DT Mini Challenge

Networking (Blockly)

1 RADIO SEND AND RECEIVE

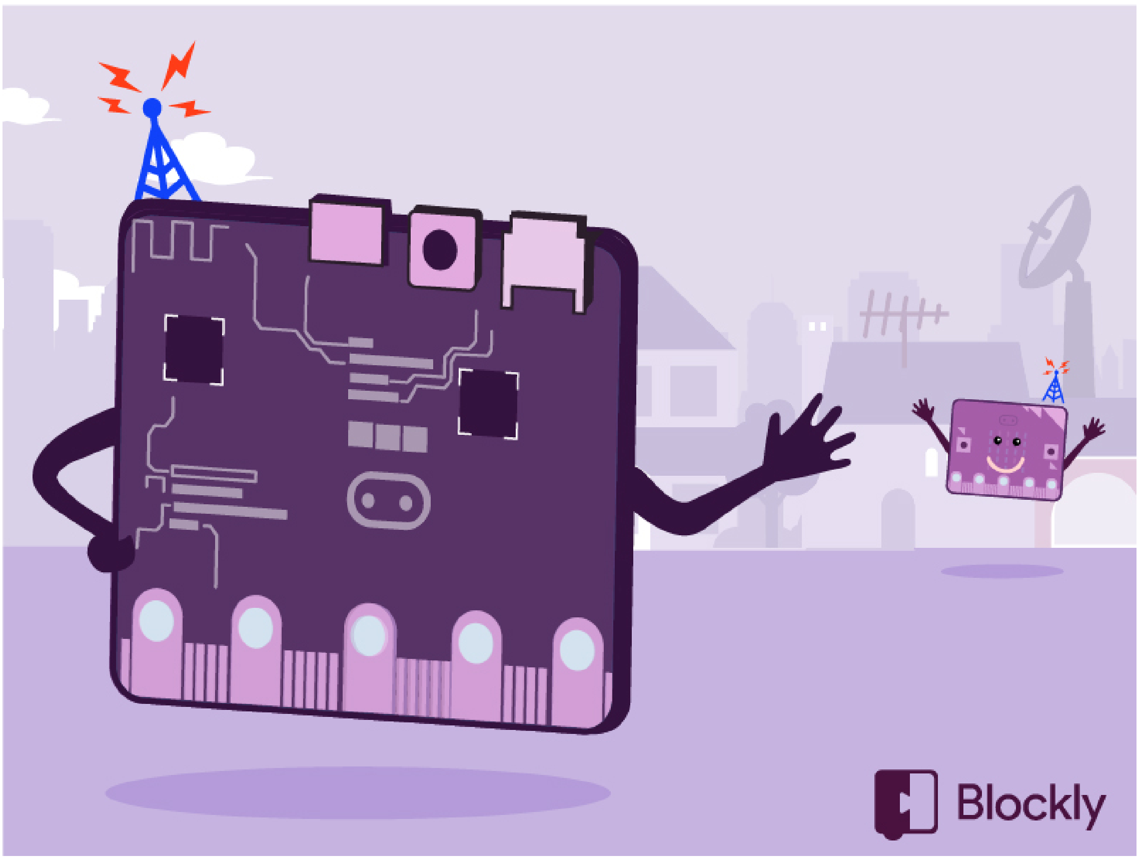
1.1. Welcome

1.1.1. Radio communication

Welcome to the Networking with micro:bit challenge!

This course is all about using the micro:bit's radio to send messages to each other.

We'll be coding using **Blockly**, a visual coding language where you drag and drop blocks of code to create a program.



Radio is all about communicating, both sending and receiving messages, so it's best to work in pairs.

Good luck! 📡

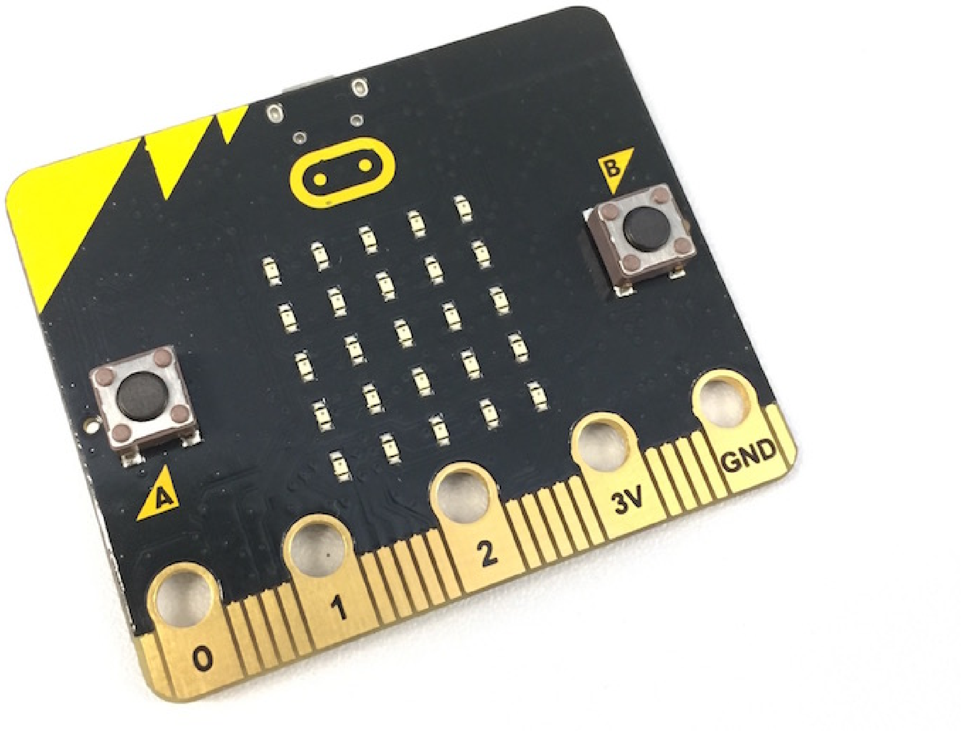
Do you and a friend both have micro:bits?

Radio is way more fun in real life! If you have some micro:bits, loading code onto them is the most interesting 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.co.uk/) is a tiny computer. You can program it with blocks.



The micro:bit has:

- **5 x 5 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**
- **a light sensor**
- **Bluetooth** (to talk to other micro:bits and phones)
- **pins** (gold pads along the bottom) to connect to robots and electronics!

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

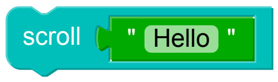
Don't worry! You can still do this course. We have a *simulator* which works like a real micro:bit.

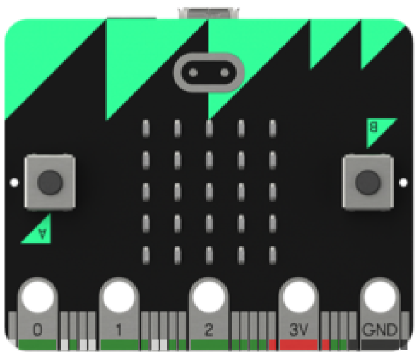
1.1.3. Scrolling messages

First, let's learn how to display messages.

We can show letters, words and sentences, using the scroll block.

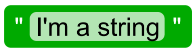
1. ▶ run the example.
2. Change " Hello " to *your* name.
3. ▶ run the example again to scroll your name across the micro:bit!





A string of letters

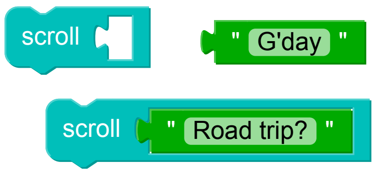
The green block is called a **string**.



1.1.4. Lots of messages!

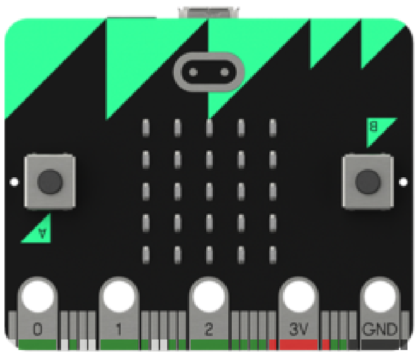
We can scroll as many messages as we like by joining blocks together!

1. Drag the text into the hole in the scroll block.
2. Join the two scroll blocks together.
3. Click ▶ to run the program.



Tip

Messing around with the code is a good way to learn. Try **changing the text** to scroll a different message!



1.1.5. Introducing loops

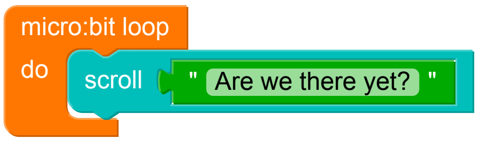
So far, our programs have only run each block once.

But we can use a micro:bit loop to repeat them!

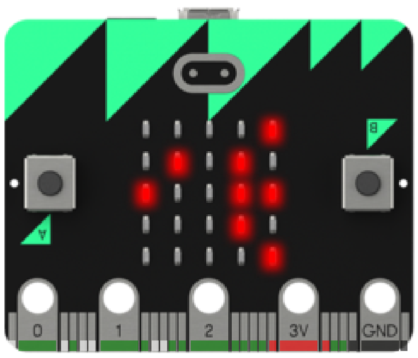
1. Click ▶ run below.

We can keep asking things forever!

2. Click the ■ button.
3. Try changing the message!
4. Run the program again!



```
micro:bit loop
do  scroll " Are we there yet? "
```



1.1.6. Sleep

That program repeats itself very quickly!

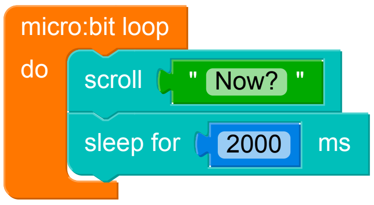
We can make the micro:bit pause by using the sleep block.

For example, this program will pause for 2 seconds between asking an annoying question.

Try changing the number in sleep to make it pause for longer or shorter. Change the message and swap the order of the blocks to see what happens.

1. Change sleep for 2000 ms to something else.
2. Change scroll " Now? " to something else

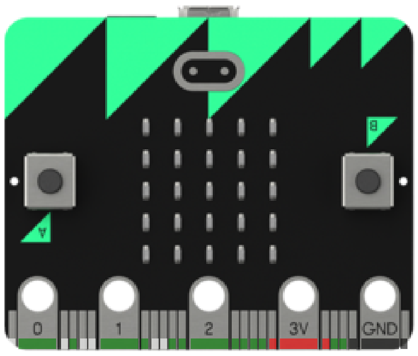
   to something else.



```
micro:bit loop
do  scroll " Now? "
    sleep for 2000 ms
```

You have to click stop!

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

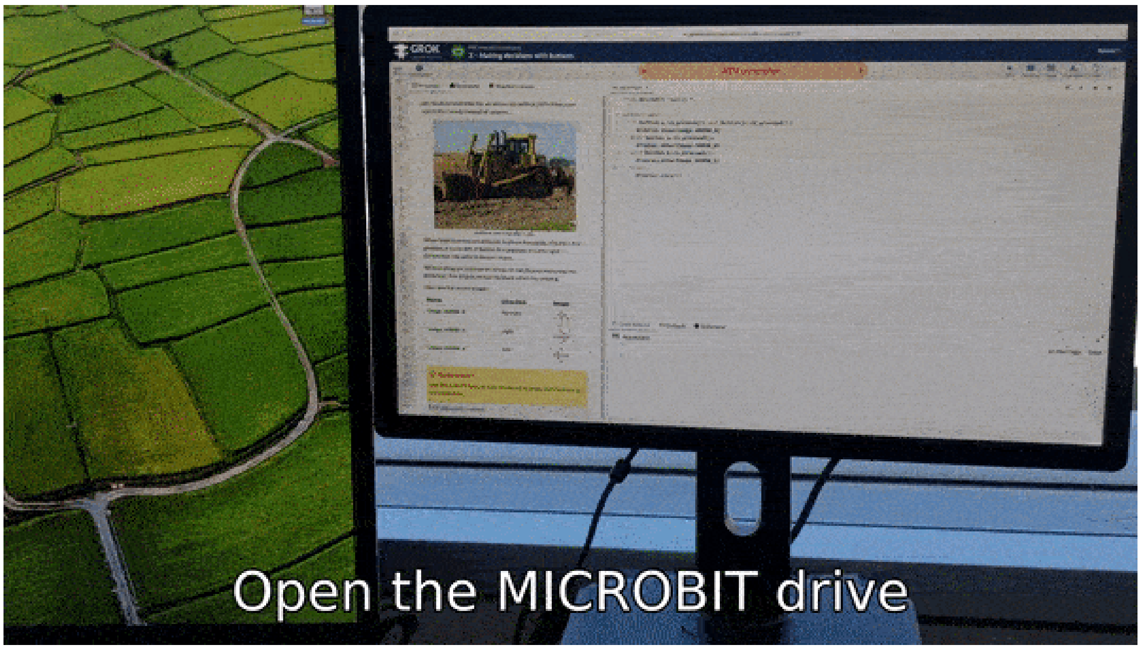


1.1.7. Downloading

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.



1.2. 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.

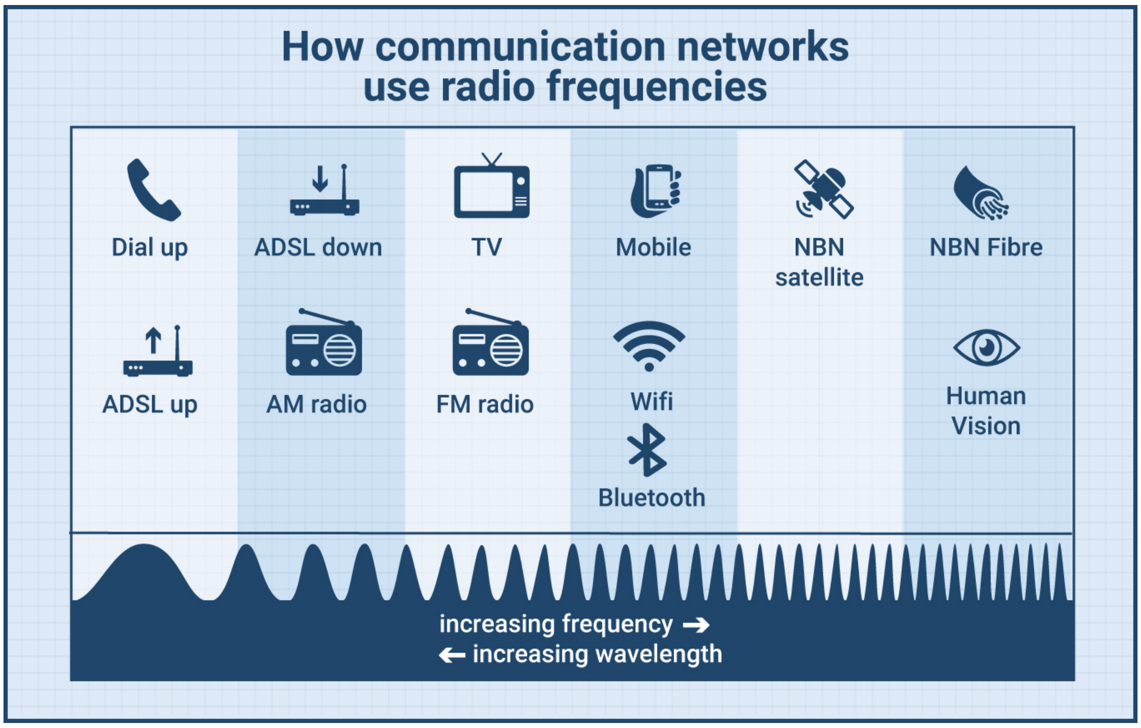
You're surrounded by technology that uses radio waves: your mobile phone, Wi-Fi networks, television and radio broadcasts, GPS, and even microwave ovens!

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!



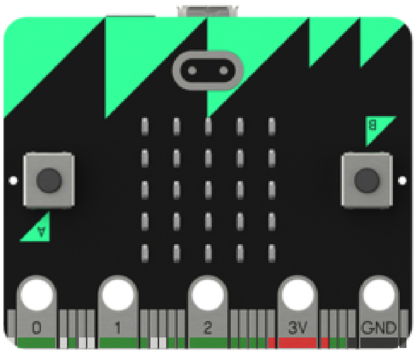
We use lots of different frequencies to relay information

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.

Before we use the radio we need to turn it on:

```
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.

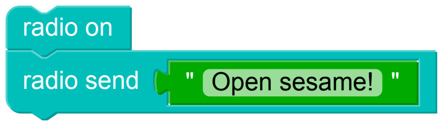
1.2.3. Sending a message

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:

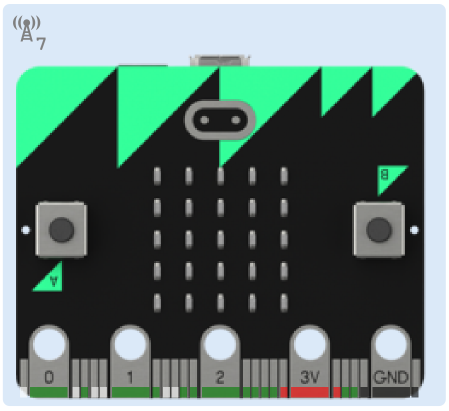
1. Run the code
2. Change the code



```
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!



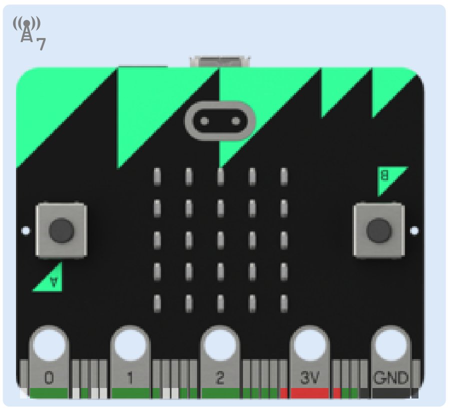
Now we've sent our message out on radio waves.

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.blockly

radio on

Testing

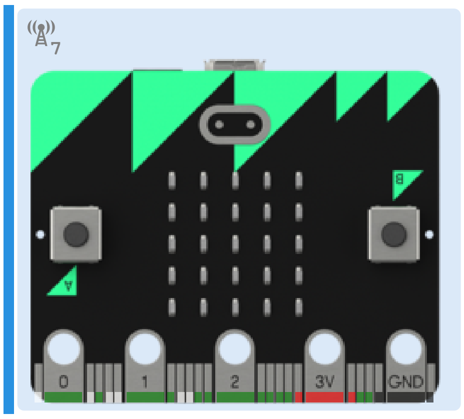
- ☐ Testing that your program sends a message.
- ☐ Testing that the program sends the correct message.
- ☐ **Awesome! You made a remote control!** 📡

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

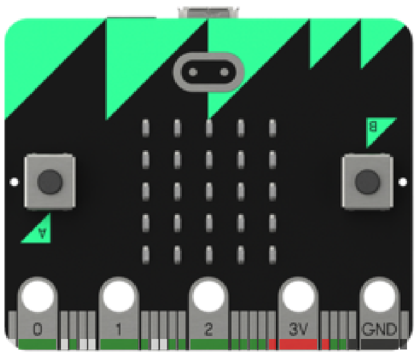
program.blockly

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

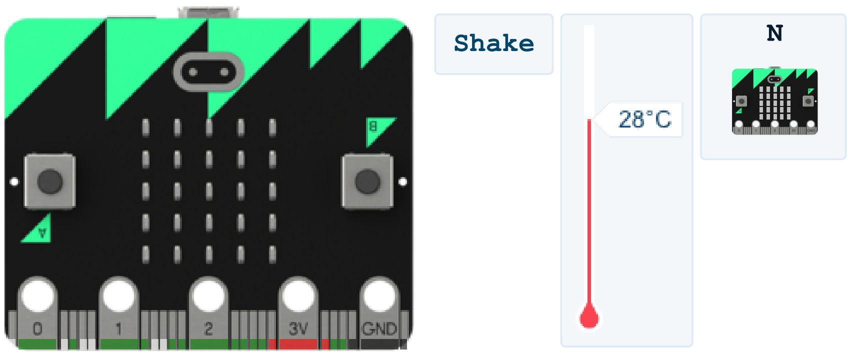
Inputs are ways for us to change the way a program runs.

This could be a button (like a lightswitch), a microphone (like a smartphone assistant).

The micro:bit has lots of inputs!

Try running this code!

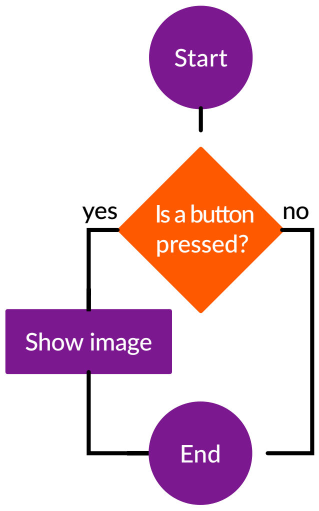
We've programmed this micro:bit to respond to lots of different inputs. Try pressing the buttons, shaking it, moving the temperature sensor, and rotating it like a compass (press the ▶ button).



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:

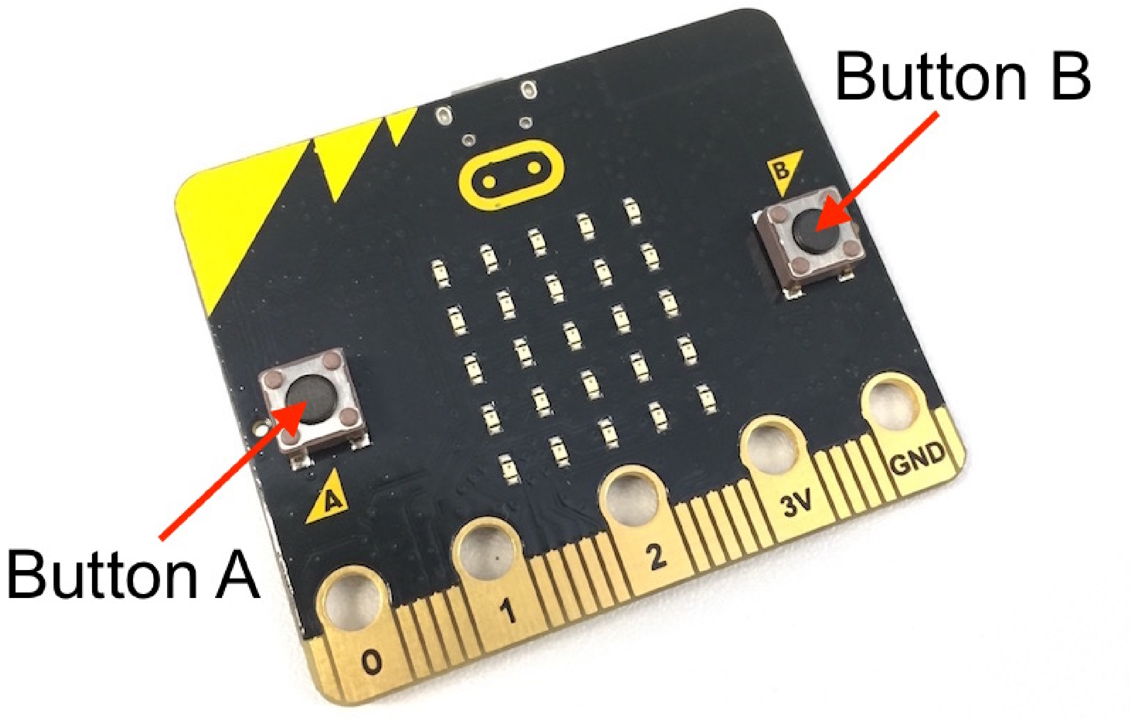


The diamond requires a yes or no decision. The answer determines which line we follow. If the answer is yes, we do the extra step of showing an image. If the answer is no no, we skip it.

1.2.9. Buttons

The BBC micro:bit has two buttons.

One is A. The other one is B.



We can use button A is pressed and if to make decisions.

1.2.10. 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.

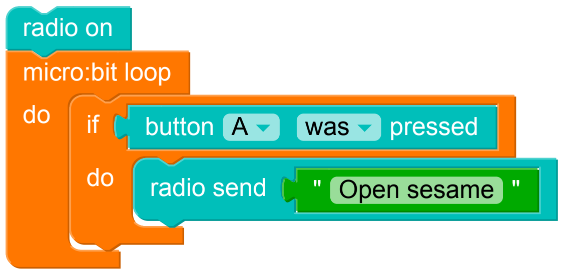
This takes a special block called an *if statement*. In this code we check to see if button A was pressed.

If it's true that the button was pressed, then we follow the instruction inside the if block. If it's false, the button wasn't pressed, then we skip it.

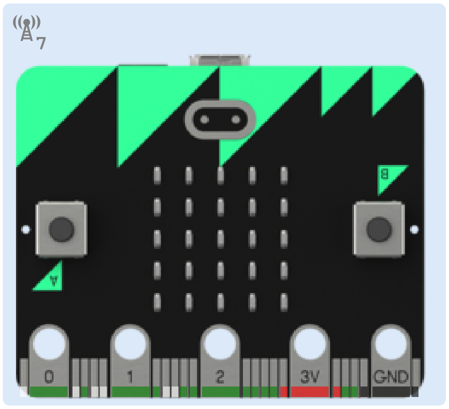
1. Change the code ⌨️

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



```
radio on
micro:bit loop
do  if  button A was pressed
    do  radio send "Open sesame"
```



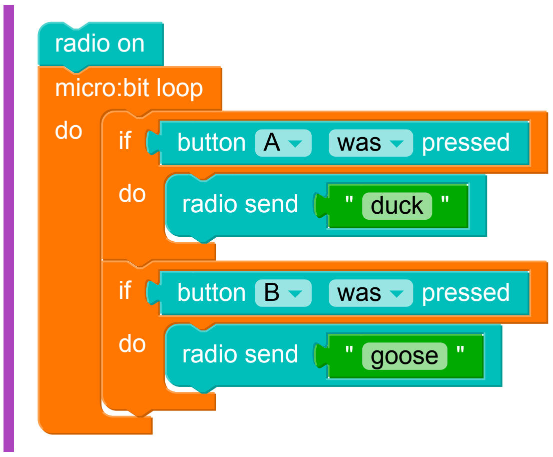
Use your mouse or keyboard

Press the buttons in the examples either by clicking with your mouse, or by pressing A or B on your keyboard.

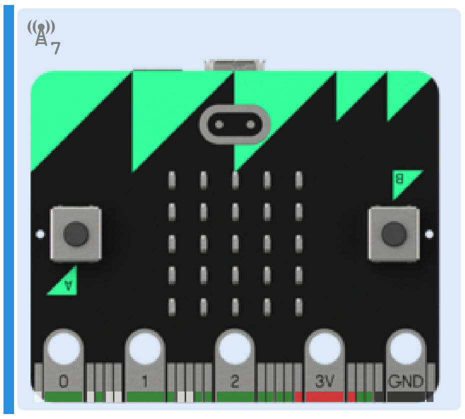
1.2.11. Multiple button presses

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

1. Change the code 🦆🐤🐦🦃 and run it!



```
radio on
micro:bit loop
do  if   button A was pressed
    do   radio send " duck "
    if   button B was pressed
    do   radio send " goose "
```

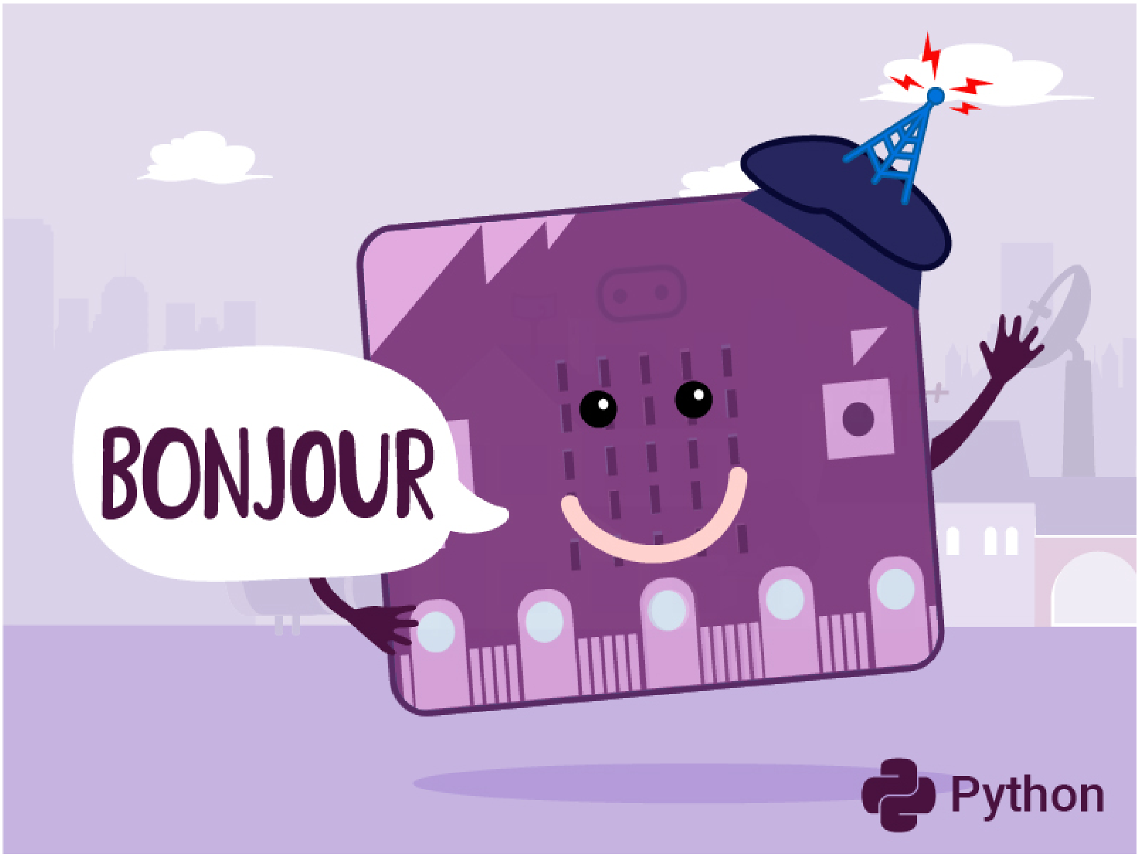


1.2.12. Problem: Hello, bonjour!

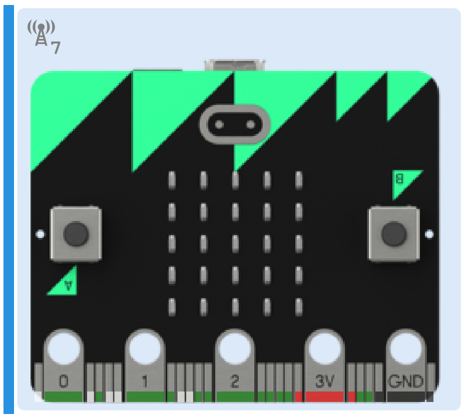
You have a French exchange student in class, and you want to make them feel welcome.

Write a program to send some a greeting in English and French 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

Testing

☐ Testing that your code contains an infinite loop.

- ☐ Testing `button A`.
- ☐ Testing the message on `button A` is correct.
- ☐ Testing `button B`.
- ☐ Testing the message on `button B` is correct.
- ☐ Bonjour!

1.3. Radio receive

1.3.1. Variables

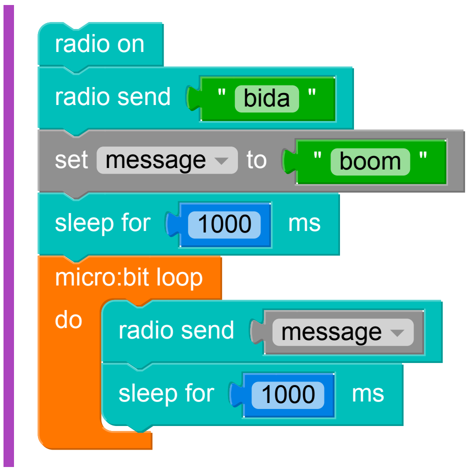
We can store messages in a variable. A variable is a place for storing data.

Each variable has a *name* like `message` (which we decide) and a *value* like `" strings! "` (which we put in it).

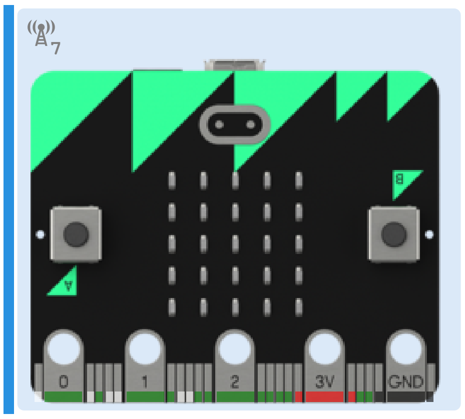
We create a new variable and give it a value using the `set message` block

When we give `radio send` a variable our program transmits its value!

1. Change the code and run it!



```
radio on
radio send " bida "
set message to " boom "
sleep for 1000 ms
micro:bit loop
do
    radio send message
    sleep for 1000 ms
```

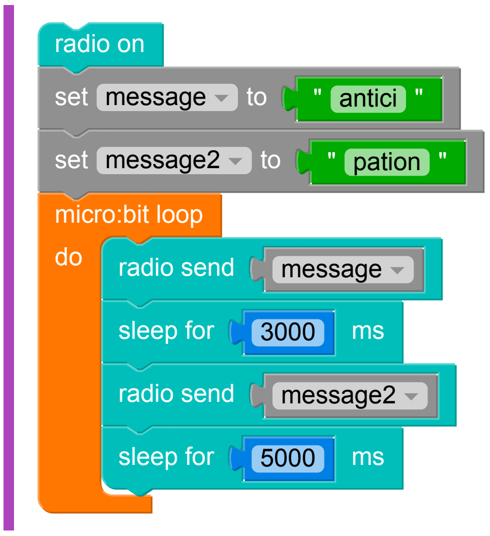


1.3.2. Radio send using a variable

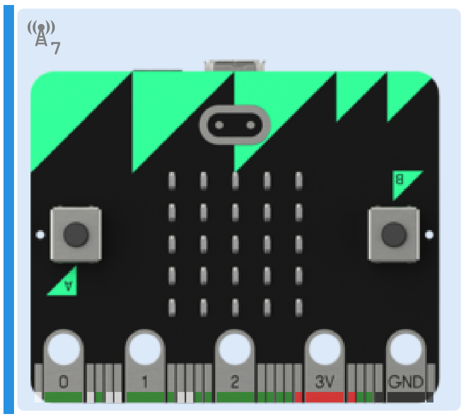
This time we're going to use two variables.

Hint!

Remember to play around with the code!



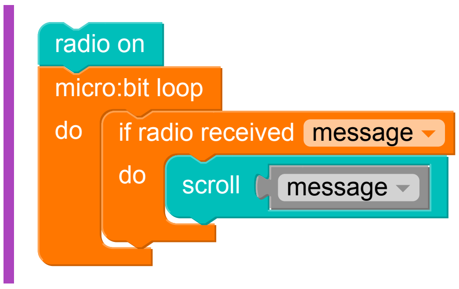
```
radio on
set message to " antici "
set message2 to " pation "
micro:bit loop
do  radio send message
    sleep for 3000 ms
    radio send message2
    sleep for 5000 ms
```



1.3.3. Waiting for a message

To receive and store a message we use the special block `if radio received message`

This checks if there is an incoming message and `if` there is, it stores it in a variable called `message`.



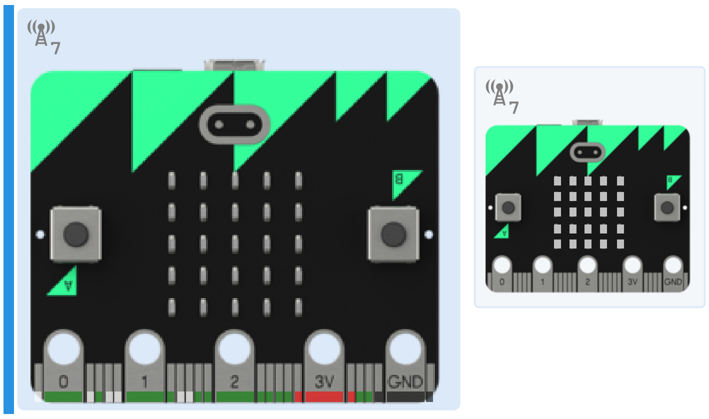
```
radio on
micro:bit loop
do  if radio received message
    do  scroll message
```

1. We turn the radio on
2. We call the `if radio received message` block to check if a message has been received.
3. `if` there is a message, we `scroll` it

The small micro:bit sends `Open sesame` when you press Button A. The program above scrolls the message on the large micro:bit:

Hint!

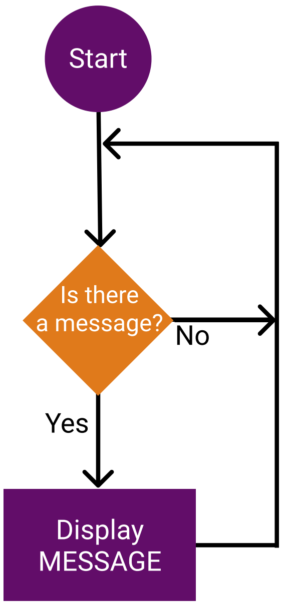
Remember to play around with the code!



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.

1.3.4. Receive flowchart

Our receive flowchart looks like this:



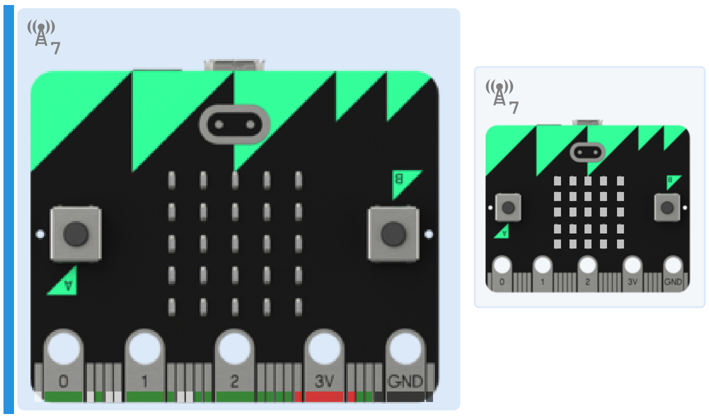
1.3.5. Problem: Received a scroll

Now let's write the program ourselves.

Your program should receive the radio message from the greeting in the previous problem and scroll *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

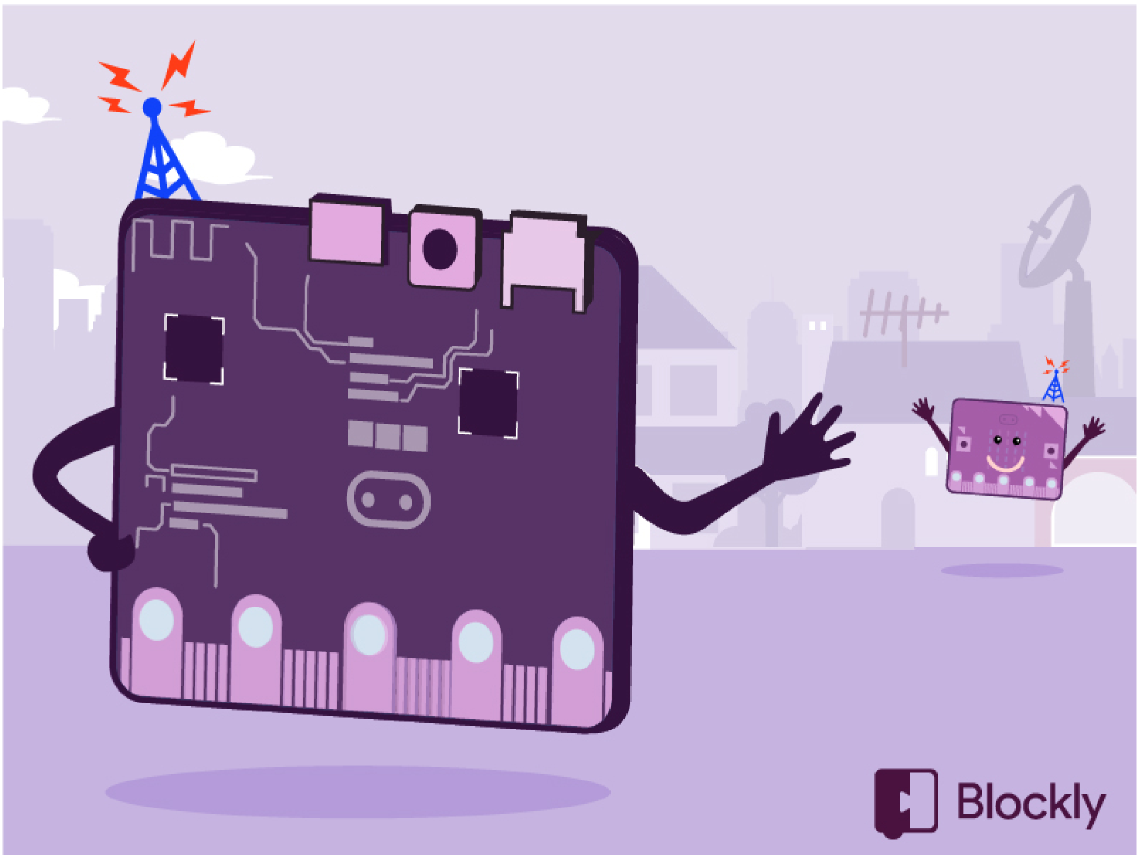
program.py

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 many messages.
- ☐ You received everything, you totally know what's going on! 👍

1.3.6. Put it together!

Now you've learned enough to both send and receive messages! Try loading the first question (`'hello, bonjour'`) onto one micro:bit and the 'receive scroll' code 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 both have micro:bits?**
>
> Lets use them both together!

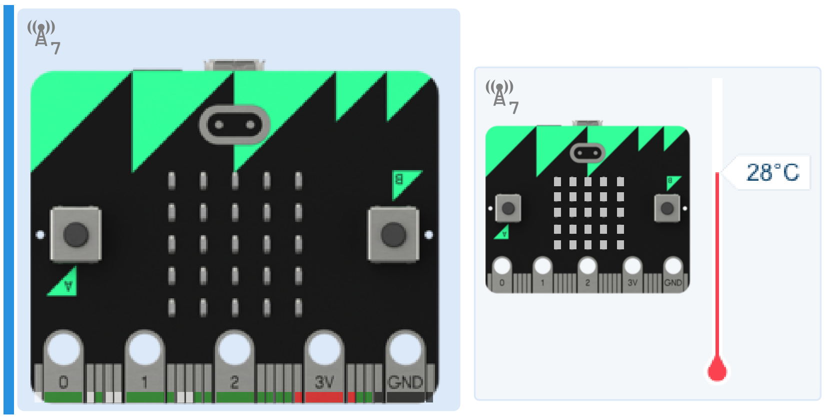
1.4. Sending numbers

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.

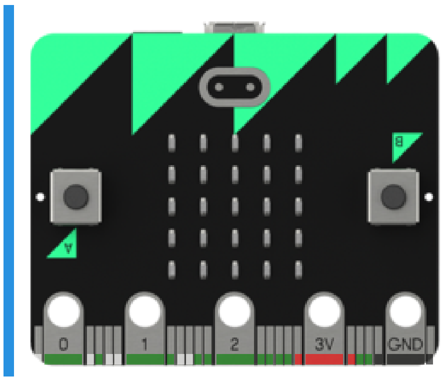


1.4.2. Numbers and strings

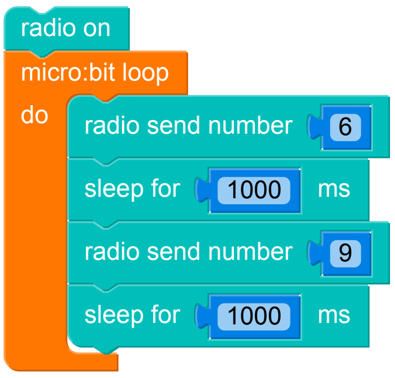
Numbers (like 27) are a different type of data than strings (like " bonjour ") so when we scroll and send them we need to use the scroll number and radio send number block.

1. Try changing the scrolled number, and running it.

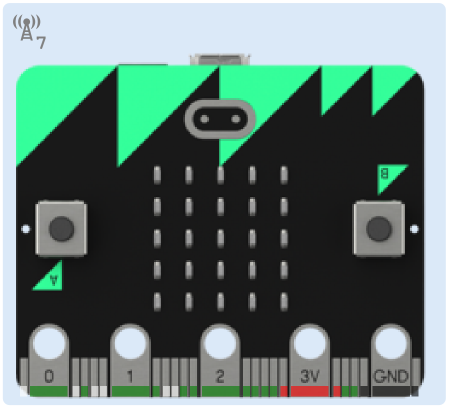
```
micro:bit loop
do
    scroll number 42
    sleep for 1000 ms
```



2. Try changing a sent number, and run it.



```
radio on
micro:bit loop
do
    radio send number 6
    sleep for 1000 ms
    radio send number 9
    sleep for 1000 ms
```



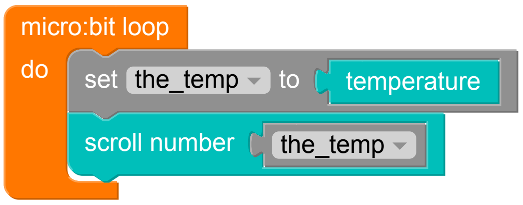
Important

When we make a remote sensor we will need to use the radio send number block

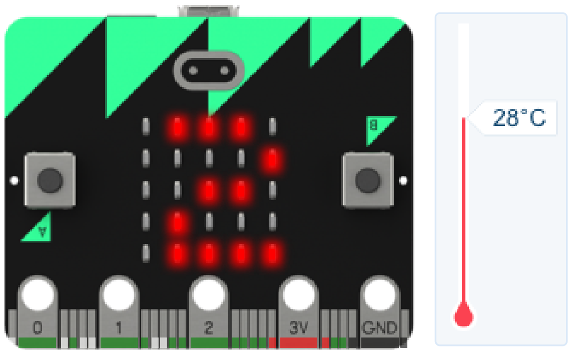
1.4.3. Measuring temperature

The micro:bit has a temperature sensor on the board. We can read it (in degrees Celsius or °C) by placing the temperature block inside set the_temp.

This example scrolls the temperature (using scroll number):



```
micro:bit loop
do
    set the_temp to temperature
    scroll number the_temp
```



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

Do you have a micro:bit?

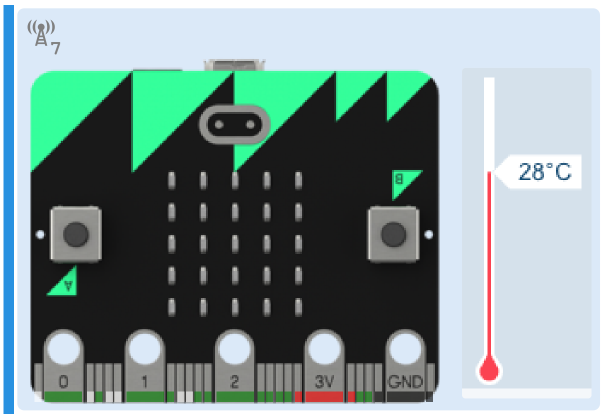
Try moving around with it and see what you can measure!

1.4.4. 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: We're using numbers now, so use the radio send number block.



Reading the temperature

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

You'll need

program.blockly

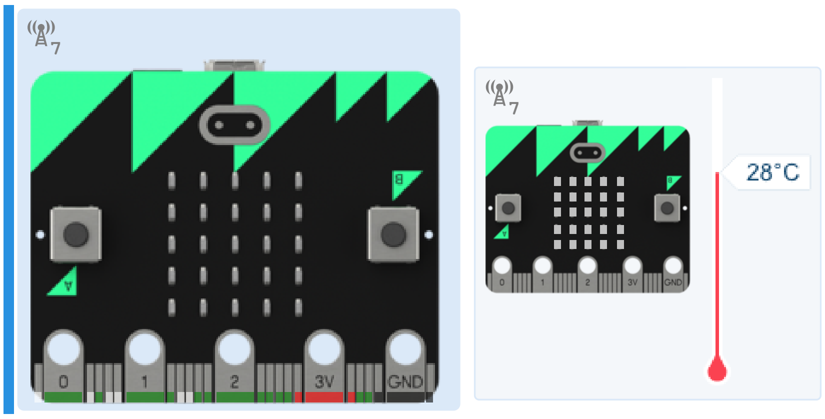
Testing

- ☐ 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.

1.4.5. Problem: Temperature Display

Now let's write the program that receives the radio message from the temperature sensor in the previous problem and scrolls it on our display.

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

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

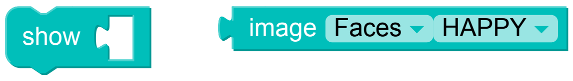
IMAGES AND STATES

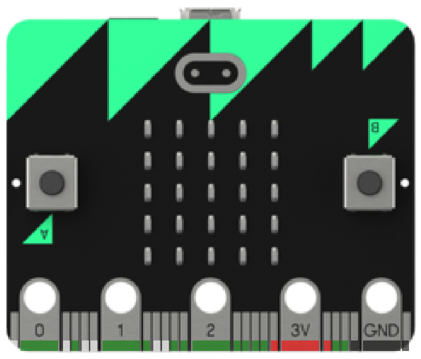
2.1. Images

2.1.1. Images

We can also use the microbit to display images!

1. Drag the image block into the hole in the show block.
2. Click ▶ to run the program. It shows a happy face!





2.1.2. More images

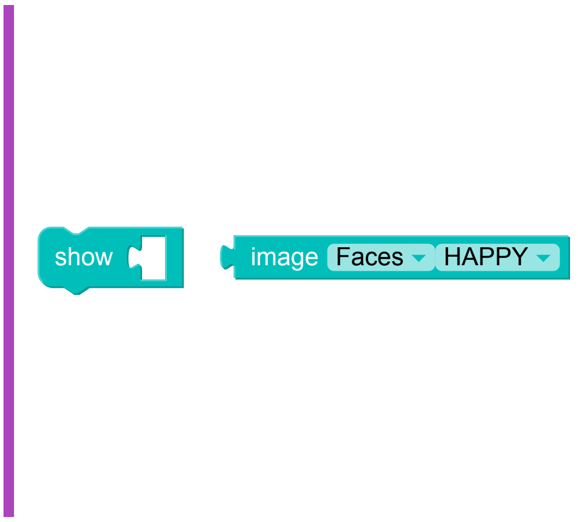
There are lots of pictures included in image for you to use. We've already seen HAPPY.

Here are some of our favourite images:

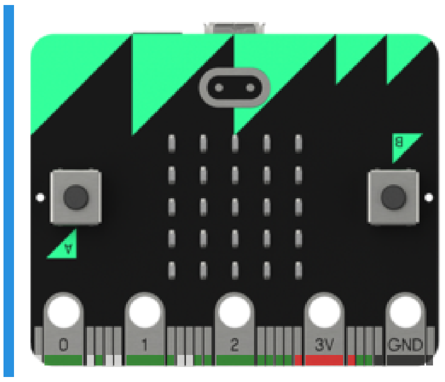
Name	Image
HAPPY	
HEART	
DUCK	
PACMAN	
ARROW_N	
ARROW_E	
SAD	

GIRAFFE	
BUTTERfLY	
GHOST	
ARROW_S	
ARROW_W	

1. Drag the image block into the hole in the show block.
2. Change the image to anything other than HAPPY
3. ▶ Run it to show *your* image!



```
show
image Faces HAPPY
```



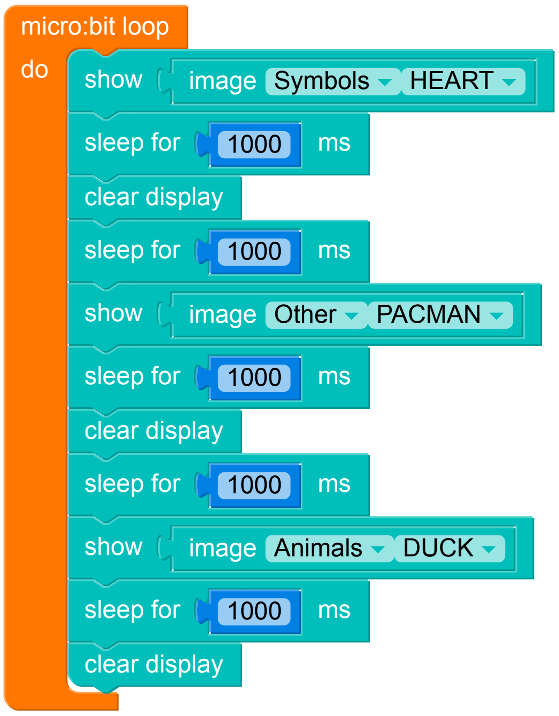
2.1.3. Clearing images

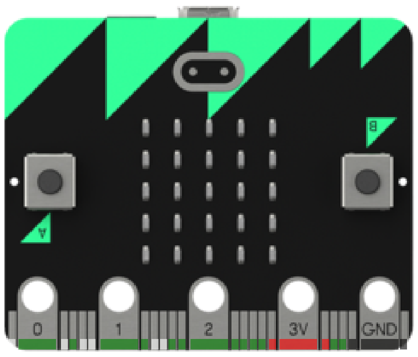
We can clear the screen using the block clear display.

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

Hint

Mess around with the code!

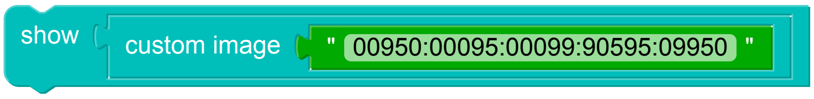

```
micro:bit loop
do  show image Symbols HEART
    sleep for 1000 ms
    clear display
    sleep for 1000 ms
    show image Other PACMAN
    sleep for 1000 ms
    clear display
    sleep for 1000 ms
    show image Animals DUCK
    sleep for 1000 ms
    clear display
```

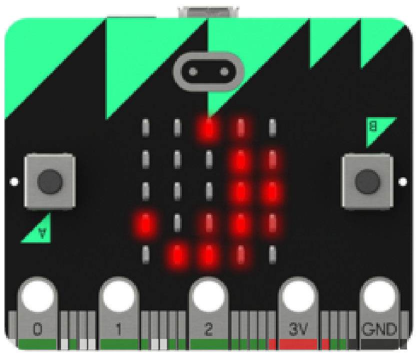


2.1.4. Image strings

So far, we've only used the built-in images from the Image block, like HEART or HAPPY.

Images can also described by " strings ". Here we've used custom image to design our own MOON block.


```
show custom image " 00950:00095:00099:90595:09950 "
```



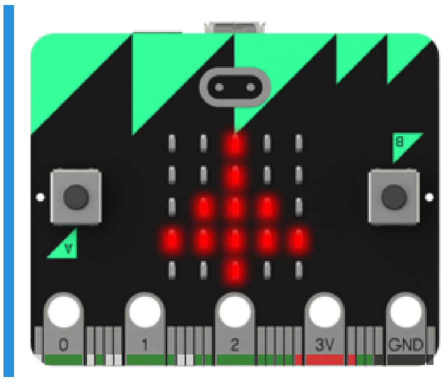
The numbers in the `string` block control the brightness of each pixel in the image. There are five lots of five numbers separated by a colon (`:`).

- The first five numbers control the top row, the second lot of numbers control the second row, and so on.
- Each number is the pixel brightness from `0` (off) to `9` (fully on).

2.1.5. DIY images

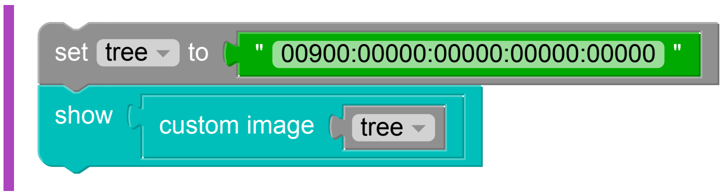
Lets make our own image.

1. Click the top ▶ button to run the program and see the custom image

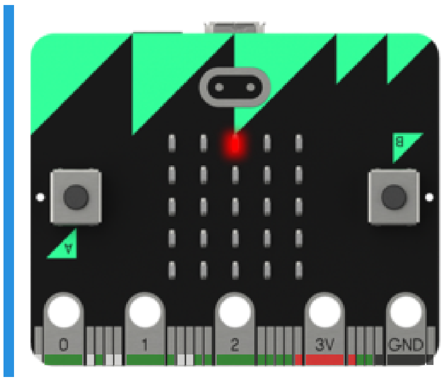


Try recreating this image of a tree. All of the pixels are at maximum brightness (`9`). We've started the first line for you.

2. Recreate the image of the tree on the lower micro:bit by coding it yourself. Make sure it's exactly the same as the one above!



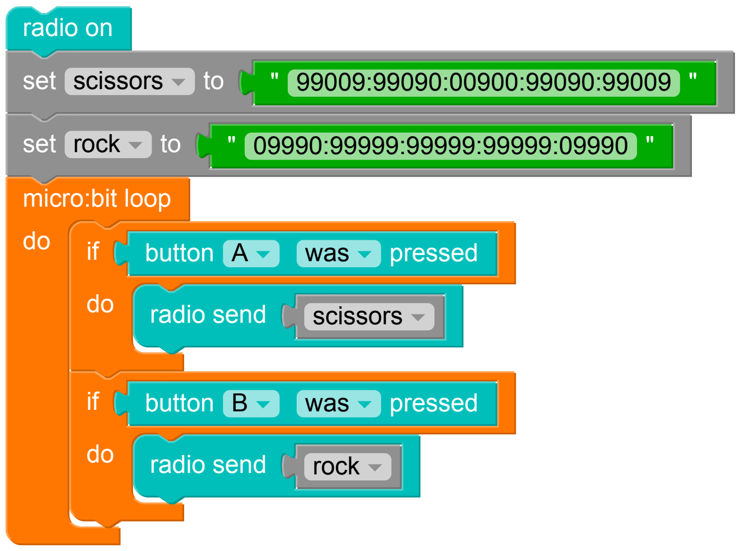
```
set tree to " 00900:00000:00000:00000:00000 "
show custom image tree
```



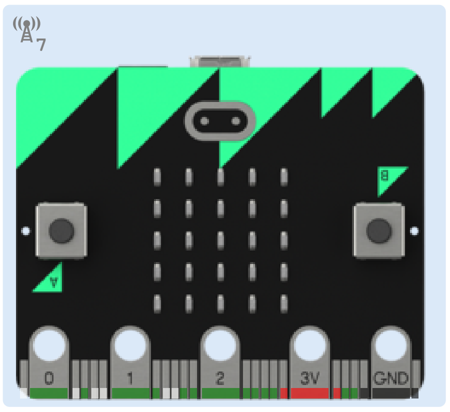
2.1.6. Sending images

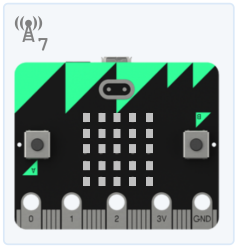
To send a `custom image`, we can just send the image `string`!

Press the buttons on the large micro:bit to send the image strings. Don't forget to play around with the code!



```
radio on
set scissors to "99009:99090:00900:99090:99009"
set rock to "09990:99999:99999:99999:09990"
micro:bit loop
do
    if button A was pressed
    do
        radio send scissors
    if button B was pressed
    do
        radio send rock
```





2.1.7. Problem: Send an image!

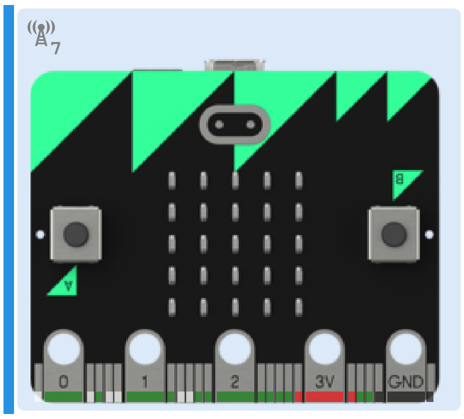
Write a program that creates an image string, then sends it over radio every 5 seconds.

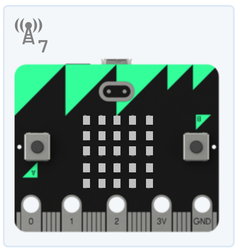
An image string has the format of 5 lots of 5 numbers separated with a colon (:). Each number is the brightness of that pixel, and must be 0 to 9 (where 0 is off).

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.blockly

```
radio send " 00900:00900:09990:99999:00900 "
```

Testing

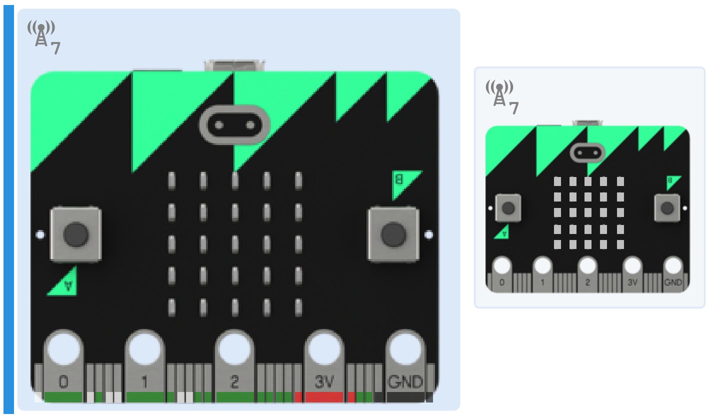
- ☐ Testing that your program contains an infinite loop.
- ☐ Testing your message has the correct number of colons.
- ☐ Testing your message has the correct format.
- ☐ Testing that your message is sent every 5 seconds.
- ☐ You sent an Image! 😊

2.1.8. Problem: Receive an image

Your friends want to send you different pictures, but it's boring to only have a few preset options. It's much better if they can send you *any* image!

Write a program to show the custom 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



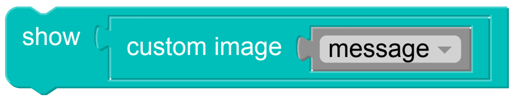
Your program should work on *any* image.

Hint!

Remember to use the custom image block before showing the message. It's been pre-loaded into your code window.

You'll need

program.blockly



```
show custom image message
```

Testing

- ☐ Testing your program contains an infinite loop.
- ☐ Testing on a scissors image. ✂️
- ☐ Testing on a rock image. 🤘

- ☐ 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?

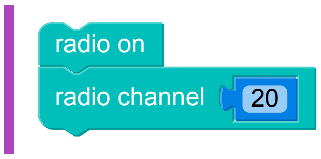
To solve this problem, we can send and receive messages on different *channels*. Different channels send messages on radio waves at different frequencies.

Source: [Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Autoradio_panasonic.jpg)](https://commons.wikimedia.org/wiki/File:Autoradio_panasonic.jpg)

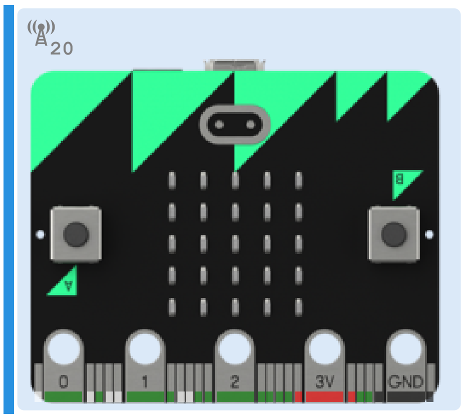
This is also how changing the TV channel or radio station works!

2.2.2. Tuning in to a channel

We use the `radio channel` block to set our channel:



```
radio on
radio channel 20
```



This program sets the radio to channel `20` (the default channel is `7`).

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

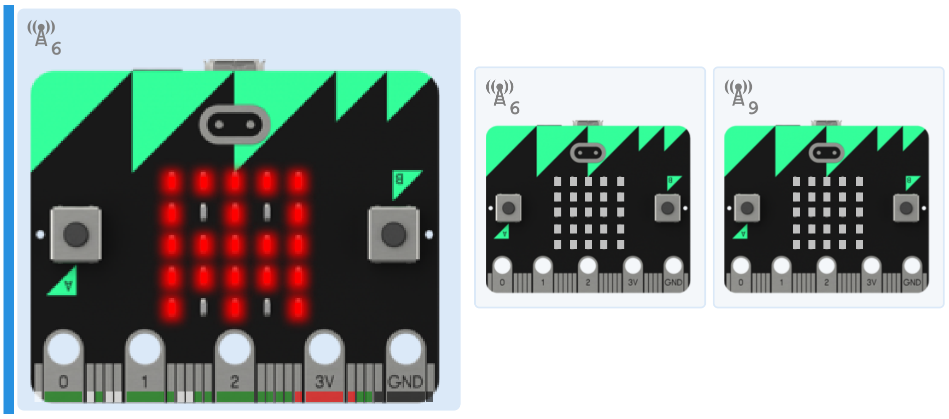
We've set up 2 simulated micro:bits that are broadcasting "TV shows" on 2 different channels:

- 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.

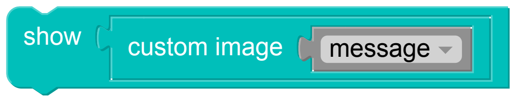
Write a "television" remote to switch between the two the TV shows. Your program should start on channel 6. Pressing button A should make it go to channel 6 and button B should make it go to channel 9.

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



You'll need

program.blockly



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

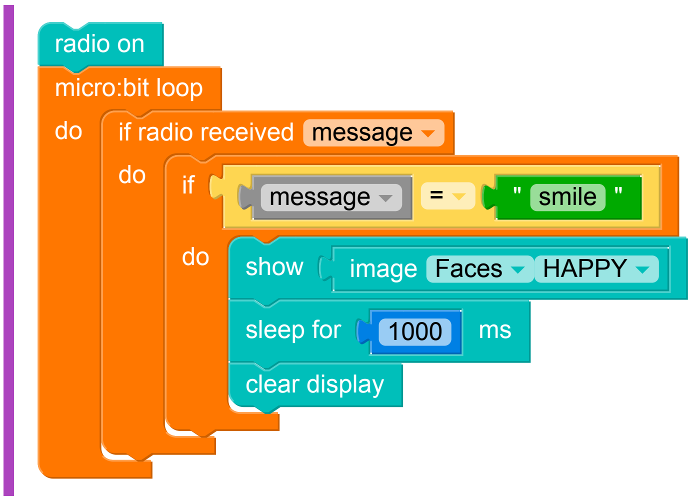
2.3.1. Comparing messages

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

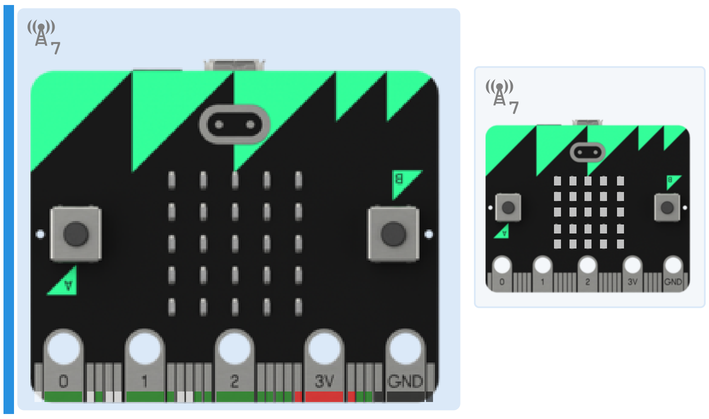
We can make logical decisions about the received message with the = block to compare the message with another string. If the message is the same as the string, it reads true, but if they are different it reads false.

Press the button A on the small micro:bit to send " smile " and test how the large micro:bit reacts!

Try changing the program to send something other than smile. What happens?



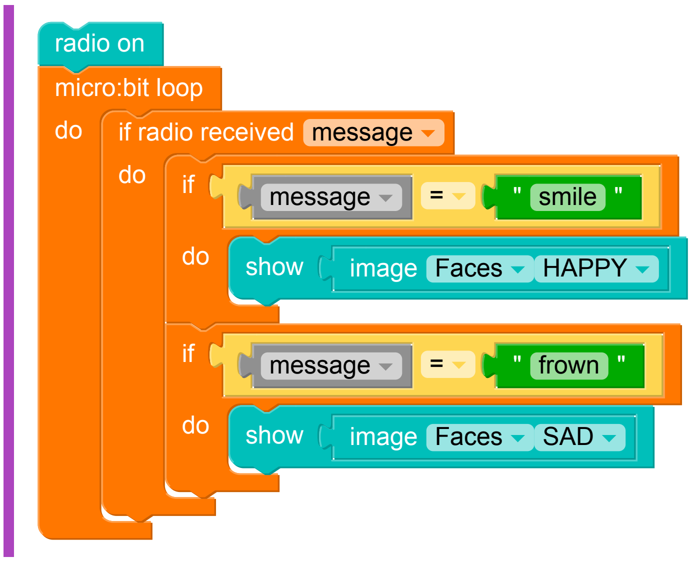
```
radio on
micro:bit loop
do  if radio received message
    do  if  message = " smile "
        do  show image Faces HAPPY
            sleep for 1000 ms
            clear display
```



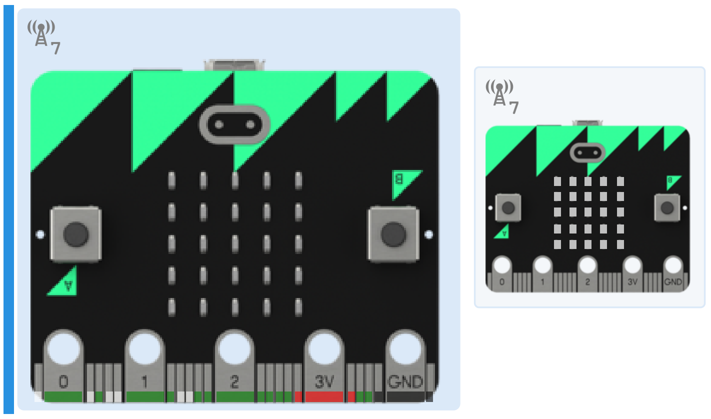
2.3.2. Check all the messages!

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.



```
radio on
micro:bit loop
do  if radio received message
    do  if message = " smile "
        do  show image Faces HAPPY
        if message = " frown "
        do  show image Faces 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?

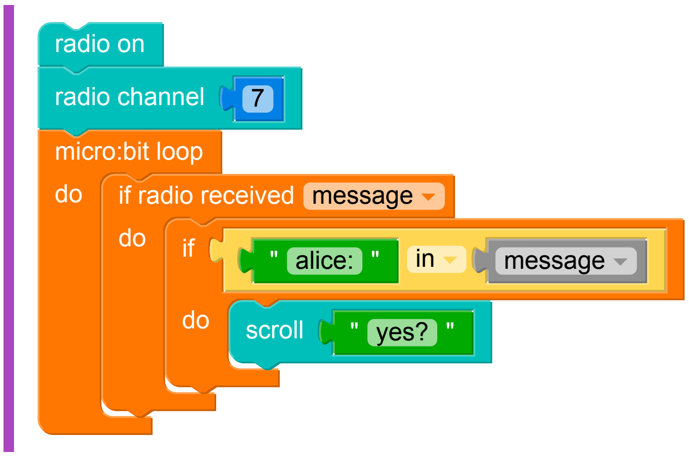
These sorts of problems are really common in computer science. We're going to learn how to solve a few of them by filtering messages.

2.3.5. Filter messages

We can use the `in` block to check if the string contains a particular word or symbol.

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

She can use the following code to filter out all messages that don't have `" alice: "` in them.

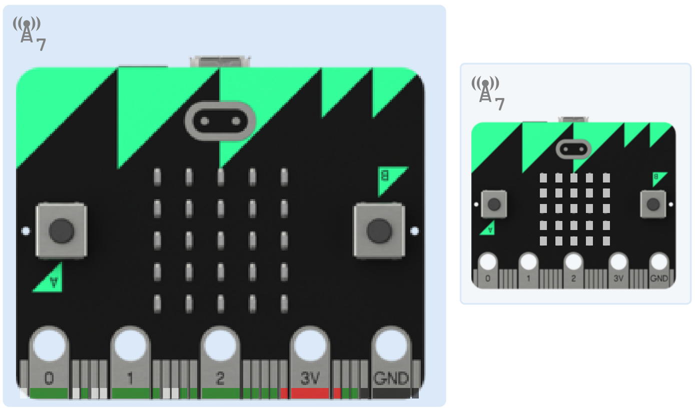


```
radio on
radio channel 7
micro:bit loop
do  if radio received message
    do  if " alice: " in message
        do  scroll " yes? "
```

Watch out!

We need to use the `if radio received message` block to first check whether there is a message at all, then use another `if` block to check if the `message` contains `" alice: "`.

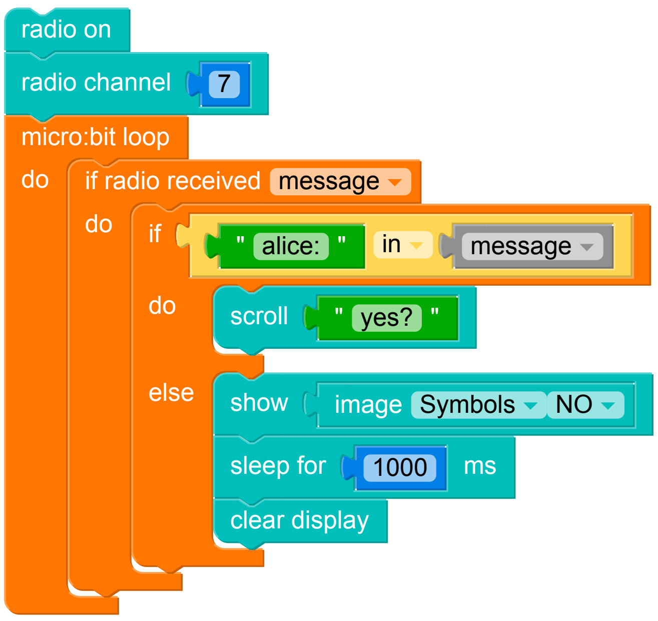
Try it out! Pressing the `button A` on the small micro:bit will send a message that starts with `" alice: "`, and pressing the `button B` will not.



2.3.6. More filtering

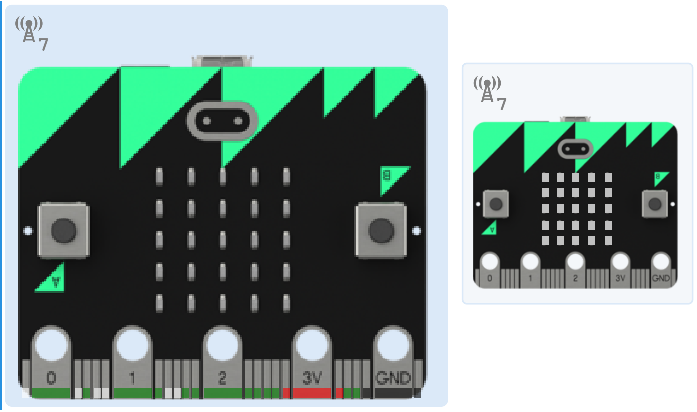
Sometimes we want to react to messages that don't contain a particular string.

In that case we can use the `if .. do .. else` block. This lets us take one action if something is true, and a different action if it isn't.



```
radio on
radio channel 7
micro:bit loop
do  if radio received message
    do  if " alice: " in message
        do   scroll " yes? "
        else show image Symbols NO
             sleep for 1000 ms
             clear display
```

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

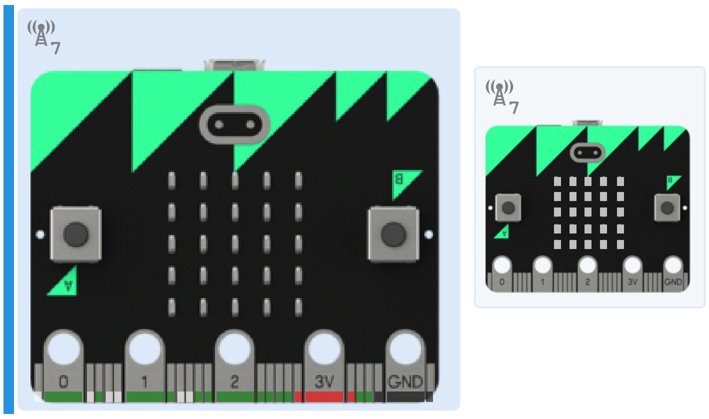
Your microbit, called Jeff, is asleep in class. If the microbit receives a message, it should scroll " zzz " on the screen unless the message starts with " jeff: ".

Once it hears its name, the microbit will wake up, transmit " what? " over the radio and display a question mark for 3 seconds. We've given you a question image:

```
set question ▾ to " 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.blockly

```
radio on
set question ▾ to " 09990:90009:00990:00000:00900 "

show ( custom image ( question ▾ ) )
```

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 if it were true, they'd still be doing better than our last program.
Source: [Wikimedia Commons (https://commons.wikimedia.org/wiki/File:%E3%83%AF%E3%82%AD%E3%83%B320120701.JPG)](https://commons.wikimedia.org/wiki/File:%E3%83%AF%E3%82%AD%E3%83%B320120701.JPG) ぱたごん CC BY-SA 3.0

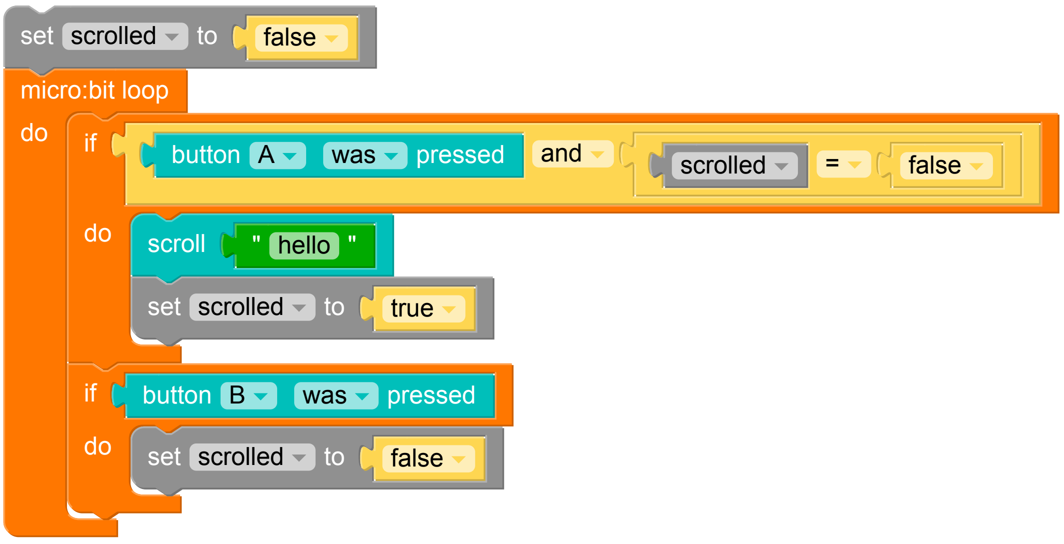
If we want to make a more intelligent program we can do it using boolean variables.

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.

Another useful block is the `and` block. This allows us to check `if` two things are true at once.

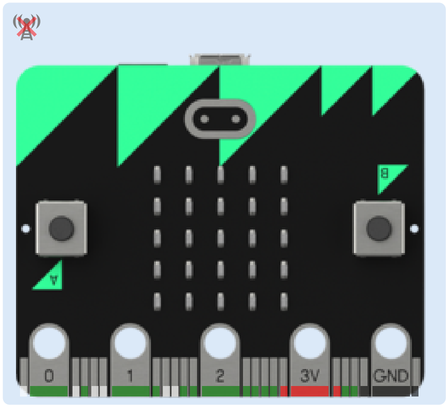
For example:



```
set scrolled to false
micro:bit loop
do
    if button A was pressed and scrolled = false
    do
        scroll "hello"
        set scrolled to true
    if button B was pressed
    do
        set scrolled to false
```

This code lets you press button A to scroll "hello", but only once.

If you want to scroll it again, you need to reset it using button B.



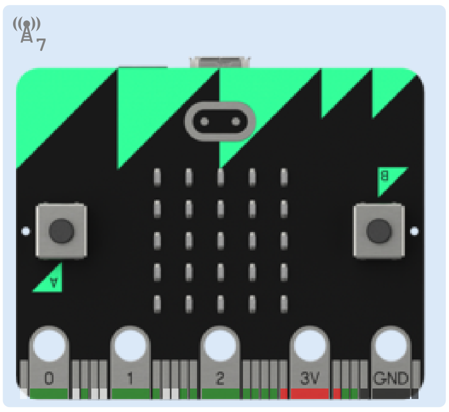
The sent variable keeps track if a message has been scrolled or not, so the program will only ever scroll a message *once*.

2.4.3. Problem: Matter of state

Write a program to that sends a message over the radio when `button A` has been pressed. But if `button A` is been pressed again, no message is sent.

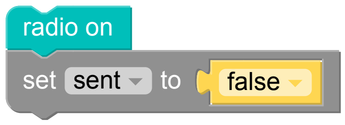
`button B` should reset the state, so a message can be sent again.

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



You'll need

program.blockly



```
radio on
set sent to false
```

Testing

- ☐ Testing your program has an infinite loop.
- ☐ Testing no message is sent initially.
- ☐ Testing a message is sent after `button A` has been pressed.
- ☐ Testing no message is sent when `button A` is pressed again.
- ☐ Testing `button 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:

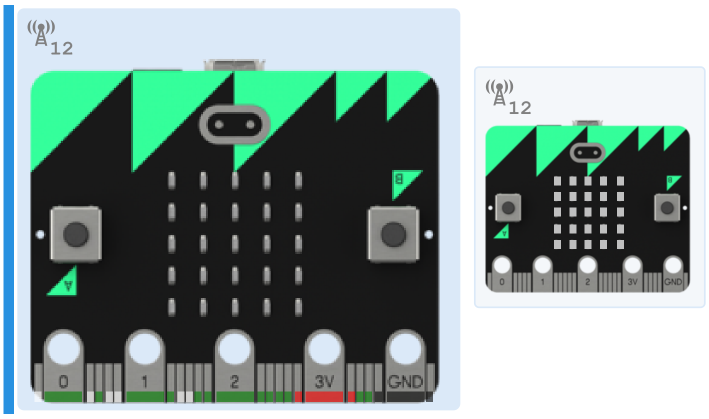
- All messages should be sent on radio channel 12
- If you press button A it should send your message
- If you press button B it should be ready to receive a message and show a question mark. ❓
- If you receive a message and you have displayed a question mark, your should scroll the message that has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

The question mark code is provided for you.

```
custom 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.blockly

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! 🎙️🙌🏽📱 🕶️🔷 🤵

2.5.2. Problem: Radio playground

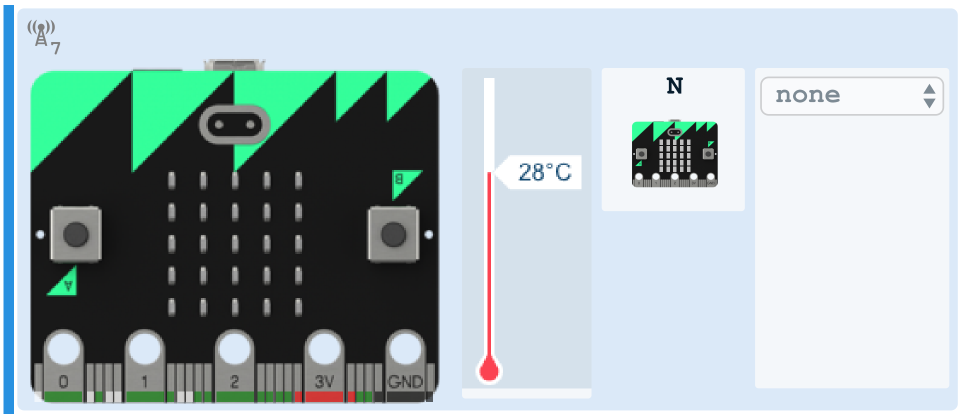
What's this, thought you were finished!?

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 reader (gives you the compass direction degrees) and a gesture sensor (which is true or false if something has happened...).

See if you can work out where these new blocks go!



Happy coding!

You'll need

program.blockly

Testing

☐ This is a playground question, there is no right or wrong!