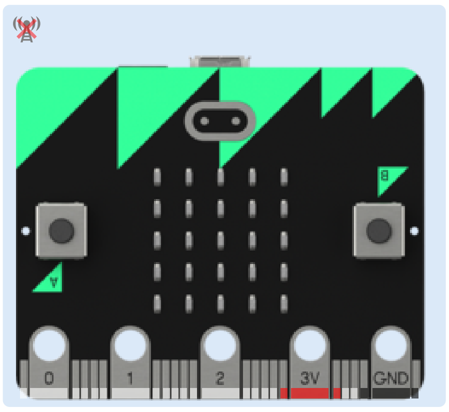
We're going to use `radio.send` to transmit messages.

`radio.send` takes a string, encodes it, and sends over radio waves:

```
from microbit import *
import radio
radio.on()

radio.send('Open sesame')
```

Radio Simulator

The simulator will show the sent message `Open sesame` above the micro:bit whenever the program is run. Click the ▶ button to test it!



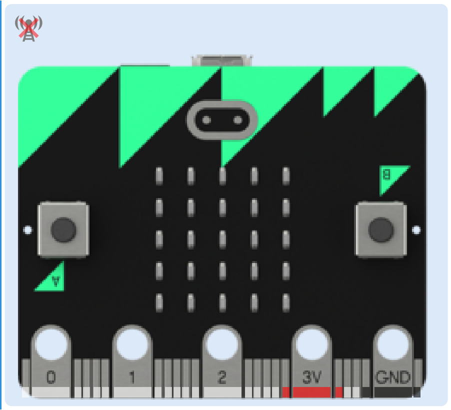
Now we've sent our message out on radio waves.

1.2.4. Problem: Power on!

You have a device, but unfortunately you've lost the remote control, so you can't turn it on. Fortunately, you know that the device can be turned on by sending a message over the radio: `'power-on'`

Write your own remote control on the micro:bit, that sends the message `'power-on'`

Here is an example of a working program (click the ▶ button to run it).



Turn the radio on!

Remember to turn the radio on before you send the message. This can be done with:

```
radio.on()
```

You'll need

program.py

```
from microbit import *
import radio
```

Testing

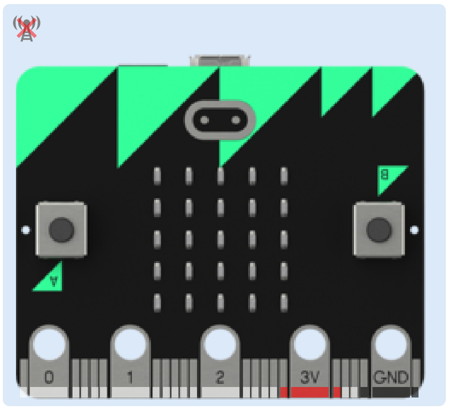
- ☐ Testing that your program sends a message.
- ☐ Testing that the program sends the correct message.
- ☐ **Awesome! You made a remote control!** 📡

1.2.5. Problem: Keep on sending

Write a program to send a message *every 5 seconds*.

The message can be anything that you like.

Here's an example sending the message: `ahoy!`, but you can send *anything*. You should start by sending the message, then wait 5 seconds.



You'll need

program.py

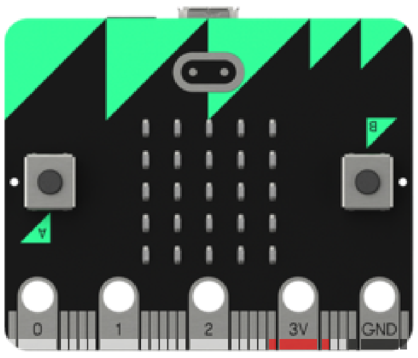
```
from microbit import *
import radio
```

Testing

- ☐ Testing there is an infinite loop.
- ☐ Testing a message is sent.
- ☐ Testing the message is sent every 5 seconds.
- ☐ Nice! 😎

1.2.6. Inputs

So far our programs have changed the micro:bit display (produced *output*). These programs have run the same way every time.

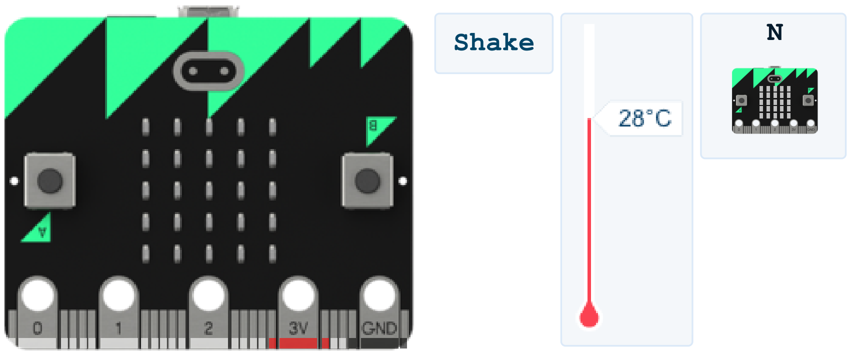


Try running the code

The micro:bit scrolls the same message every time (press the ▶ button).

1.2.7. Change the program

But we like it when a program can do more than that. Ideally a program can respond to *inputs*



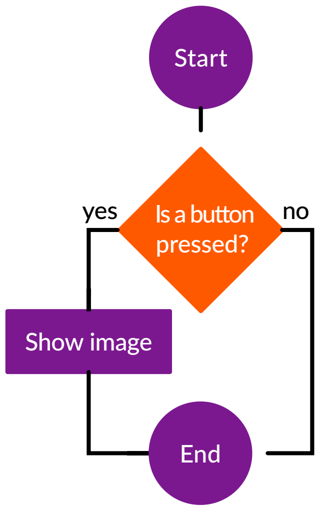
Now try running this code

The micro:bit will run a different piece of code depending on what inputs you use. Try pressing the buttons, shaking it, moving the temperature sensor, and rotating it like a compass (press the ▶ button).

1.2.8. Flowcharts

We can understand this using a flowchart.

This flowchart describes a simple process (or *algorithm*) that makes the program run differently if a button is pressed:



The diamond requires a `yes` or `no` decision. The answer determines which line we follow. If the answer is `yes`, we do the extra step of showing an image. If the answer is `no`, we skip it.

1.2.9. Send the message if ...

Now we can control when we send messages!

We're going to use `radio.send` to transmit messages, but only `if` a button was pressed.

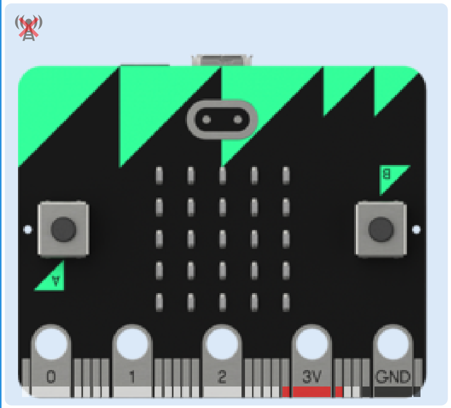
This takes a special line of code called an *if statement*. In this code we check to see if `button_a` was pressed.

If it's true that the button was pressed, then we follow the next line of indented code. If it's false, the button wasn't pressed, then we skip it.

```
from microbit import *
import radio
radio.on()

while True:
  if button_a.was_pressed():
    radio.send('Open sesame')
```

Press the button!

The simulator will show the sent message `Open sesame` above the micro:bit only after you press the button! Remember, you still have to click the ▶ button first



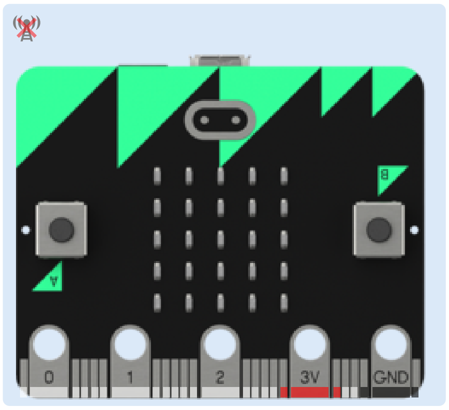
Remember the :

If you don't put a colon (:) after an if statement your code won't work

1.2.10. Multiple button presses

To send multiple button presses, we can just add another `if` statement. Here's an example:

```
from microbit import *
import radio
radio.on()

while True:
  if button_a.was_pressed():
    radio.send('duck')
  if button_b.was_pressed():
    radio.send('goose')
```

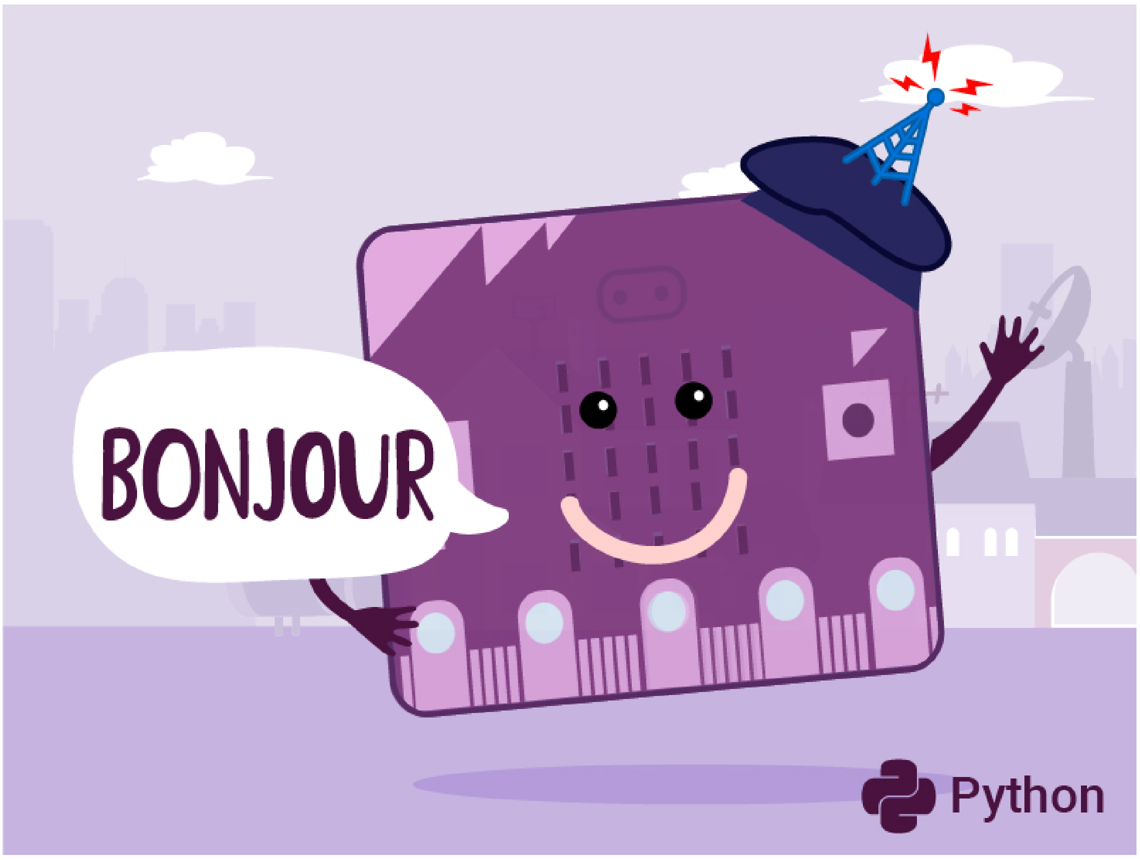


1.2.11. Problem: Hello, bonjour!

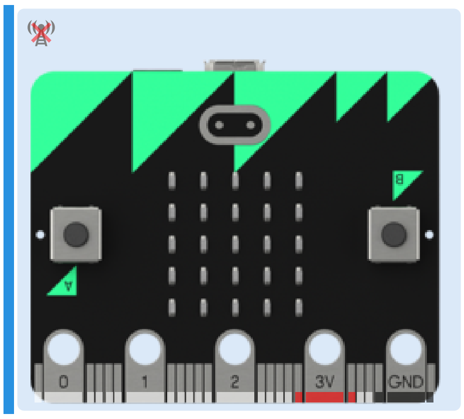
There are some French exchange students coming to visit you! Build a program to say hello in French for them!

Write a program to send some messages when the buttons have been pressed.

- `button_a` should send `'hello'`
- `button_b` should send `'bonjour'`



Your complete program should work like this:



You'll need

program.py

```
from microbit import *
import radio
```

Testing

- ☐ Testing that your code contains an infinite loop.
- ☐ Testing that nothing happens initially.
- ☐ Testing button a.
- ☐ Testing the message on button a is correct.
- ☐ Testing button b.
- ☐ Testing the message on button b is correct.
- ☐ Woo! Oui Oui! 🇫🇷

1.3. Radio receive

1.3.1. Variables

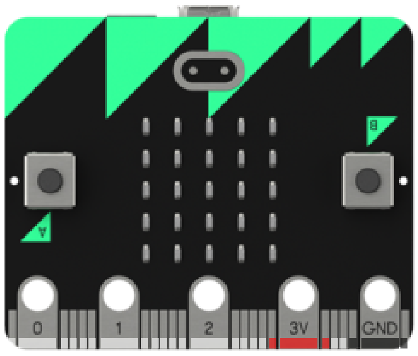
Just like how we use `radio.send()` to send a string over radio we can use `radio.receive()` to get a string back.

The incoming message gets put in a variable called message. A variable is a place for storing data.

Each variable has a name (which we decide) and a value (which we put in it).

We create a new variable and give it a value using an equals sign:

```
from microbit import*

message = 'sup'

while True:
  display.scroll(message)
```



When we give `display.scroll()` a variable our program prints its value!

Hint!

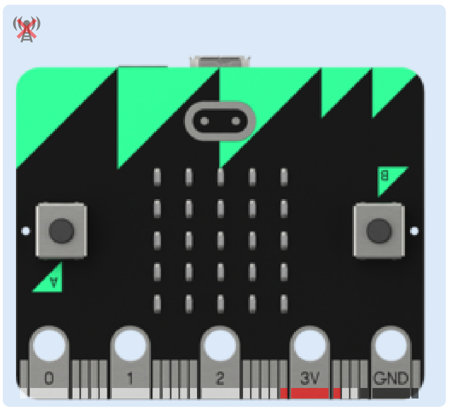
Try changing the value of the variable and see what happens!

1.3.2. Radio send using a variable

We can use variables to store radio messages before we send them.

In this case we store two strings as variables, `'antici'` and `'pation'`to send later.

```
from microbit import *
import radio
radio.on()

part1 = 'antici'
part2 = 'pation'

while True:
  radio.send(part1)
  sleep(3000)
  radio.send(part2)
  sleep(5000)
```



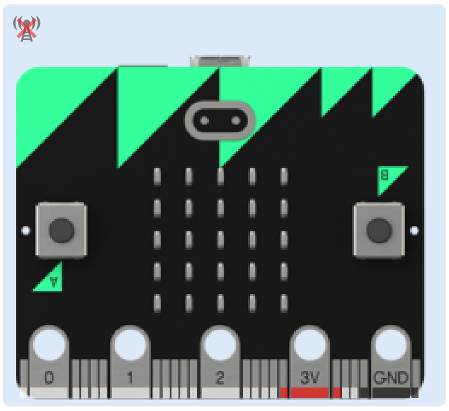
1.3.3. Undefined error!

Before we use a variable we have to make sure that we've *defined* it (given it a value). If we forget, our code will throw an error.

Error!

This piece of code has a problem with it. Press ▶ to see what happens!

```
from microbit import *
import radio
radio.on()

while True:
  radio.send(message)
  sleep(3000)
```



1.3.4. What if there isn't a message?

When we call `radio.receive()` it takes whatever `radio.send()` transmits, and stores it in a variable.

But what happens if nothing has been transmitted?

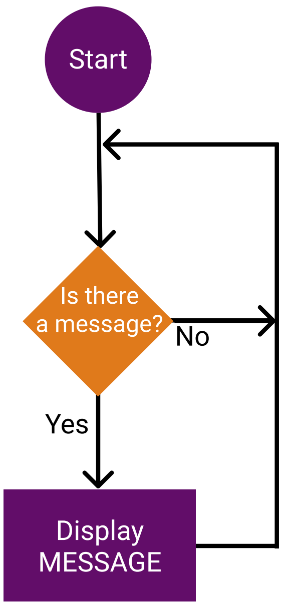
Remember, this code runs really fast. If you only send a message every 3 seconds, then the code will spends lots of time checking for a message and not finding one. Most of the time, the variable will be empty.

The problem is, if we try to use an empty variable our code will throw an error!

To deal with this we need to check `if` we've received a message before we try to use it.

1.3.5. Receive flowchart

Our flowchart will look something like this:



and if we put it in words...

1. Turn the radio on to receive messages.
2. Call `radio.receive()` to store whatever we got in a variable called `message`.
3. `if` there is a message, scroll it on the display. If there isn't a message, go back to step 2.

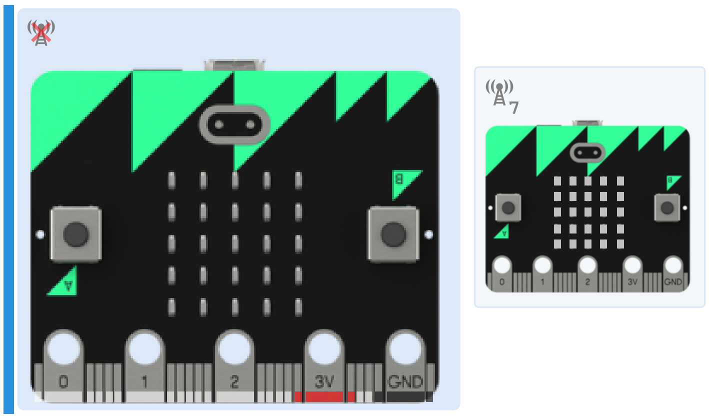
1.3.6. Receiving a message

The code should look a bit like this.

```
from microbit import *
import radio

radio.on()
while True:
  message = radio.receive()
  if message:
    display.scroll(message)
```

`if message:` is a way of checking whether there is anything in the variable. If the variable `message` contains a value it is true and the program runs through the if statement.

The small micro:bit sends `Open sesame` every 10 seconds. The program above scrolls the message on the large micro:bit:



Messages sent on radio waves can be received by more than just one micro:bit, just like more than one person can tune into the same radio station.

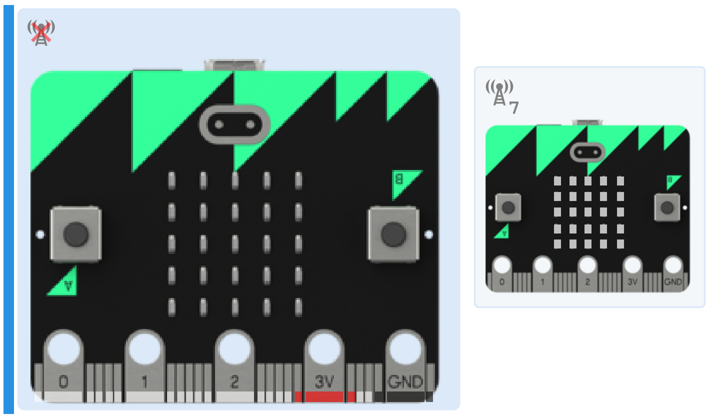
1.3.7. Problem: Received a scroll

Now let's write the program ourselves.

Your program should receive the radio message from the greeting in the previous problem and display *whatever the other micro:bit sends*.

This program shouldn't store the messsage itself. It should wait to receive a message (from the small micro:bit), and scroll the message on the display once it does.

Remember to press the ▶ button to start the example.



When you ▶ Run your code, the smaller micro:bit will transmit a message every 5 seconds like the previous problem.

You'll need

program.py

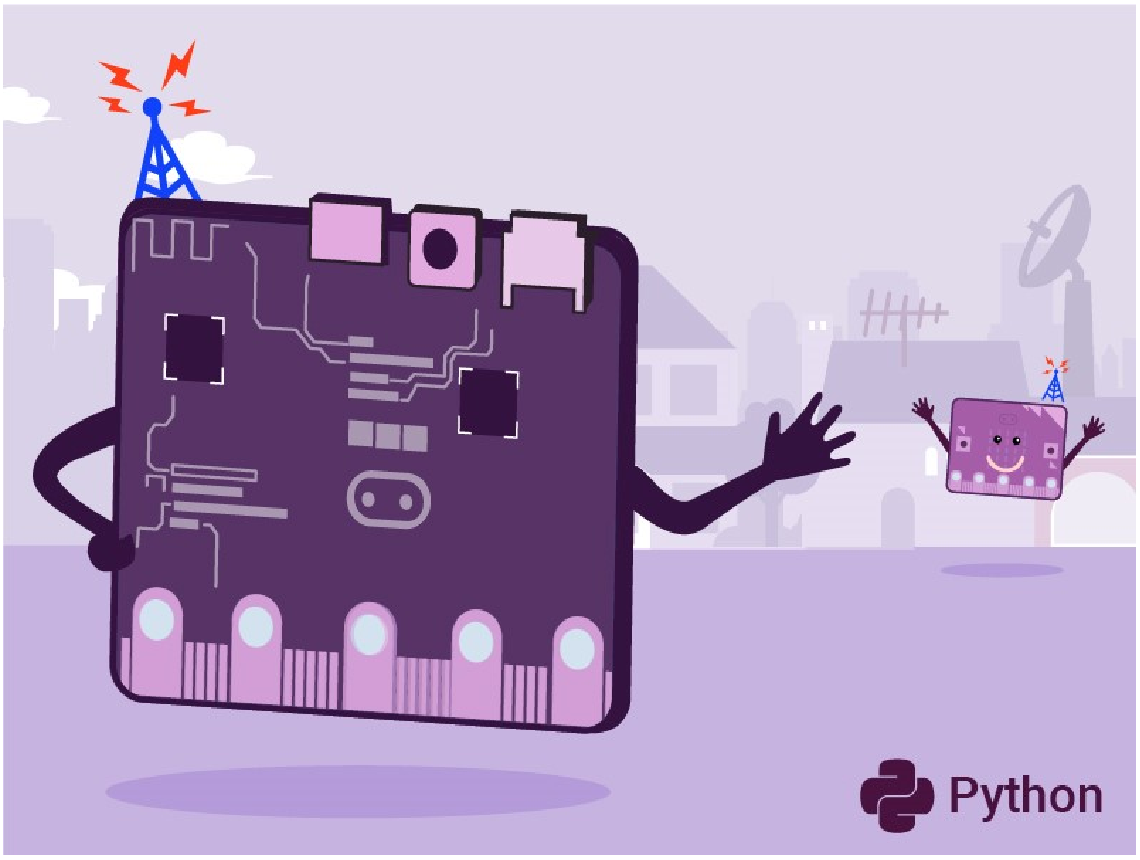
```
from microbit import *
import radio
```

Testing

- ☐ Checking that your code contains an infinite loop.
- ☐ Testing that the display is blank while there is no messages.
- ☐ Testing that the display changes when a message is received.
- ☐ Testing when `'ahoy'` is received.
- ☐ Testing that the display shows the received message.
- ☐ Testing that the display shows each new message it arrives.
- ☐ You received everything, you totally know what's going on! 👍

1.3.8. Put it together!

Now you've learned enough to both send and receive messages! Try loading the first question (sending `'hello'` and `'bonjour'` (https://groklearning.com/learn/aca-dt-mini-78-py-microbit-radio/radio/19/)) onto one micro:bit and the `receive.scroll()` code (https://groklearning.com/learn/aca-dt-mini-78-py-microbit-radio/radio/26/) onto the other!



Remember, the micro:bit has a range of about 30 metres, and can receive messages from more than one sender at a time.

If you're in a large class and you're all using micro:bits it could get noisy!

Do you and a friend have micro:bits?

Lets use them both together!

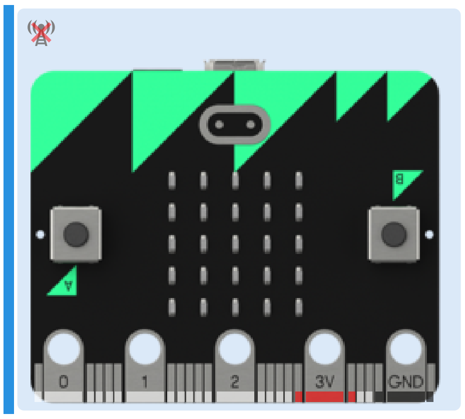
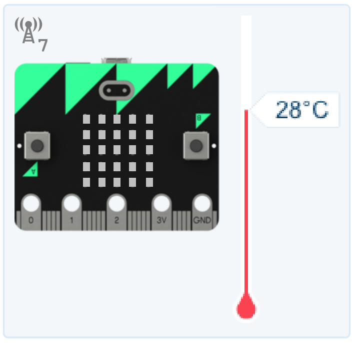
1.4. Sending numbers

1.4.1. Temperature sensor

So far we've only sent messages over the radio, now we're going to learn how to send numbers!

This allows us to send data over the radio, such as the direction of a magnetic field (compass!), how much the micro:bit is accelerating (did we drop it?) or what the temperature is.

By the end of this section we will have learned how to make a remote temperature sensor that works like this.

1.4.2. Display temperature

First let's test using `display.scroll()` to scroll a number.

When we try, the code throws an error:

```
from microbit import *

count = 7
display.scroll(count)
```

```
Traceback (most recent call last):
  File "__main__", line 4, in &lt;module&gt;
TypeError: can't convert 'int' object to str implicitly
```

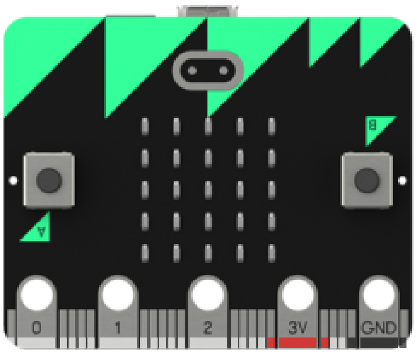
`display.scroll` complains! It can only scroll *strings* (messages), and doesn't know what to do with *integers* (whole numbers).

Python gives a `TypeError` because strings and integers are different *types* of information.

1.4.3. Numbers vs Strings

To fix this, we use the `str` function to turn an integer into a string:

```
from microbit import *

count = 7
display.scroll(str(count))
```



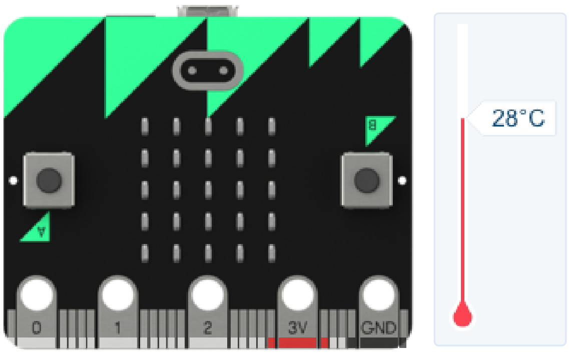
The `radio.send()` method will only accept strings too. When we make a remote sensor we will need to remember the `str` function!

1.4.4. Measuring temperature

The micro:bit has a temperature sensor on the board. We can read it (in degrees Celsius or °C) by calling the `temperature` function.

This example scrolls the temperature (after converting to a string):

```
from microbit import *

while True:
  the_temp = temperature()
  display.scroll(str(the_temp))
```



Once the program is running, *drag the arrow on the thermometer to change the simulated temperature:*

Do you have a micro:bit?

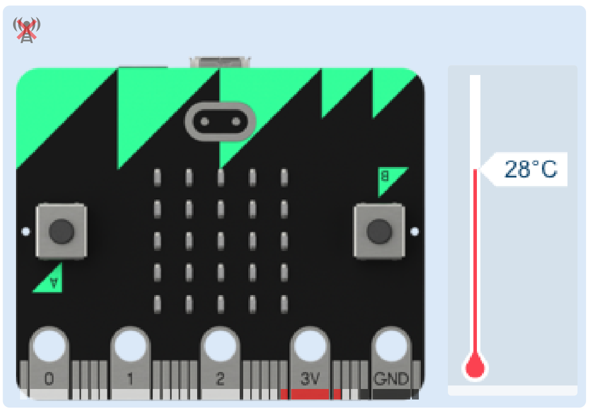
Try moving around with it and see what you can measure!

1.4.5. Problem: Temperature Sensor

Now that we can measure the temperature lets try sending it.

Write a program that measures the temperature and sends it over radio every three seconds.

Remember: `radio.send` can only send strings.



Reading the temperature

Have a look back if you forgot how to do it!

You'll need

program.py

```
from microbit import *
import radio
```

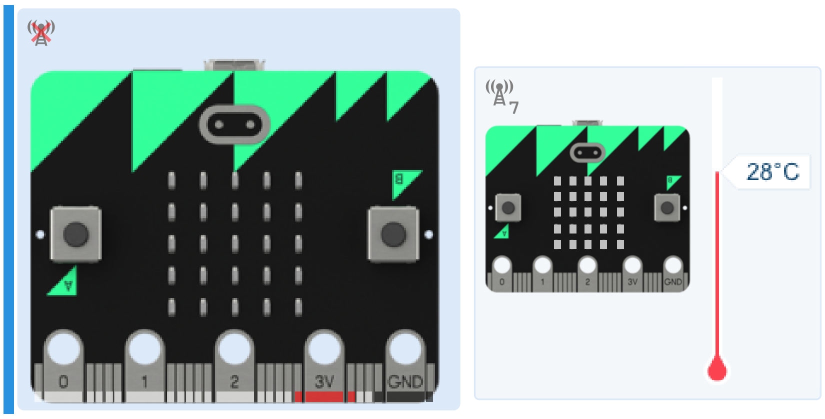
Testing

- ☐ Checking that your code contains an infinite loop.
- ☐ Testing that your program sends a message.
- ☐ Testing that your program sends a message once every 3 seconds.
- ☐ Testing that the program sends the current temperature.
- ☐ Testing that your program checks the temperature once every 3 seconds.
- ☐ Testing that the program keeps sending different temperatures.
- ☐ You made a thermometer! 👌

1.4.6. Problem: Temperature Display

Now let's write the program that receives the radio message from the temperature sensor in the previous problem and displays the temperature.

This program shouldn't read the temperature. It should wait to receive a message (from another micro:bit), and scroll the message on the display once it does.



When you ▶ Run your code, the smaller micro:bit will transmit the temperature every 3 seconds like the previous problem.

If you have real micro:bits...

Try loading the temperature sensor and display programs onto two micro:bits, and holding the sensor micro:bit near a heater, or next to an ice pack!

You'll need

program.py

```
from microbit import *
import radio
```

Testing

- ☐ Checking that your code contains an infinite loop.
- ☐ Testing that the display is blank while there is no messages.
- ☐ Testing that the display changes when a message is received.
- ☐ Testing that the display shows the received temperature.
- ☐ Testing that the display shows each new temperature reading as they arrive.
- ☐ It's getting hot in here! 🌡️

2

IMAGES AND STATES

2.1. Images

2.1.1. Images

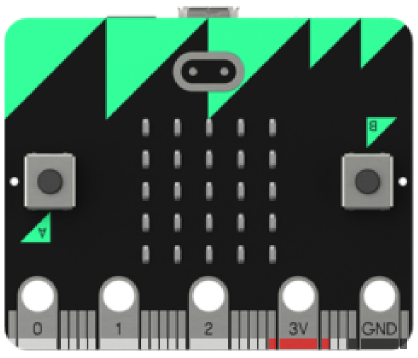
We can also use the microbit to display images.

```
from microbit import *

display.show(Image.HAPPY)
```

Everything we use in the second statement is imported from the `microbit` module:

- `display.show` is a method of the `display` object, which controls the micro:bit's display.
- `Image.HAPPY` is a built-in image provided by the `Image` class.

We *call* (run) the `display.show` method by putting brackets after it. The image we want to display goes inside the brackets:



What is a statement?

A statement is the smallest stand-alone part of a program. It tells the computer to do something. Importing the `microbit` module and calling `display.show` are both examples of statements.

2.1.2. More images

There are lots of images included in `Image` for you to use. We've already seen `Image.HAPPY`.

Here are some of our favourite images:

`Image.HAPPY`	
`Image.HEART`	
`Image.DUCK`	
`Image.PACMAN`	
`Image.ARROW_N`	

`Image.ARROW_E`	

Name	Image
`Image.SAD`	
`Image.GIRAFFE`	
`Image.BUTTERFLY`	
`Image.GHOST`	
`Image.ARROW_S`	
`Image.ARROW_W`	

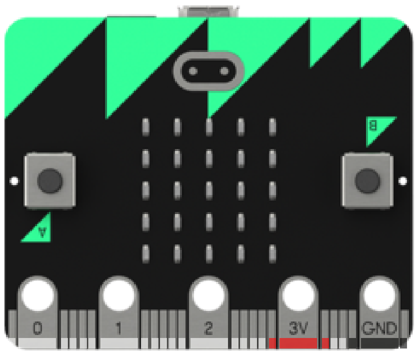
You can find the [full list here (https://microbit-micropython.readthedocs.io/en/latest/tutorials/images.html)](https://microbit-micropython.readthedocs.io/en/latest/tutorials/images.html).

2.1.3. Clearing images

We can clear the screen using the function `display.clear()`.

This code uses shows an image then pauses, clears the screen and prints a new image. Have a play around and change the code.

```
from microbit import *

while True:
  display.show(Image.HEART)
  sleep(1000)
  display.clear()
  sleep(1000)
  display.show(Image.PACMAN)
  sleep(1000)
  display.clear()
  sleep(1000)
  display.show(Image.DUCK)
  sleep(1000)
  display.clear()
```



2.1.4. Image strings

So far, we've only used the built-in images from the `Image` class:

```python
from microbit import *

display.show(Image.HAPPY)
```

Images can also described by *strings*. Here's a do it yourself `HAPPY`:

```python
from microbit import *

SMILE = Image('00000:'
              '09090:'
              '00000:'
              '90009:'
              '09990:')
display.show(SMILE)
```

`SMILE = Image()` creates an image and stores it in the constant variable `SMILE`. It can then be used like any builtin image.

The strings represent the brightness of each pixel in the image:

- Each line is a row of the image and ends with a colon (`:`).
- Each number is the pixel brightness from `0` (off) to `9` (fully on).

If we create an image without passing a string, e.g. `SMILE = Image()`, it creates a blank image, with all off the pixels off.

2.1.5. DIY images

We can now create our own images (with varying brightness):

```python
from microbit import *

FLAG = Image('40904:'
             '04940:'
             '99999:'
             '04940:'
             '40904:')
display.show(FLAG)
```

Try making your own image!

We split the 5x5 image over five lines to make it easier to read, but you can combine them into one string. The colon separates each row, so you can also leave the last one off:

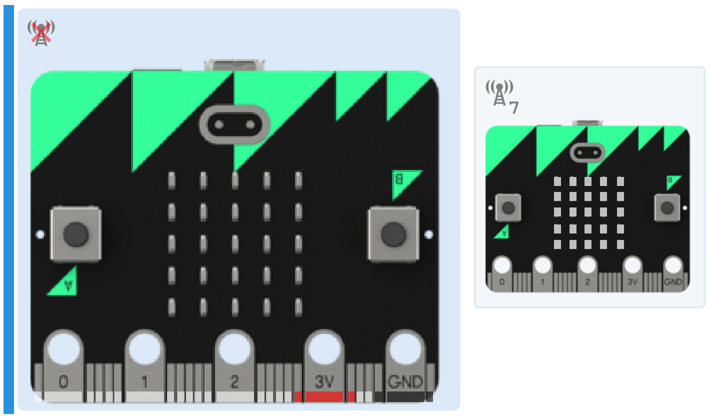
```python
from microbit import *

SOUTHERN_CROSS = Image('09009:00000:00000:00509:90000')
display.show(SOUTHERN_CROSS)
```

2.1.6. Sending images

To send an image, we can just send the image string.

```
from microbit import *
import radio
radio.on()
scissors = '99009:99090:00900:99090:99009'
paper = '00900:95500:03540:00559:00900'
rock = '09990:99999:99999:99999:09990'

while True:
  if button_a.was_pressed():
    radio.send(scissors)
  if button_b.was_pressed():
    radio.send(rock)
```

Test this code by pressing the buttons on the large micro:bit.



The receiving micro:bit will need to call the `Image` function over the received string before displaying.

2.1.7. Problem: Send an image!

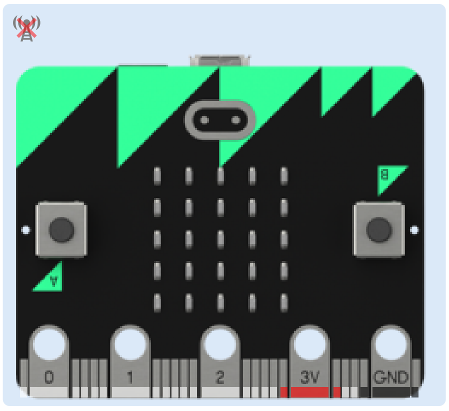
Write a program that creates an image string, then sends it over radio every 5 seconds.

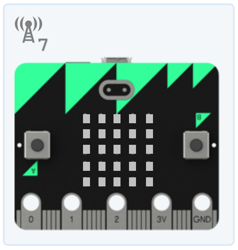
An image string has the format of 5 lots of 5 numbers separated with a colon (:). Each number is the brightness of that pixel, and must be 0 to 9 (where 0 is off).

For example, the wifi symbol has the following `Image` string:

```
'99900:00090:99009:00909:90909'
```

You can make the image string anything your like! Here's an example of sending an image:





You'll need

program.py

```python
from microbit import *
import radio
```

Testing

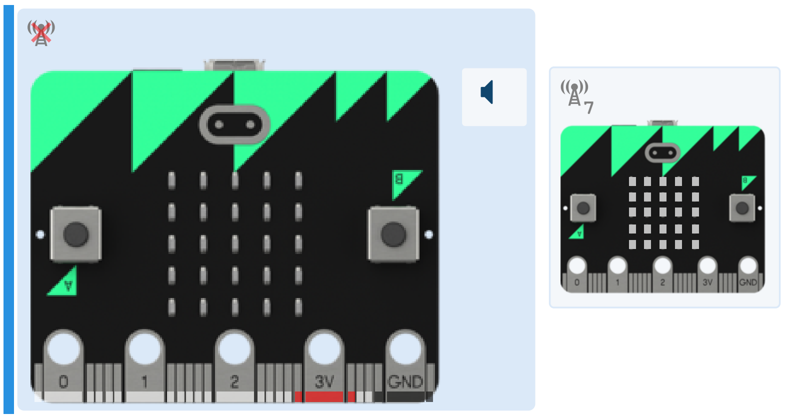
- ☐ Testing that your program contains an infinite loop.
- ☐ Testing your message has the correct number of colons.
- ☐ Testing your message has the correct format.
- ☐ Testing that your message is sent every 5 seconds.
- ☐ You sent an Image! 😊

2.1.8. Problem: Receive an image

Your friends want to send you different pictures, but it's boring know what image you'll be able to display. It's much better if they can send you *any* image!

Write a program to display the image of any *image-string* sent over the radio.

Here's an example, press button A or button B on the small micro:bit to send an image

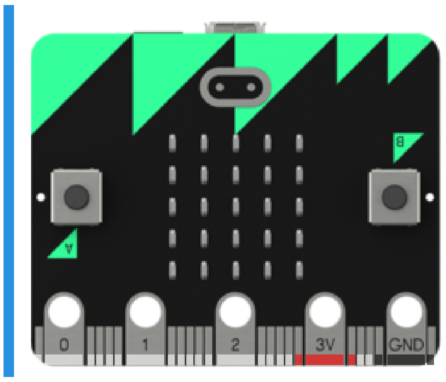


Your program should work on *any* image.

Hint!

Remember to call `Image` on the received message. For example, to draw scissors:

```python
from microbit import *
SCISSORS = '99009:99090:00900:99090:99009'
display.show(Image(SCISSORS))
```



You'll need

program.py

```python
from microbit import *
import radio
```

Testing

- ☐ Testing your program contains an infinite loop.
- ☐ Testing on a happy face.
- ☐ Testing on a sad face.
- ☐ Testing many random images.

☐ You can receive any image! 📡

2.2. Configuring the radio

2.2.1. Radio channels

So far, our programs will receive any message being transmitted by nearby micro:bits.

Imagine you're in a classroom where all sorts of different messages are being sent. How do we know which ones are meant for us?

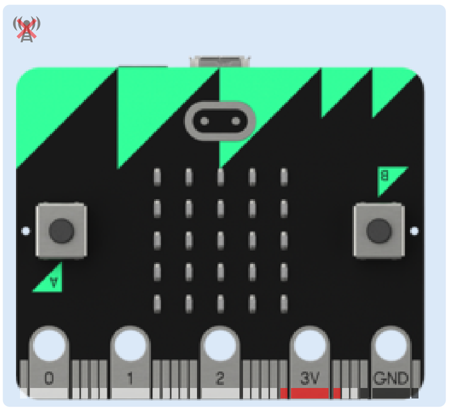
To solve this problem, we can send and receive messages on different *channels*. Different channels send messages on radio waves at different frequencies.

This is also how changing the TV channel or radio station works!

2.2.2. Tuning in to a channel

We use `radio.config` to set our channel:

```python
from microbit import *
import radio

radio.on()
radio.config(channel=20)
```



This program sets the radio to channel `20` (the default channel is `7`) using the ***keyword argument*** `channel`.

Our program will send messages on this channel, and only listen to messages received on this channel, so we need to set both the receiver and transmitter to the same channel.

Channels 0 to 83 (inclusive) are available to use.

The simulated micro:bit shows the current channel next to the indicator in the top-left corner.

2.2.3. Problem: Changing Channels

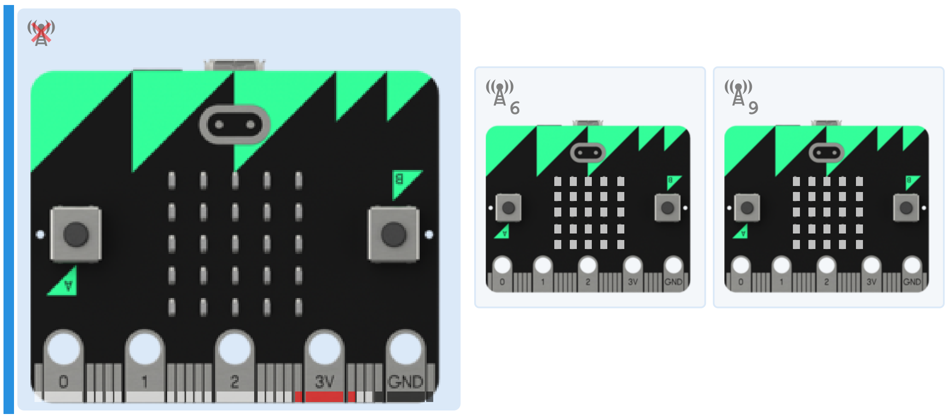
We've set up 2 simulated micro:bits that are broadcasting "TV shows" on 2 different channels:

- Channel 6 is playing a ghost story
- Channel 9 is playing a nature documentary

Each TV show is transmitting an image string every second on its radio channel.

Write a "television" remote to switch between the two the TV shows. Your program should start on channel 6. Pressing `button_a` should make it go to channel 6 and `button_b` should make it go to channel 9.

If there's something playing on the channel, your program should create images with the received strings and show them on the display.



You'll need

program.py

```
from microbit import *
import radio
```

Testing

- ☐ Checking that your code contains an infinite loop.
- ☐ Testing that the program starts off showing a picture.
- ☐ Testing that the program starts off showing the channel 6 signal.
- ☐ Testing that the program displays channel 9 (after pressing B once).
- ☐ Testing that program switches back to channel 6 (after pressing A).
- ☐ You can surf all the channels! 📻

2.3. Checking messages

2.3.1. Comparing messages

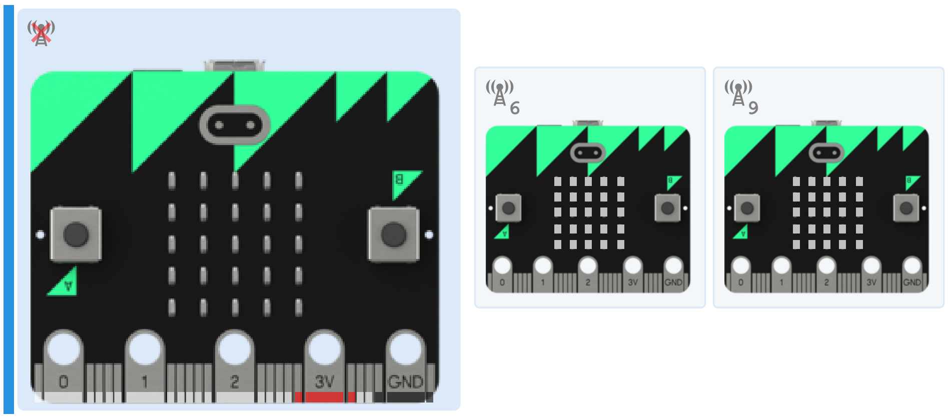
So far in our code we've just scrolled every message we get. What if we want to do more?

We can make decisions about the received message with the == operator to compare the message with another string. If the message is the same as the string, it reads `True`, but if they are different it reads `False`.

Beware!

We use = to assign a value to a variable, but we use == to compare two different things. If you mix them up your code will throw an error.

Press the `a_button` on the small micro:bit to send `'smile'` and test how the large micro:bit reacts.

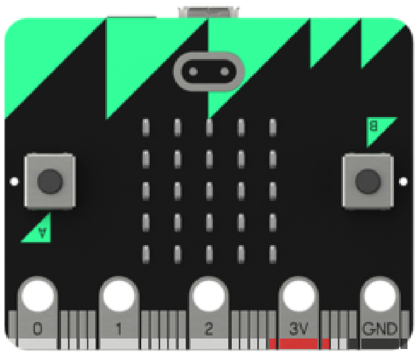


2.3.2. Check all the messages!

We can use if statements to check for multiple inputs - as many as we like!

Press `button_a` and `button_b` on the small micro:bit and see what happens.

```
from microbit import *
import radio

radio.on()
while True:
  message = radio.receive()
  if message == 'smile':
    display.show(Image.HAPPY)
  if message == 'frown':
    display.show(Image.SAD)
```



2.3.3. Too much information!

Sometimes we need to check that every input a program receives is expected before we act on it, so that the program is usable.

Remember running the Radio send and Radio receive problems? Were you in a large class of students? Imagine trying to have a personal micro:bit conversation with someone while the whole class is transmitting at once.

You would have no way of knowing where each message came from!

Too much information!

2.3.4. Security

Sometimes not checking our data can be a security problem.

Data security is important too!

Remember making a remote Temperature sensor. Could you always be sure that the temperature value that you were getting came from the sensor? If someone else in the class decided to send false information, pretending to be a temperature sensor - how could you tell?

These sorts of problems are really common in computer science. We're going to learn how to solve a few of them by filtering messages.

If you want to learn more - you should check out the [Schools Cyber Security Challenges (https://aca.edu.au/projects/cyber-challenges/)](https://aca.edu.au/projects/cyber-challenges/).

2.3.5. Filter messages

We can use the method `startswith()` to check if the string starts with a particular word or symbol.

For example, lets say Alice is daydreaming, and will only listen to somebody if they call her name first.

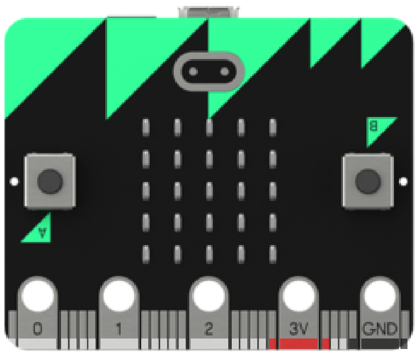
She can use the following code to filter out all messages that don't start with `'alice:'`.

```
from microbit import *
import radio

radio.on()
while True:
  message = radio.receive()

  if message and message.startswith('alice:'):
    display.scroll('yes?')
```

Watch out!

We need to use `if .. and` to first check whether there is a message at all, and then check whether it starts with `'alice:'`. If we try to call the `startswith()` method on an empty variable it will throw an error.

Try it out! Pressing the `a_button` on the small micro:bit will send a message that starts with `alice:`, and pressing the `b_button` will not.



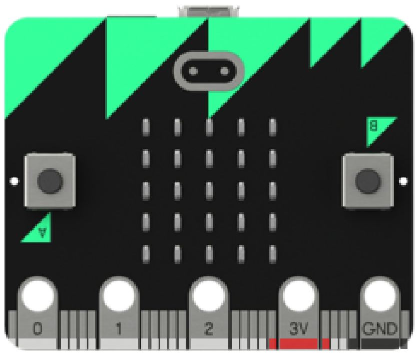
2.3.6. More filtering

Sometimes we want to react to messages that don't start with a particular string.

In that case we can use the command: `and not`

```
from microbit import *
import radio

radio.on()
while True:
  message = radio.receive()
  if message and not message.startswith('alice:'):
    display.show(Image.NO)
    sleep(1000)
    display.clear()
```

Try it out! Pressing `button_a` on the small micro:bit will send a message that starts with `alice`, and pressing `button_b` will not.



2.3.7. Problem: Wake Up Jeff

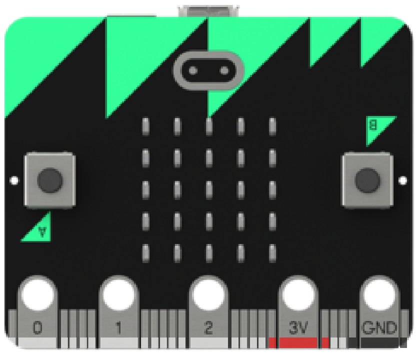
Your microbit, called Jeff, is asleep in class. If the microbit receives a message, it should scroll `zzz` on the screen unless the message starts with `jeff:`.

Once it hears its name, the microbit will wake up, transmit `what?` over the radio and display a question mark for three seconds. We've given you a `QUESTION_MARK` image:

```
QUESTION_MARK = Image('09990:90009:00990:00000:00900')
```

Then the program will continue on, as if nothing has changed.

The simulation below shows how the program should work. You can simulate sending messages to the 'Jeff' microbit by pressing `button_a` and `button_b`.



You'll need

program.py

```
from microbit import *
import radio

QUESTION_MARK = Image('09990:90009:00990:00000:00900')
```

Testing

- ☐ Checking that your code contains an infinite loop.
- ☐ Testing that the display is blank while there is no messages.
- ☐ Testing that the display changes when a message is received.
- ☐ Checking that `'zzz'` is scrolled when the message does not start with `'jeff:'`.
- ☐ Check it doesn't scroll `'zzz'` first for `'jeff:'` messages
- ☐ Shows question mark when `'jeff:'` is received.
- ☐ Shows question mark for 3 seconds.
- ☐ Testing that `'what?'` is sent after receiving a message starting with `'jeff:'`.
- ☐ Testing lots of messages.
- ☐ You woke up Jeff! 🔈🔈🔈😴😴➡️😲😲

2.4. Tracking state

2.4.1. A smarter program

In the last question, we could yell at Jeff the micro:bit all we liked but it never woke up. Even though the program reacted to inputs, it couldn't remember what happened before.

It's a myth that goldfish have a 3 second memory. But even if it were true, they'd still be doing better than our last program

If we want to make a more intelligent program we can do it using boolean variables.

2.4.2. Boolean variables

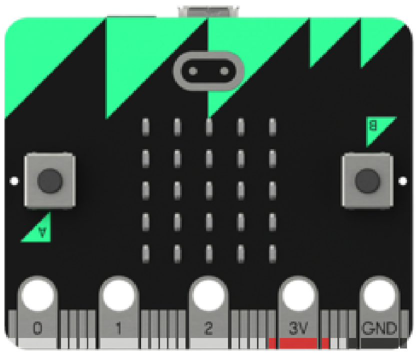
A boolean variable, is a variable that can either contain `True` or `False`. Setting these variables is really handy when checking if an event has happened or not.

For example:

```
from microbit import *
import radio
radio.on()
sent = False

while True:
  if button_a.was_pressed() and sent == False:
    radio.send('hello')
    sent = True
  if button_b.was_pressed():
    sent = False
```

This code lets you press a to send hello, but only once.

If you want to send hello again, you need to reset it using `button_b`.



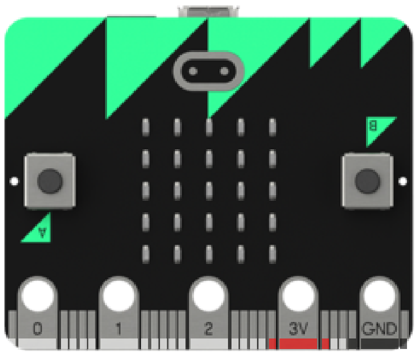
The `sent` variable keeps track if a message has been sent or not, so the program will only ever send a message *once*.

2.4.3. Problem: Matter of state

Write a program to that sends a message when `button_a` has been pressed. But if `button_a` has been pressed again, no message is sent.

`button_b` should reset the state, so a message can be sent again.

Your program should behave like this, where the message sent is `'hi'`, but you can send any message you like!



You'll need

program.py

```
from microbit import *
import radio

radio.on()
sent = False
```

Testing

- ☐ Testing your program has an infinite loop.
- ☐ Testing no message is sent initially.
- ☐ Testing a message is sent after a has been pressed.
- ☐ Testing no message is sent when a is pressed again.
- ☐ Testing b clears the state.
- ☐ You're a real statesman! 👸 🤴

2.5. Project

2.5.1. Problem: Project: Say my name

Now lets put everything together! We want to make it so you'll be able to say your name on a friends screen, and they'll be able to scroll their name onto yours! 🎙️🙌📱

But scrolling the received name should only work after you're ready.

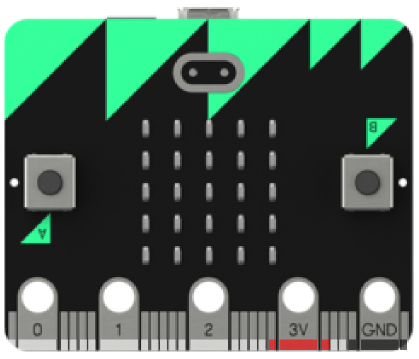
Here's how the program should work:

- All messages should be sent on channel 12
- If you press `button_a` it should send your message
- If you press `button_b` it should be ready to receive a message and display a question mark. ❓
- If you receive a message and you have displayed a question mark, your should scroll the message that has arrived, and not display any more messages.

The question mark code is provided for you.

```
QUESTION_MARK = Image('09990:90009:00990:00000:00900')
```

When you ▶ Run your code, the both micro:bits will run the same code sending different messages. Press `button_a` to test sending messages, and `button_b` to reset the state.



You'll need

program.py

```
from microbit import *
import radio

radio.on()

QUESTION_MARK = Image('09990:90009:00990:00000:00900')
```

Testing

☐ Checking that your code contains an infinite loop.

☐ Testing that nothing is sent over the radio initially.

☐ Testing that something is sent when `button_a` was pressed.

☐ Testing nothing happens if a message is received before `button_b` is pressed.

- ☐ Testing `QUESTION_MARK` image is displayed when you press `button_b`.
- ☐ Testing the received message is scrolled after `button_a` has been pressed.
- ☐ Testing that no further messages are scrolled.
- ☐ Testing that pressing `button_b` again resets the state.
- ☐ You did it! Say my name, say my name! 🎙️🙌🏽📱 🕶️🔷 🕴️

3

EXTENSION: FUNCTIONS

3.1. Real world radio

3.1.1. Functions

Functions are repeatable blocks of code.

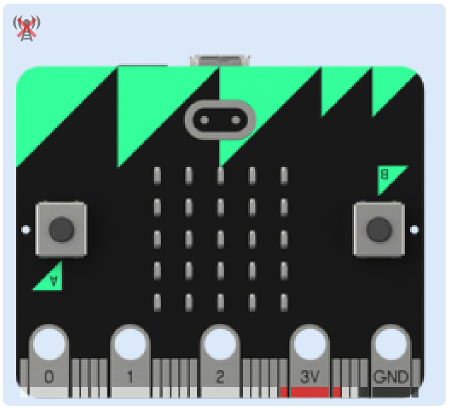
To create a function, you use the `def` keyword, and then the name of the function. Here's an example:

```
def process_image(image):
  # code here
```

It's possible to *call* a function by using the function's name, which can be anything you like. Here it's called `process_image`. You can send information into the function as an input, which in this case is a variable called `image`.

Have a look at it run!

```
from microbit import *
import radio

radio.on()

def process_image(image):
  display.show(Image(image))
  radio.send(image)
  sleep(2000)
  display.clear()

MY_IMAGE = '00900:00990:00909:99900:99900' # your image here
YOUR_IMAGE = '09990:90009:09990:09190:90009' #someone else's image

process_image(MY_IMAGE)
process_image(YOUR_IMAGE)
```



See how we called the `process_image` function *twice*, but we only had to write it *once*. You can run functions as many times as you like.

> **Hint**
> You have to define your function before you use it. Place it at the top of your code.

3.1.2. Set the range

The micro:bit radio's range depends on how much power it's given.

We can configure the radio to increase or decrease its range by using the `radio.config` method.

```
radio.config(power=2)
```

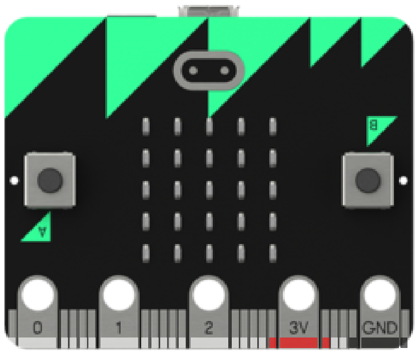
The setting can be any integer from 0 (which gives a range of ~10cm) to 7 (range ~30m).

Experiment to find the perfect power level to send a message to the people next to you, but no one else. Like whispering, it's a way of controlling who receives your messages.

You can use the following code for testing. It will send and display an image of your choice when you press that `button_a`, and display any image that you receive.

> **Hint**
> We've provided you with a new function called `quick_display`. How is it different from `process_image`?

```
from microbit import *
import radio

radio.on()
radio.config(power=2)

MY_IMAGE = '09090:99999:99999:09990:00900' # TRY CHANGING THIS IMAGE!

def quick_display(image):
 display.show(Image(image))
 sleep(700)
 display.clear()

while True:
 if button_a.was_pressed():
   radio.send(MY_IMAGE)
   quick_display(MY_IMAGE)

 msg = radio.receive()
 if msg:
   quick_display(msg)
```

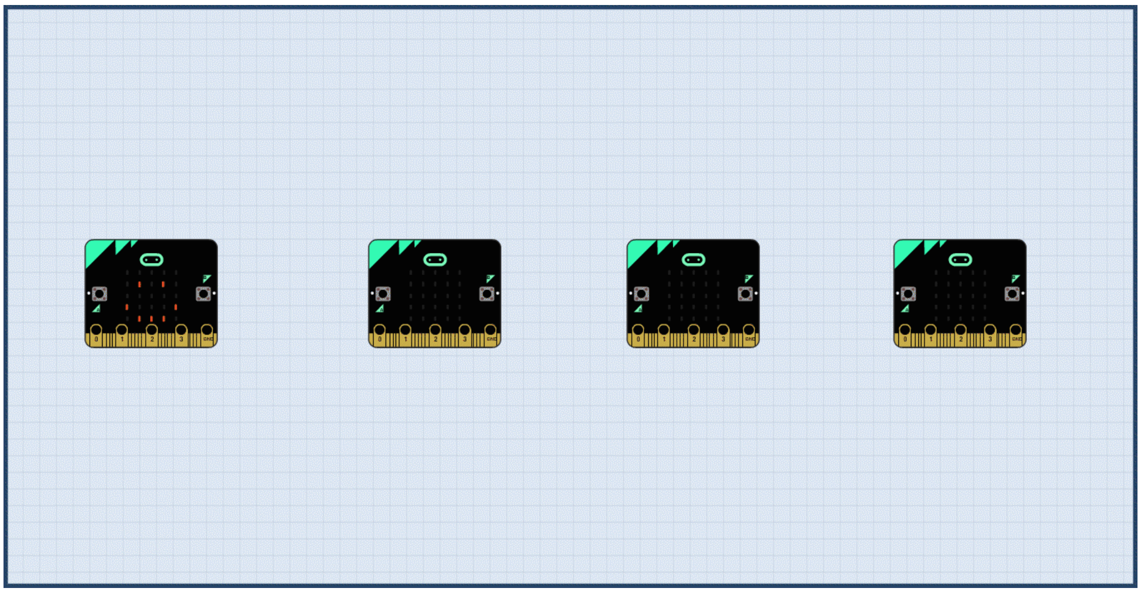


3.1.3. Mexican wave

A Mexican Wave (https://en.wikipedia.org/wiki/Mexican_Wave) is a celebration at sporting events where successive groups of spectators briefly stand, yell and raise their arms. When you see the person next to you start a wave, you copy them and then sit back down. The result is a visible wave moving through the crowd.

Mexican Wave in Frankfurt (Florian K, Wikimedia (https://commons.wikimedia.org/wiki/File:Confed-Cup_2005_-_Laolawelle.JPG), cropped)

We can make the micro:bit send a mexican wave by transmitting an image a short distance, and making nearby micro:bits copy it and pass on.


Micro:bits making a Mexican Wave.

For this to work best your micro:bit must transmit further than the closest person to you, but not as far as the next person. Experiment to find the ideal range!

Hint

Go back to the previous slide to work out the ideal range.

3.1.4. Problem: Challenge: Make some waves 🌊

Your final challenge is to make your class's micro:bits light up like a mexican wave. Use states to make sure that your micro:bit can only light up once per wave. Use the provided `process_image` function to simplify your code.

Your program should work like this:

If you press `button_a`: start the wave!

- use the `process_image` function to display your image and send it along
- change the state, so that your micro:bit will not react to any received messages

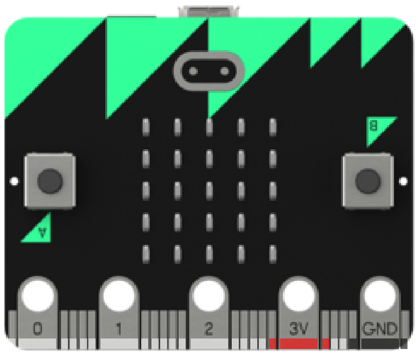
If you receive a message and you didn't start the wave: continue the wave!

- wait 500ms (micro:bits react faster than people)
- use the `process_image` function to display the image for 700ms and send it along
- change state, so that your micro:bit will not react to any received messages

If you press the `b_button`: reset the state.

Remember

Your micro:bit should only react to a received message and display the image once. Have a look at 'Matter of state' for an example.



You'll need

program.py

```
from microbit import *
import radio

radio.on()
radio.config(power=2)

MY_IMAGE = '09090:99999:99999:09990:00900' # heart
ready = True

def process_image(image):
  display.show(Image(image))
  radio.send(image)
  sleep(700)
  display.clear()

while True:
  # your code here
```

Testing

- ☐ Testing that there is an infinite loop.
- ☐ Checking radio power was changed
- ☐ Checking message is sent after `button_a` was pressed.
- ☐ Checking image is displayed for 700ms.
- ☐ Checking `process_image` is called 500ms after a message is received.
- ☐ Testing the micro:bit ignores subsequent messages
- ☐ Checking `button_b` resets the state.
- ☐ Congratulations! 👐🏾🙌🙌🏿🙌🏽👐🏻👐🏼 We're on the same wavelength. 🎉

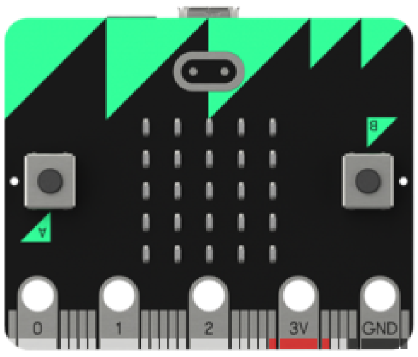
3.1.5. Problem: Radio playground

What's this, thought you were finished!?

This is a playground question where you can program a micro:bit to do whatever you want!

- Can you come up with a way to send more than two messages by pressing the buttons?
- What about sending a secret message that only the receiver can read?
- Can the other microbit sensors help?

We've set you up with a microbit with a temperature sensor, compass (https://microbit-micropython.readthedocs.io/en/v1.0.1/compass.html) and gesture sensor (https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/gestures.html).



Happy coding!

You'll need

program.py

```
from microbit import *
import radio

radio.config(channel=7)
radio.on()
```

Testing

- [ ] This is a playground question, there is no right or wrong!