Australian
Computing
Academy

DT Challenge Blockly

Smart Garden

1. Displaying images and text
2. Sensors and pins
3. Buttons & Custom Images
4. Putting it all together

The Australian Digital Technologies Challenges is an initiative of, and funded by the Australian Government Department of Education (https://www.education.gov.au/).

1

DISPLAYING IMAGES AND TEXT

1.1. Let's get started

1.1.1. Sensing the world around you

A biologist uses sensors to measure soil temperature and moisture. Public Domain, NPS photo. (https://www.flickr.com/photos/alaskanps/8446139444)

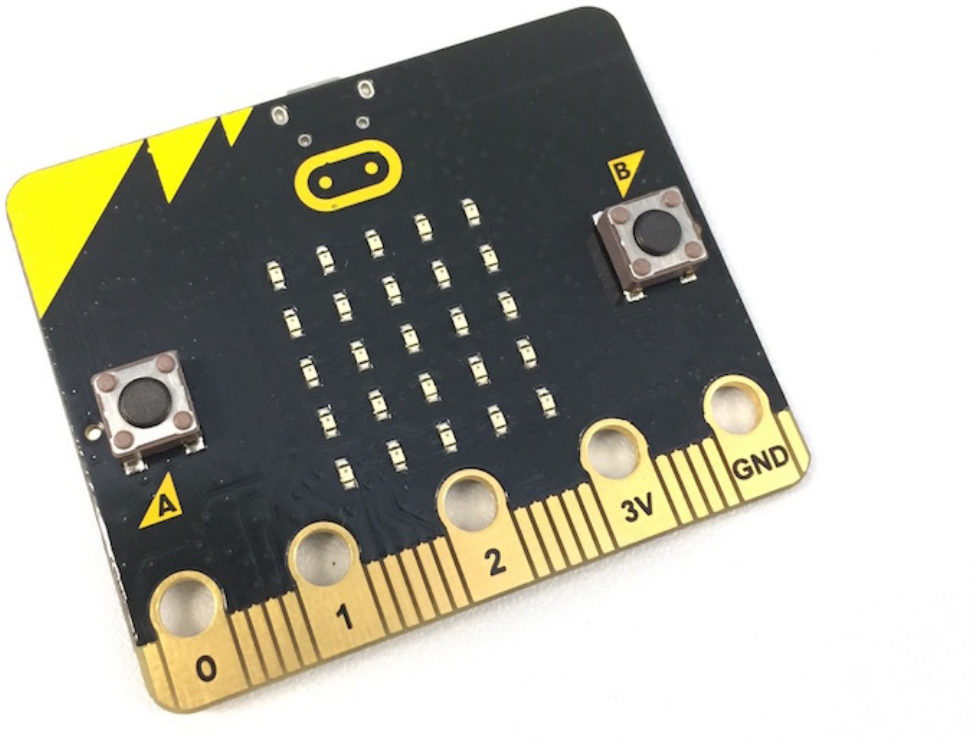
Scientists measure the world around us using all sorts of devices. For example, biologists studying plant growth use thermometers to measure temperature, light meters to measure how the Sun's brightness, and moisture sensors to measure how wet the soil is. Many of these devices run on simple computers.

In this course you'll make your own Smart Garden device, using the BBC micro:bit. Along the way, you'll learn three things:

- how the growth and survival of living things are affected by physical conditions of their environment
- how to write computer programs for the micro:bit with Blockly
- how to use the micro:bit's built-in sensors and external connections to measure the world around you

1.1.2. Introducing the BBC micro:bit!

The BBC micro:bit (https://www.microbit.co.uk/) is a tiny computer. You can program it with blocks .



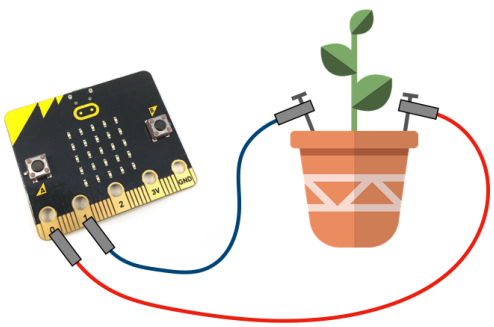
The micro:bit has:

- **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.

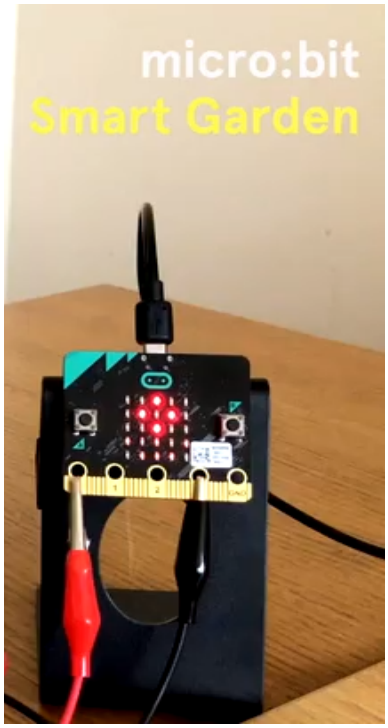
1.1.3. DIY Smart Garden

We're going to build a device with the micro:bit called a Smart Garden. It will use the built-in temperature sensor, and you will learn how to connect a home-made soil moisture probe to the micro:bit.



By the end of this challenge, your Smart Garden will:

- **measure the temperature** when you push one of the micro:bit's buttons
- **measure the amount of moisture in the soil** when the other button is pressed
- display the temperature and moisture levels using special, **customised images**



0:00 / 0:43

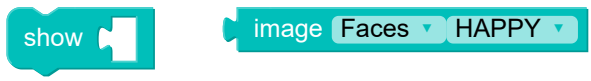
The micro:bit Smart Garden in action! (https://groklearning-cdn.com/modules/cU5geH65oKwqHZLrVZLEZg/SmartGarden4.mp4).

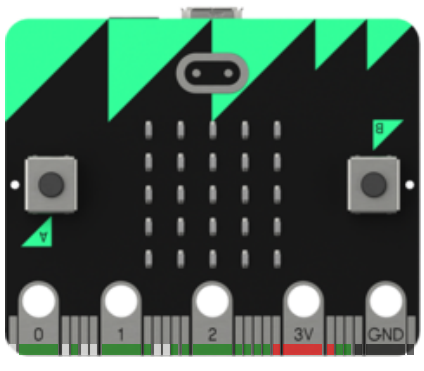
1.2. Displaying images

1.2.1. Hello, micro:bit!

Let's jump right in with a program on the micro:bit — put an image on the LED display:

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



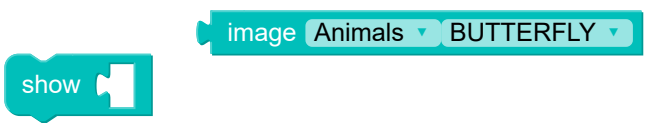


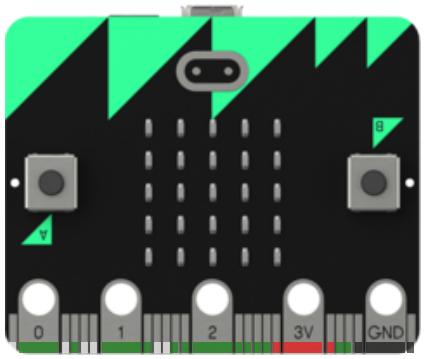
Congratulations! You've written your first micro:bit program!

1.2.2. Change the face

Let's put a different image on the display!

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





You changed the image! Nice work! Play around with the example above to see all the different images the micro:bit has.

Can you make your own images?

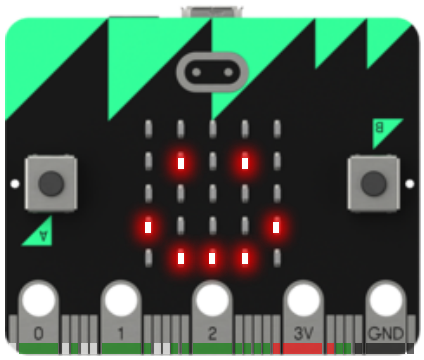
We'll see later on how to make your own images for the micro:bit.

1.2.3. Problem: Happy micro:bit!

Let's get the micro:bit to show a smiley face.

Follow the steps to complete your first problem:

1. Join the blocks in the problem editor together.
2. Click the ▶ Run button.
3. Click the ★ Mark button to **Submit**.



You'll need

program.blockly

Testing

- [ ] Testing that the display is showing a happy face.
- [ ] **Congratulations, you've written your first micro:bit program!**

1.2.4. Building Blocks

You won't always have the blocks you need ready to go — sometimes you'll need to gather more blocks to make your program.

To get a new block:

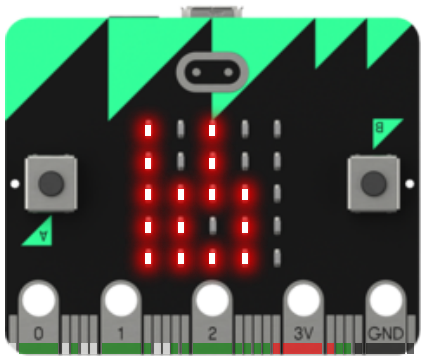
1. Click the micro:bit drawer (this is the micro:bit button on the grey part of the screen).
2. Drag the block .
3. Make your program!

Dragging blocks from the drawer

1.2.5. Problem: Wascally Wabbit

In this course, you're going to learn a bit about plants and their environments.

We can't easily show an picture of a *plant* on our micro:bit ... but we can show an animal! So let's make the display show a rabbit, like in this example:



1. Click the micro:bit drawer.
2. Drag the show block.
3. Create the program to show a rabbit.
4. Run and Mark your program.
5. Yay! You have made a fluffy micro:bit rabbit!

Where's the rabbit?

You can find the rabbit image in the image Animals list.

You'll need

program.blockly

Testing

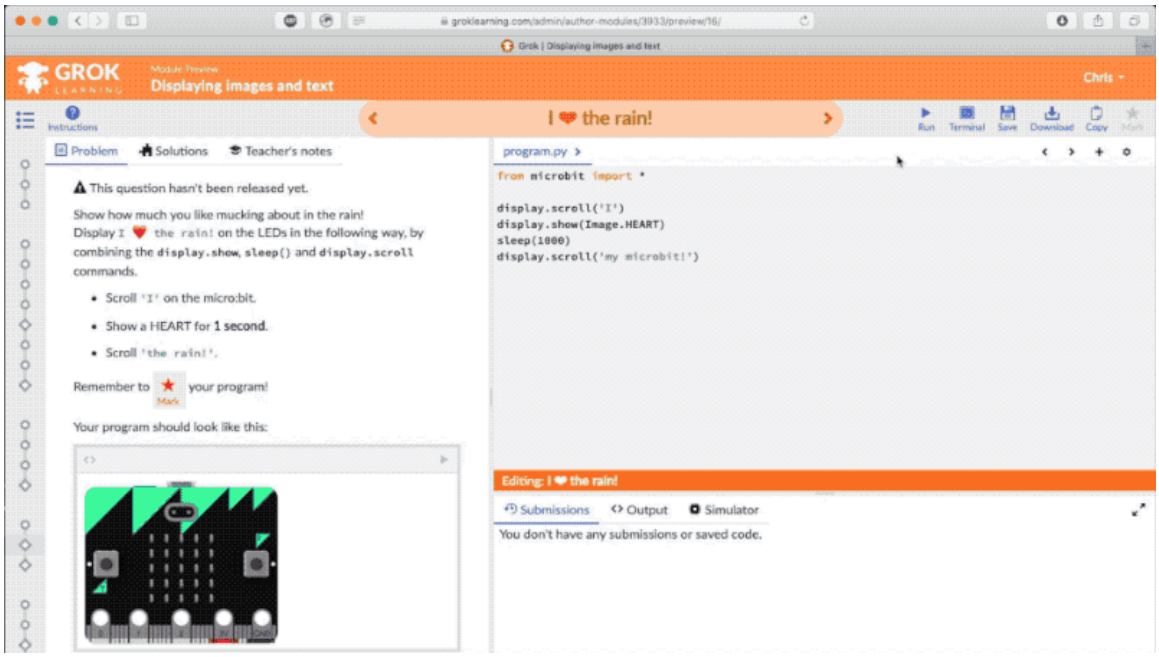
- [ ] Testing that the display is showing a rabbit.
- [ ] **Congratulations! You made a cute bunny wabbit!**

1.2.6. Downloading

If you have a micro:bit, you can see your program in real life! Put your code onto the micro:bit like this:

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

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



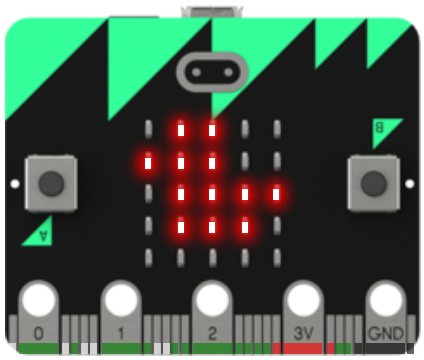
1.3. More than one

1.3.1. Giraffe Duck

OK, we can now show an image on the micro:bit. How do we show *two* images?

Let's try something. What if we put two image blocks together like the example below?

Click ▶ run to see what happens:



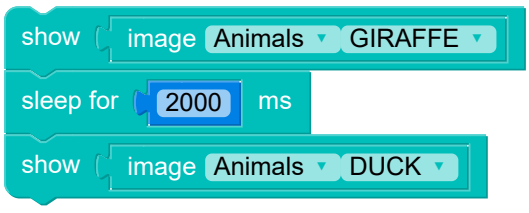
Uh oh! That's weird. **We only see a duck!**
Where did the giraffe go?

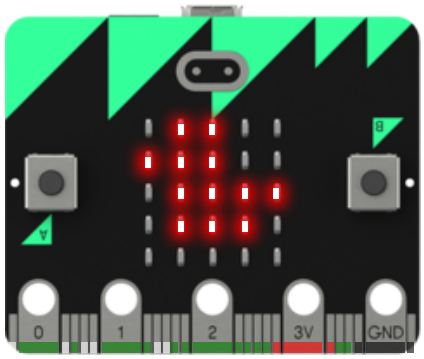
The micro:bit is *really* fast. It shows the giraffe, but it's too fast for us to see. We need to slow things down!

1.3.2. The sleep block

We can stop the micro:bit going too fast using sleep .

1. Click ▶ run.
2. The giraffe appears for 2 seconds. Then the duck appears.





Seconds and *milliseconds*

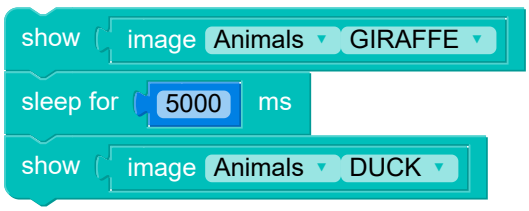
The sleep block measures time in *milliseconds* (or *ms* for short), not seconds. There are 1000 milliseconds in a second — just like there are 1000 millimetres in a metre!

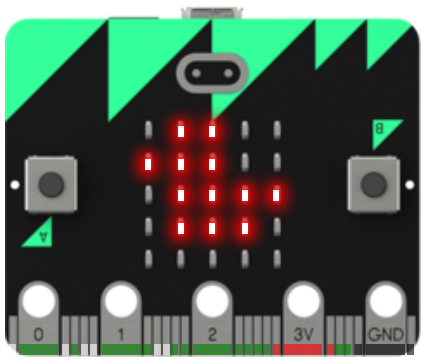
So you need to remember to put the sleep time in ms — for example, to pause the micro:bit for 3 seconds you need sleep 3000 ms .

1.3.3. How much sleep ?

We can sleep for different lengths of time.

The example below shows the giraffe for **5 seconds**. Click ▶ to run the example to see what happens:





Now try changing the example, like this:

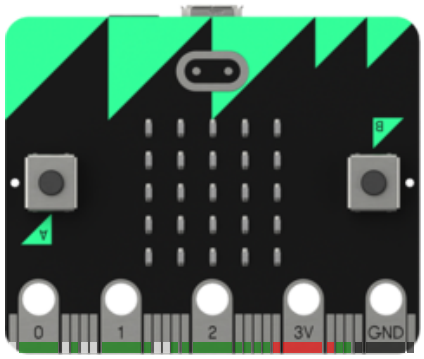
1. Change sleep 5000 ms to sleep 1000 ms .
2. Click ▶ to run the example again. The giraffe appears for only **one second** now!

1.3.4. Problem: I love the rain

It's great when it rains — as long as you remember your umbrella! Write a program to `show` an UMBRELLA for **one second**, then `show` a HAPPY face.

1. Drag the blocks you need from the workspace.
2. Select the correct images (look in the `image` Other list for the UMBRELLA, and in `image` Faces for the HAPPY face!)
3. Choose the correct `sleep` amount.
4. Join the blocks together.

Here's how your program should work — click ▶ to run the example:



Sleep for milliseconds

You need to use `sleep` `1000` `ms` to sleep for **one second**.

Testing

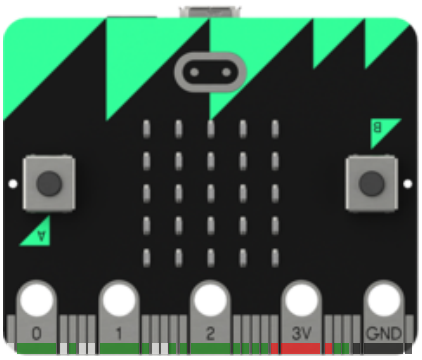
- [ ] Testing that the display starts with an umbrella.
- [ ] Testing that the umbrella is still on the screen less than 1 second later.
- [ ] Testing that the display changes to a happy face after 1 second.
- [ ] Testing that the happy face stays on the display for 1.5 seconds.
- [ ] **Congratulations!!**

1.4. Letters and words

1.4.1. Scrolling letters and words

So far we've used the display to show pictures. We can also show letters, words, even sentences, using the scroll block.

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



Now you know how to make the micro:bit display your name — *awesome!*

A string of letters

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

" I'm a string "

1.4.2. Problem: I ♥ the rain

Show how much you like mucking about in the rain!

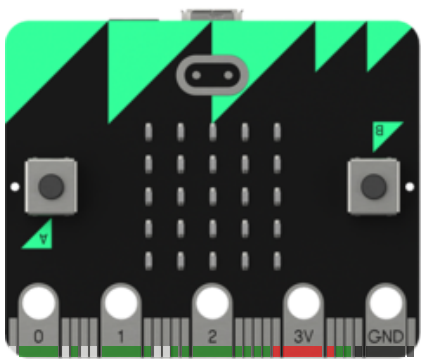
Display `I ♡ the rain!` on the LEDs in the following way:

1. Scroll " I " on the micro:bit.
2. Show a **HEART** image for **1 second**.
3. Scroll " the rain! " .

You already have the all the blocks you need — you just need to change them to make the program run correctly.

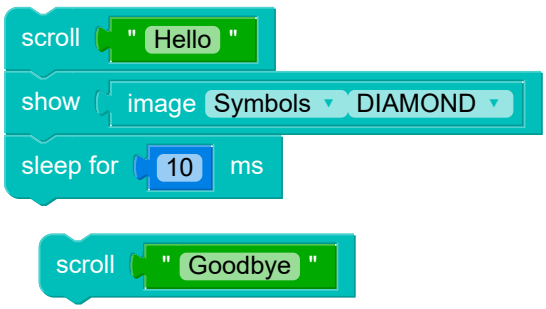
Remember to ★ Mark your program!

Your program should run like this:



You'll need

program.blockly



Testing

- [ ] Testing that an `I` scrolls past.
- [ ] Testing that a heart appears after the `I`.
- [ ] Testing that the heart stays on the display for 1 second.
- [ ] Testing that a `t` scrolls past.
- [ ] Testing that `the rain!` scrolls past.

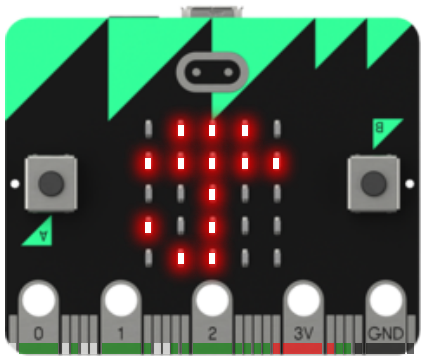
1.4.3. Problem: A 🦆 doesn't need an ☂!

Ducks love the rain!

Write a program to display `A 🦆 doesn't need an ☂`:

1. Scroll " A " on the micro:bit.
2. Then show a DUCK for **1 second**.
3. Then scroll " doesn't need an " .
4. Then show an UMBRELLA .

It should run like this — click ▶ to run the example:



Testing

- [ ] Testing that an `A` scrolls past.
- [ ] Testing that a duck appears after the `A`.
- [ ] Testing that the duck stays on the display for 1 second.
- [ ] Testing that `doesn't need an` scrolls past.
- [ ] Testing that an umbrella appears after the `doesn't need an`.
- [ ] Testing that the umbrella stays on the display.

1.5. Summary

1.5.1. What do living things need?

Take a closer look at the landscape in your local area — at your school, around your home, or in the local park. Do you notice similar sorts of plants growing in these areas?

Different plants thrive under different conditions. Some, like tomato plants require 8 hours of continuous sunlight and warm weather ... whereas raspberries thrive in cooler weather.

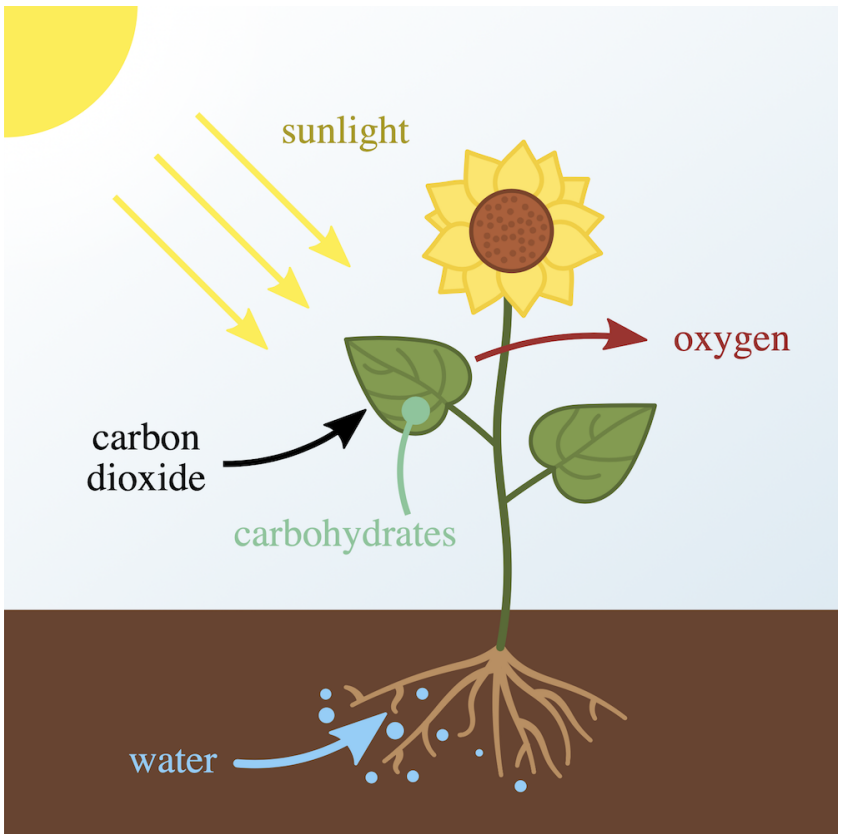
They also require different amounts of water, and different soil conditions too — they like different soil textures, and dirt with different kinds of nutrients in it.

In an earlier question, you displayed `I ♡ the rain!` on the micro:bit. Guess what, many plants ♡ the rain too!

1.5.2. Photosynthesis

What tomatoes and raspberries do have in common is that they use **photosynthesis**.

Photosynthesis is the process used by all plants and algae (and even some bacteria) to harness energy from the sun and turn it into food.



Plants use the Sun's energy to make their own food through a process called photosynthesis. CC-4.0, image by At09kg. (https://commons.wikimedia.org/wiki/File:Photosynthesis_en.svg)

All plants do this — but different plants have adapted to different types of soil, different amounts of water, and different amounts of sunshine!

1.5.3. Problem: Adaptation

Different plants have evolved to survive in very different envionments — they have **adapted** their shape, size and structure to grow happily in their natural surroundings.

Which of these is **not** an example of plant adaptation?

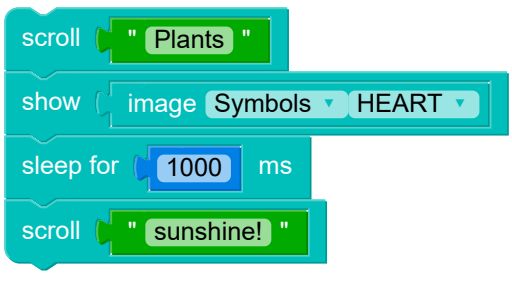
- ○ A cactus that stores water in its stem.
- ○ Rainforest plants with large, waxy leaves that catch sunlight and let water run off.
- ○ Water lilies with leaves and flowers that float on the water surface.
- ○ Tomato plants grown in a garden greenhouse to keep out the frost.

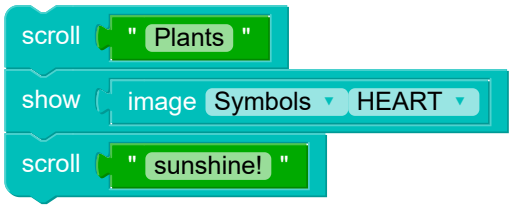
Testing

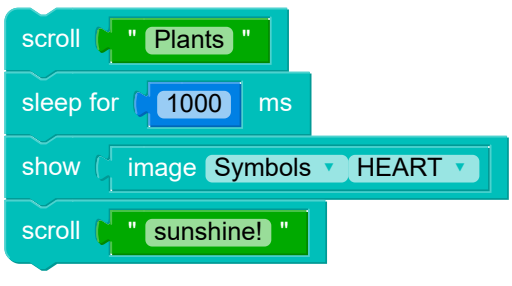
- ☐ That's right!

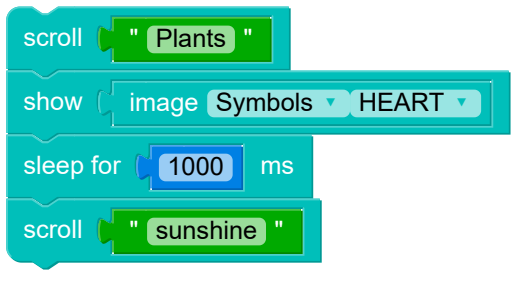
1.5.4. Problem: Plants love sunshine!

Which of the following code snippets will correctly display `Plants` ♡ `sunshine!` on the micro:bit's LEDs?

- 

- 

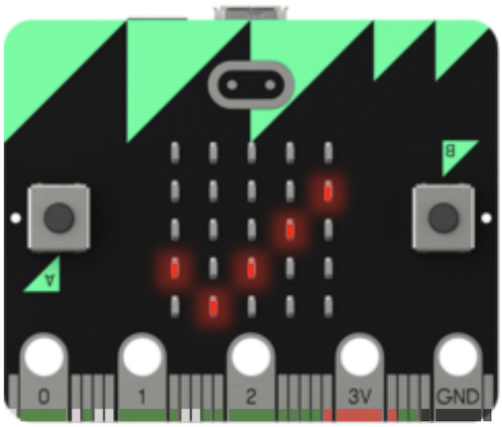
- 

- 

Testing

- [ ] That's right!

1.5.5. Congratulations!



Well done! You finished Module 1 — you're on your way to making your micro:bit Smart Garden!

We learned about:

- what's in the BBC micro:bit
- how to show an image on the micro:bit
- choosing your own image
- making the micro:bit wait with sleep
- scrolling " strings " on the micro:bit
- joining blocks together to make more things happen

Click » to learn how to use the micro:bit's temperature sensor, and build your own soil moisture sensor.

2

SENSORS AND PINS

2.1. Sensors and loops

2.1.1. What's needed for good science?

Testing predictions is an important part of science. To do this, scientists collect data and use evidence.

There are so many different types of data that scientists collect, and they use lots of different instruments and methods to collect it!

Left: A hydrologist measures streamflow (image by Deena Green ("https://www.flickr.com/photos/usgeologicalsurvey/15303060773"), Public Domain. Middle: Scientific weather station (image by Michal Osmenda ("https://commons.wikimedia.org/wiki/File:Weather_station_on_Mount_Vesuvius_(2437693238).jpg"), CC-2.0. Right: Seismograph used to measure earthquakes (image by Yamaguchi先生 ("https://commons.wikimedia.org/wiki/File:Kinemetrics_seismograph.jpg"), CC-3.0.)

During this challenge, You'll learn the skills to collect data with the micro:bit Smart Garden, which you can use to make predictions about the health of your plants.

2.1.2. Senses and sensors

In the olden days, a farmer might go outside to see what the sky looked like to work out whether they had to bring the sheep in. These days, many farmers use technology and data to get weather predictions.

Will it rain? You could ask the cows, or check the weather station. (Cows image by Holgi ("https://pixabay.com/photos/cattle-australia-victoria-landscape-63729/"). Weather station image by (image by Ché Lydia Xyang ("https://commons.wikimedia.org/wiki/File:Rosalie_Park_weather_station_-_panoramio.jpg"), CC-3.0.)

We can use technology to collect data about our plants. Why? Well, with good data, we can make good decisions! If the data tells us that the plants are getting too much water, then we can reduce the *volume* of water (or *how often* we water them), giving the plants the best chances of survival.

Humans generally use their senses (sight, hearing, smell, taste, touch) to collect data and make decisions. If you feel hot when you go out, then that's your skin collecting the data and sending it to your brain, helping you make the decision to wear sunscreen and a hat!

The micro:bit doesn't have the *senses* we do, but it does have some cool *sensors* — these can collect information about the world, and send it back to the micro:bit's 'brain', it's central processor unit (CPU).

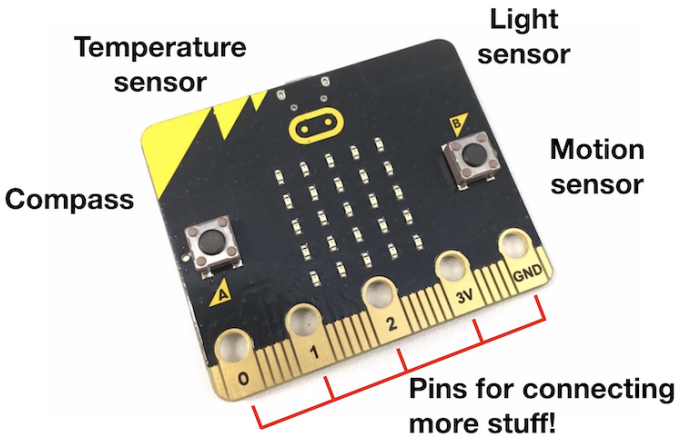
We'll be using these sensors to collect data about the health of our plants.

2.1.3. micro:bit the mega-sensor!

The micro:bit has a bunch of built-in sensors that are really handy for measuring the world around us.

It can measure the temperature, how bright the light is, whether it's moving around or staying still — it even has a built-in compass!

Plus, you can connect things to the micro:bit through the connection points — called **pins** — along the bottom edge.

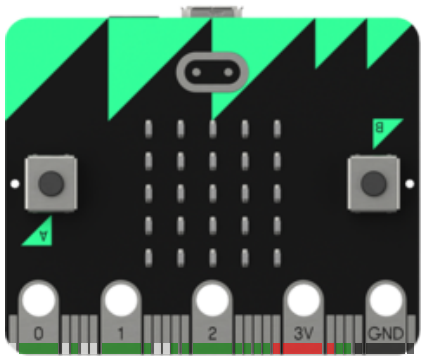


We can use a few of these sensors to help look after our plants. First, we're going to find out how to make a program to measure the *temperature*.

2.2. Temperature sensor

2.2.1. What's the temperature?

The micro:bit's built-in temperature sensor is really easy to use — you just need the temperature block. You put it inside a scroll number block like this:



The sensor tells you the temperature in **degrees celsius**, °C. (In this example, the temperature is *always* 28°C!)

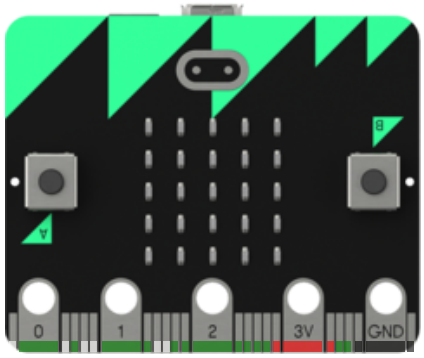
Scrolling Numbers

The temperature sensor gives the temperature as a number, not a string. Numbers and strings are treated differently by computers, so you need to use a scroll number block instead of a scroll block.

2.2.2. Problem: The temperature is ...

The temperature block just gives you a number — which is boring!

Our Smart Garden should be a bit fancier, so write a program that shows the temperature in a nicer way. For example, if the temperature is 28°C, the micro:bit scrolls `T: 28 deg` across the display like this:



1. First, scroll " T: " on the micro:bit.
2. Then scroll the temperature
3. Finally, scroll " deg " .

Remember to ★ Mark your program!

> **Temperature is a number!**
>
> So remember to use a scroll number block!)

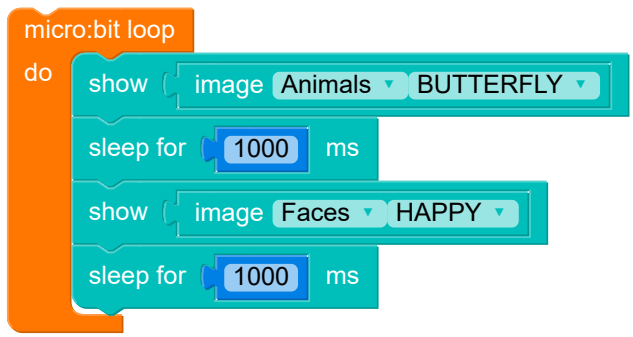
Testing

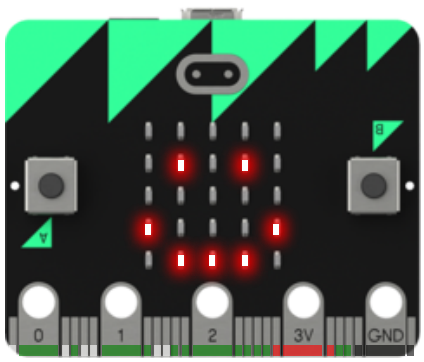
- [ ] Testing that an `T:` scrolls past.
- [ ] Testing that your program scrolls the temperature.
- [ ] Testing that `deg` scrolls past.

2.2.3. Looping forever

So far, our programs make the micro:bit do something, and then stop. But what if we want the micro:bit to keep running forever?

We can use a micro:bit loop block to repeat code over and over. In the example below, the images will keep repeating until you stop it:





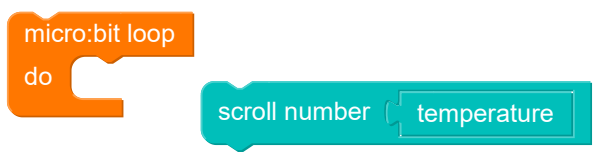
You have to click ■ to stop!

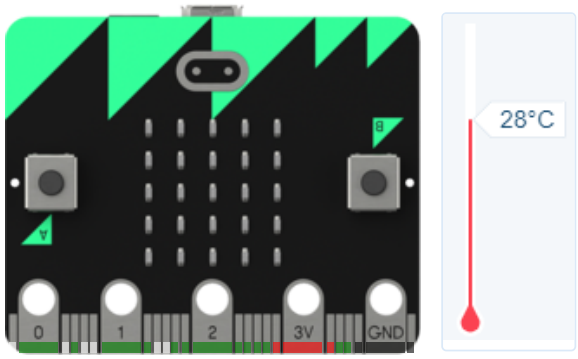
Don't forget to use the ■ button to turn off the micro:bit, otherwise it will keep running forever.

2.2.4. Weather update!

Now we can make a program that *keeps checking* the temperature!

In the example below, drag the scroll number block inside the micro:bit loop block, and ▶ run the program:





The micro:bit keeps scrolling the temperature until you tell it to stop!

Change the temperature!

See the temperature slider next to the micro:bit in the example above? When the program is running, you can drag the slider up and down to change the temperature, and see the number change on the micro:bit's display.

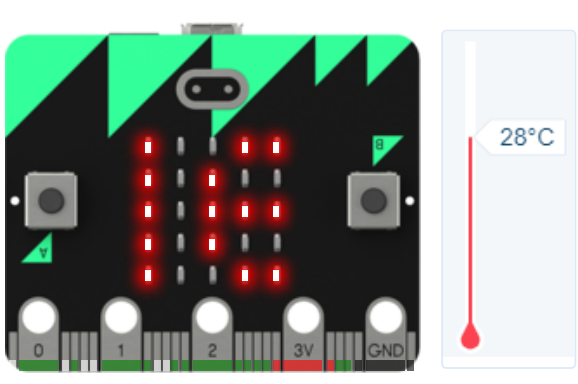
2.2.5. Problem: What's the temperature NOW?

We want our Smart Garden to measure the temperature, and keep updating it until we tell it to stop.

Write a program that scrolls the temperature nicely — so if it is 28°C, the display scrolls `T: 28 deg` — and repeats that until you stop it:

1. Get a `loop` block
2. Put a `scroll` block inside to display `" T: "`
3. Then put a `scroll number` block to show the `temperature`
4. Finally, put another `scroll` block to display `" deg "`

Your program should run like this:



Testing

- [ ] Testing that an `T:` scrolls past.
- [ ] Testing that your program scrolls the temperature.
- [ ] Testing that `deg` scrolls past.
- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display goes back to scrolling T.
- [ ] Testing that the animation loops continuously.

2.3. Soil Moisture sensor

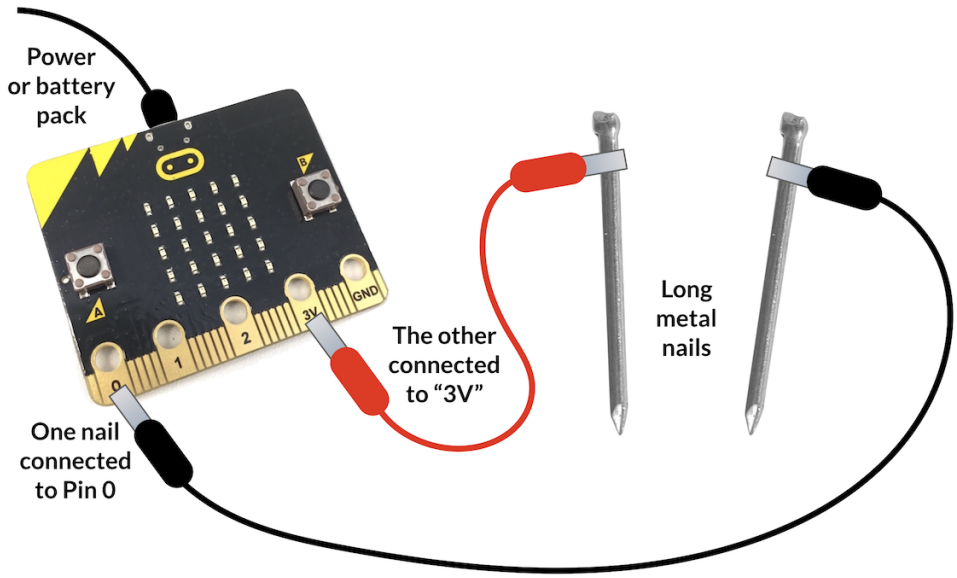
2.3.1. Hooking up the probes

OK, you've worked out how to get the micro:bit to measure temperature. Now it's time to make a soil moisture sensor.

Yes — *make!* The micro:bit doesn't have a moisture sensor built in. But that's OK, it's easy to make one yourself. You'll need:

- two metal probes to stick into the soil — good, thick metal nails will do the job!
- two leads with crocodile clips on each end (some people call them alligator clips).

Use the leads to connect the nails to the gold pins along the bottom edge of the micro:bit — you can see in the picture below how to connect your two probes:



How to hook up the soil moisture probes.

That's all you have to do! Now, to write the code ...

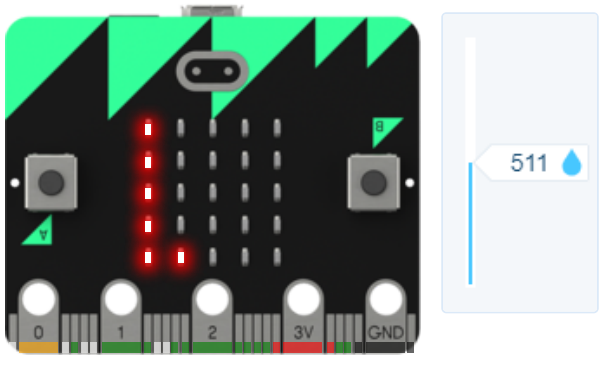
2.3.2. Measuring moisture

Soil moisture sensors work by sending some electricity through the soil from one probe to the other.

- If there is **lots of moisture** in the soil, the electricity can flow easily,
- but if the soil is **dry**, it's hard for electricity to flow.

With the nail probes connected to the pins, the micro:bit sends electricity through the soil from the `3V` pin, and then we *read* how much electricity we get at `pin0` with a read analog pin0 block.

Run ▶ the example below, and try moving the slider on the moisture scale next to the micro:bit:



The micro:bit keeps measuring and displaying the moisture level until you tell it to stop.

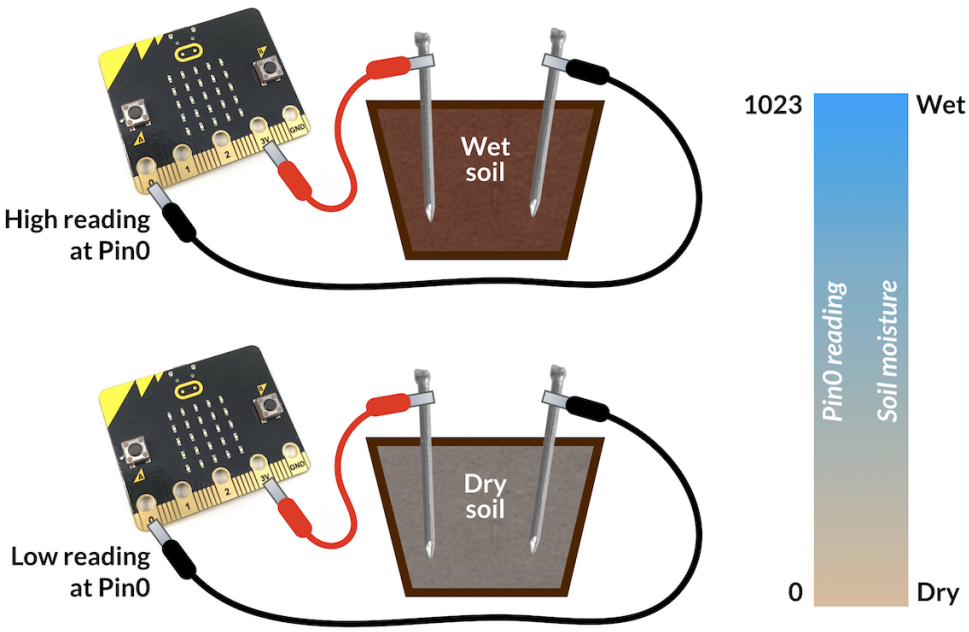
But hold on — the sensor is displaying numbers, but what do the numbers mean??

2.3.3. Wet and dry

Our micro:bit doesn't know it's measuring soil moisture — it just knows it's reading some electricity at `pin0`!

The read analog block measures this electricity as a number from 1 to 1023. So we need to work out what these numbers *mean*.

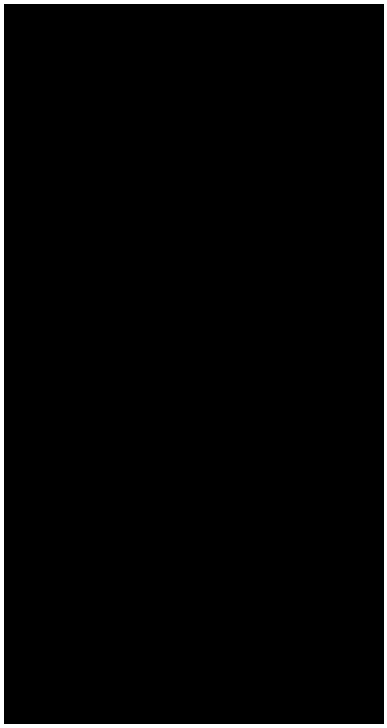
- **High numbers** mean `pin0` is reading lots of electricity. That happens when the soil is very **wet**.
- **Low numbers** mean `pin0` is reading very little electricity, which means the soil is **dry**.



Pin0 readings for wet and dry soil.

2.3.4. Wet and dry video

Here's video showing the Smart Garden moisture sensor at work, with dry soil and wet soil.



Dry soil, low readings. Wet soil, high readings. (https://groklearning-cdn.com/modules/yprjyPs3K8HUAk3wPWZwcR/SmartGardenWetDry3.mp4).

You can see that with the dry soil, the sensor reading is around 300, and for the wet soil it is over 1000.

So higher numbers mean more moisture!

2.3.5. Problem: Moisture sensor

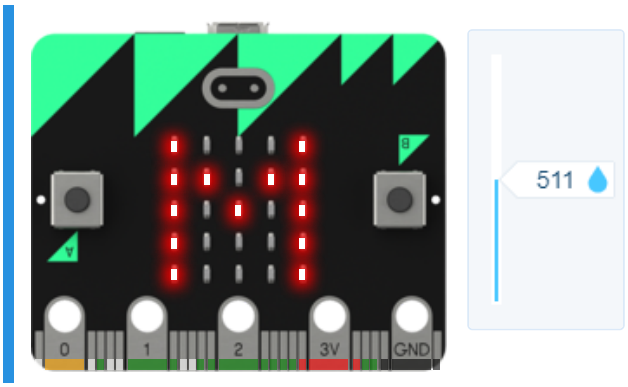
Your Smart Garden device needs to measure how wet the soil is, and to *keep measuring* until you tell it to stop.

Write a program that measures the soil moisture, displays it in an elegant way, and repeats until you stop it.

For example, if the moisture reading is 759, then the display should scroll `M: 759`.

1. Get a `micro:bit loop` block
2. Put a `scroll` block inside the loop, and set it to scroll `" M: "`
3. Put a `scroll number` block inside the loop
4. Then put a `read analog` block inside the scroll number block, and make sure it is set to **`Pin0`**

Your program should run like this:



Testing

- [ ] Testing that the display starts with M:.
- [ ] Testing that the display scrolls the moisture reading.
- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display goes back to scroll M:.
- [ ] Testing that the display scrolls a range of moisture readings.

2.4. Doing investigations

2.4.1. Science and measurement

If you were going to do a *scientific investigation* on plant growth, you might wonder:

- How does changing the *temperature* affect the way plants grow?
- How does changing the amount of *water* I'm giving them affect their growth?

Now you can make a micro:bit measure the temperature, as well as hook up a moisture sensor — so you could use your micro:bit to help you do your investigation!

2.4.2. Change one thing only

Let's say you have some plants. Some of them you water every day, and keep them in a nice warm spot. And some of them you water only once a week, and you keep them in a colder part of the room.

After a week, the first bunch of plants are growing well, but the second bunch don't seem very happy.

Was it the regular watering that helped the plants to grow? Or the temperature? Or both??

You don't know. So this is really important:

If you're doing an investigation you need to only change one thing between the two groups of plants, and keep everything else the same.

2.4.3. ... keep everything else the same!

If you decide to change the *temperature* for one bunch of plants, you need to make sure that everything else is the same for all the plants in your investigation.

They all need the same amount of water each day, the same amount of sunlight, the same kind of soil, ... *everything you can think of*.

That way, if you do see a difference in the way some of the plants are growing, you can be pretty sure it was because of the different temperatures — because that was the *only* thing that was different!

2.4.4. Problem: Plant investigation

You're interested in seeing how plants grow with different amounts of sunlight. You have two sets of plants, all the same type, in the same kind of soil. Which of the following investigations would allow you to test how sunlight affects plant growth?

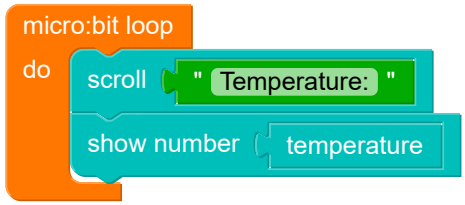
- ○ Give all the plants plenty of sunlight each day, because plants need sunlight to grow properly.
- ○ Give half of the plants more sunlight, and the other half less sunlight, but keep everything else the same (water, soil, temperature). Then compare how well each group of plants grew.
- ○ Put one group of plants in a warm place, and the other group in a cool place, but keep everything else the same (water, soil, amount of sunshine). Then compare how well each group of plants grew.
- ○ Give all the plants lots of sun for one week, and then only a little sunlight for one week, and see how well the plants grow.

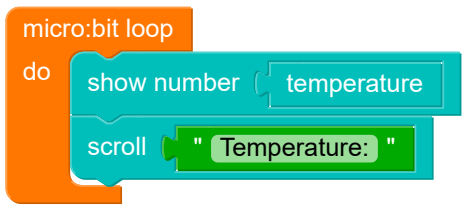
Testing

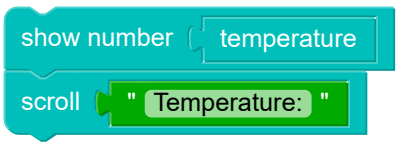
☐ That's right!

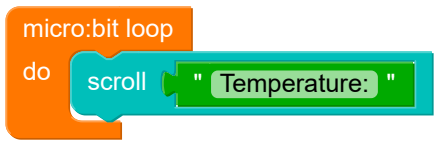
2.4.5. Problem: Measure the temperature

Which of the following code snippets would scroll `Temperature:` on the micro:bit display, and then show the temperature, and repeat that until the program is stopped?

- 

- 

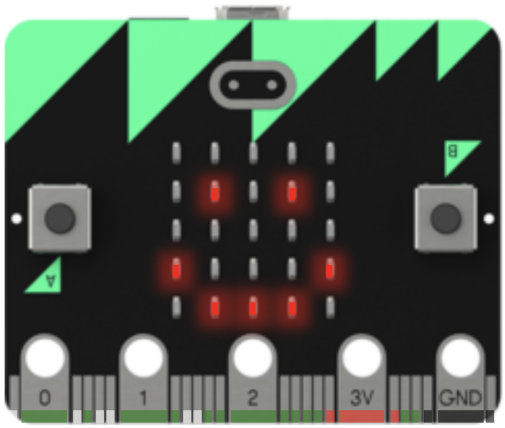
- 

- 

Testing

- [ ] That's right!

2.4.6. Keep going!



So far you've programmed the micro:bit to measure the temperature, and put together a soil moisture probe. We're making great progress — you'll have a Smart Garden in no time!

Up next we'll find out how to control the micro:bit with its built-in buttons!

BUTTONS & CUSTOM IMAGES

3.1. Controlling micro:bit with buttons

3.1.1. Take control!

So far, our micro:bit programs have been basic: show an image, scroll some words, display the temperature, that sort of thing.

To make more complicated and interesting programs, we need to get the micro:bit working a bit harder. We need to learn how to tell the micro:bit to do what we want, when we want — by using its built-in buttons!

By the end of this part of the challenge, you'll have mastered:

- making decisions in your programs with `if` and `if/else` blocks
- using the micro:bit buttons to do different things in your programs
- displaying custom images on the micro:bit display

3.2. Making decisions with buttons

3.2.1. Button A and Button B

The BBC micro:bit has two buttons, labelled A and B.

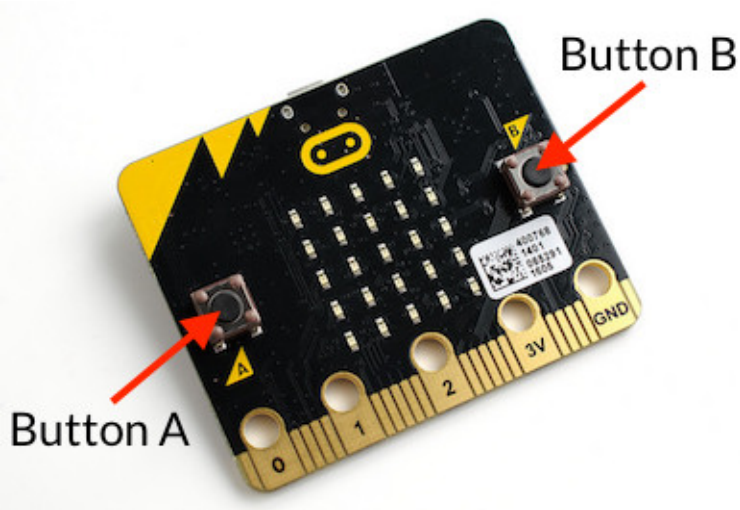


Photo by Gareth Halfacree, CC BY-SA 2.0.

Using these buttons, we can make the micro:bit do different things when we press the different buttons.

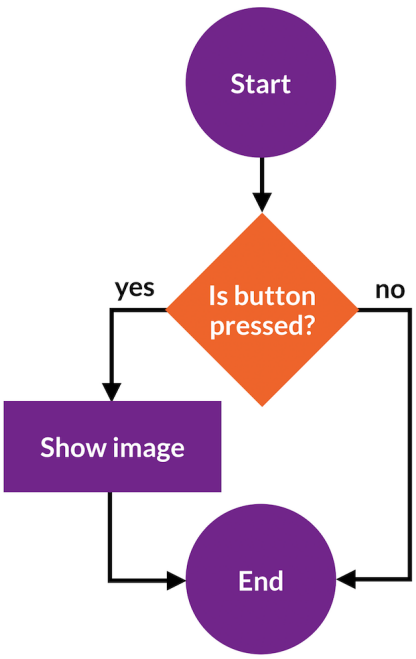
The button A is pressed block can be used to see if a button is currently pressed.

3.2.2. Making decisions

All our programs so far have shown images, or measured the temperature and scrolled it on the display. We call this "output" — and our programs have done the same thing every time we run them.

We also want our programs to react to things — for example, pressing a button. We call this "input".

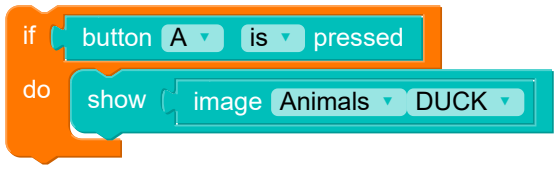
This flowchart describes a process (or *algorithm*) that makes the program run differently if a button is pressed:



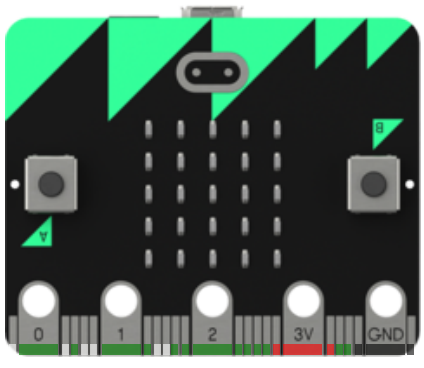
The diamond requires a **yes** or **no** decision — the answer determines which line we follow. If the answer is **yes**, we do the extra step of showing the image. If the answer is no **no**, we skip it.

3.2.3. Using the `if` block

We can use an `if` block to make the decision in the orange diamond of the flowchart: *is button A being pressed?*



Try running this program and then pressing Button A.



It doesn't work! Why not?!

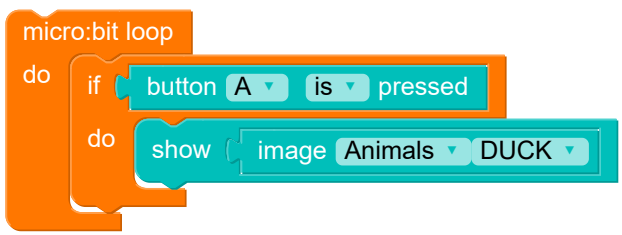
Because the code runs too fast — the program ends before we can press the button!

We need to *keep checking* whether the button is pressed ...

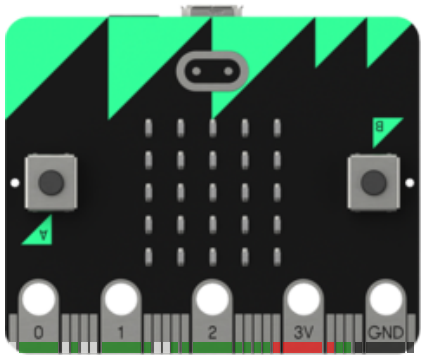
> **Use your mouse or keyboard**
>
> Press the buttons in the examples either by clicking with your mouse, or by pressing **A** or **B** on your keyboard.

3.2.4. Decisions inside the loop

We fix this by putting the `if` block *inside* a `loop`:



Try running this example, and pressing button A (or the **A** on your keyboard):

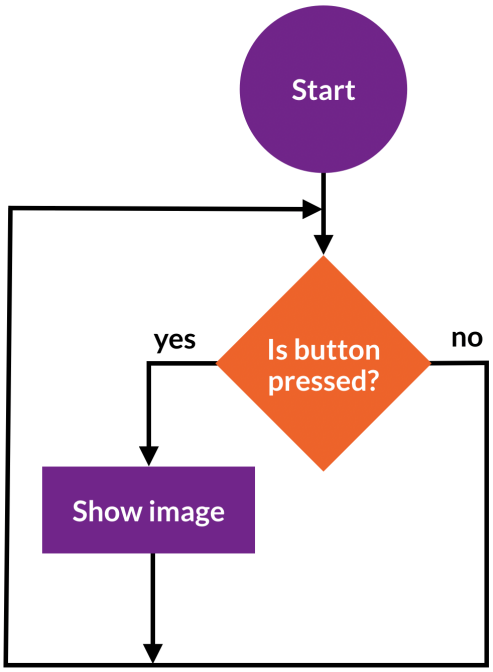


Now it works! *Whew!*

The `if` block only runs the `show` block *if* Button A is pressed — and the loop keeps running, around and around, checking if Button A is pressed or not.

3.2.5. Decision loop flowchart

Here's a flowchart for a decision inside a loop:



See how it works? Follow the arrows around the loop:

- Is the button pressed?
- **Yes!** OK, show the image.
- **No!** OK, no image then.
- ... and around the loop we go!
    - Is the button pressed?
    - **Yes!** OK, show the image.
    - **No!** OK, no image then.
    - ... and around the loop we go!
        - Is the button pressed?
        - ...

(... you get the idea.)

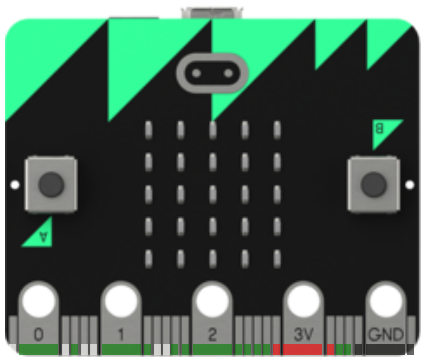
3.2.6. Problem: Rain, rain, go away!

Rain, rain, go away ... or at least let me get my umbrella!

Write a program to scroll `Rain, GO!` when button **A** is pressed.

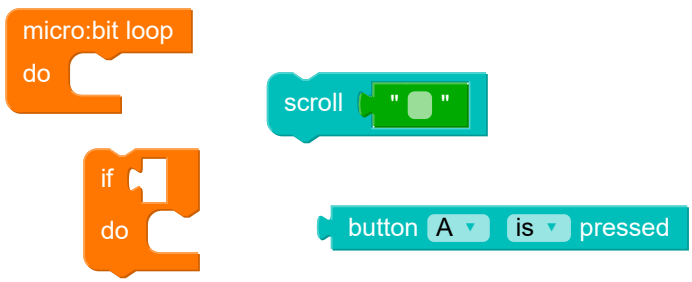
1. Make an `if` block, so that if the **A** button is pressed, then `scroll` the words `Rain, GO!`
2. Put the `if` inside a `loop` so that it checks if the button is pressed forever!

Here's what your code should do — in the example below, click on ▶ and press the **A** button to try it out.



You'll need

program.blockly



Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off being blank.
- [ ] Testing that the display scrolls correct words down when the A button is pressed.
- [ ] Testing that it went back to a blank screen afterwards.
- [ ] Testing that it continues to work multiple times.

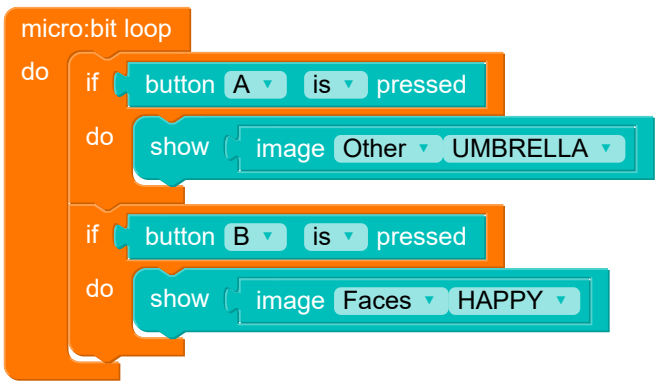
3.3. More decisions!

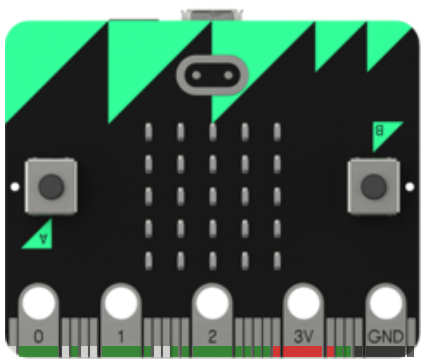
3.3.1. What if s?

The micro:bit has more than one button.

We can use more than one if !

Run ▶ the example below, and try pressing the **A** and **B** buttons!





You can change the micro:bit images using buttons! ☂ 😀

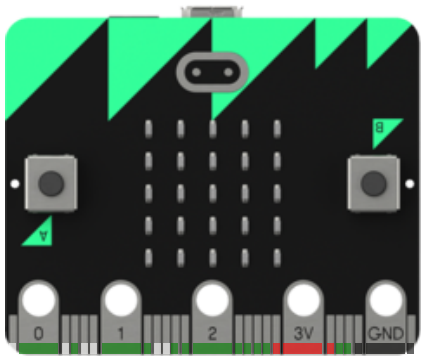
3.3.2. Problem: Sun or rain?

Will it be sunny today, or pour with rain?

Make your own weather predictor: write a program to scroll `Sun!` if button **A** is pressed, and `Rain!` if **B** is pressed.

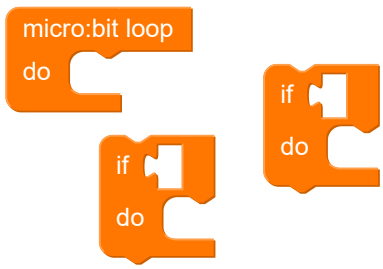
You'll need a loop block, and two if blocks — so we've made sure you have those. The rest is up to you!

Run this example to see how your code should work — try pressing **A** and **B**:



You'll need

program.blockly



Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off blank.
- [ ] Testing that it scrolls "Sun!" when the A button is pressed.
- [ ] Testing that it scrolls "Rain!" when the B button is pressed.
- [ ] Testing that it scrolls "Sun!" then "Rain!" when the A button then the B button is pressed.
- [ ] Testing that it continues to work multiple times.

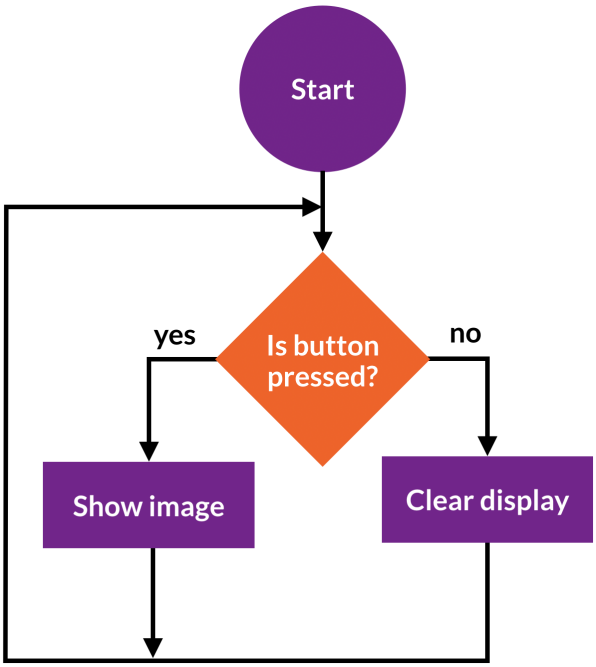
3.3.3. Decisions with two options

So far, we've asked if the button is pressed, and if the answer is "yes", we have displayed an image. But we didn't actually **do** anything if the answer was "no", the loop just started again.

Sometimes when we make a decision, we might care about **both** answers. If we ask "Is the button pressed?" we could:

- **Show** an image if the answer is "yes".
- **Hide** the image (clear the display) if the answer is "no".

That's the decision shown in this flowchart:

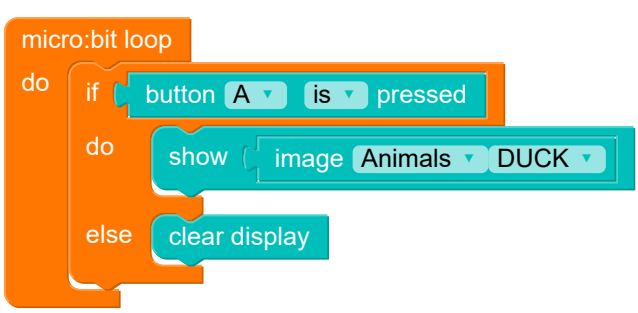


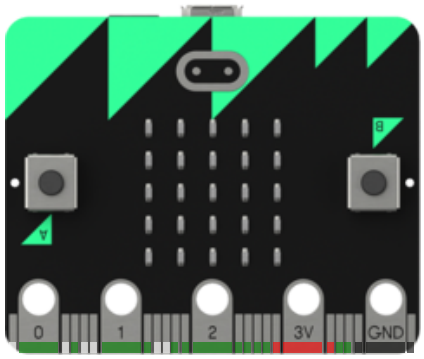
3.3.4. The if/else block

For decisions with two options we use the if/else block.

Run ▶ the example below, and then press the A button.

What happens if you press A again? Or hold down A?





When **A** is pressed, the image appears. When it is *not* pressed, the image goes away because the program runs clear display .

3.3.5. Problem: Smile for the camera!

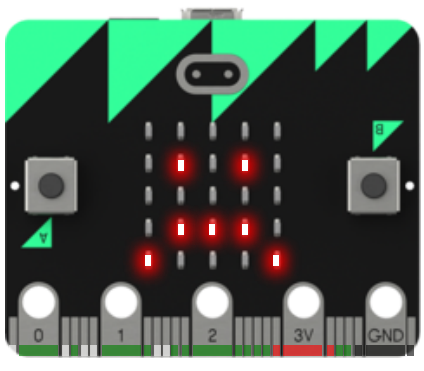
Smile for the camera! 📸**click!**😀

Let's pretend the micro:bit is a camera, and Button A takes your photo. You have to smile when the button is pressed ... but when it's not pressed, you don't have to.

Write a program that shows a happy face `image Faces HAPPY` while Button A is pressed, and a sad face `image Faces SAD` when no button is pressed.

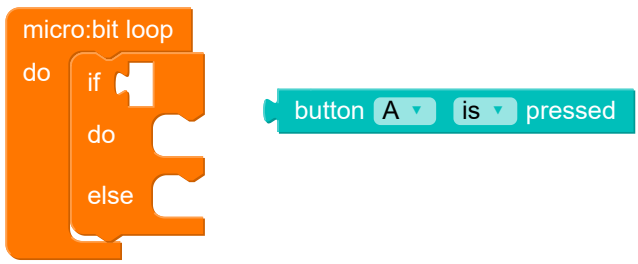
We've given you a start — your job is to fill in the gaps in the `if` and `else` blocks to show the correct images.

Here's what it should look like — click ▶ to run and then try pressing the A button:



You'll need

program.blockly



Testing

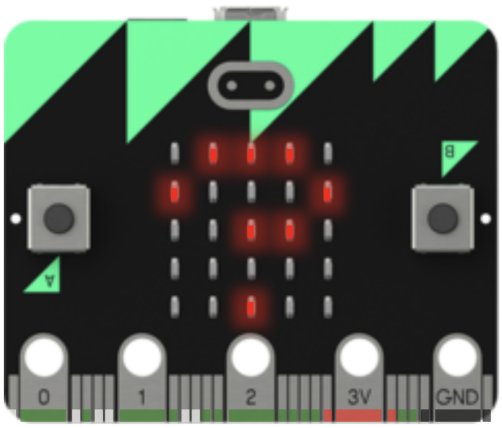
- [ ] Checking that the display starts off showing a sad face.
- [ ] Checking that it becomes happy.
- [ ] Expected that it went back to a sad face after the button was released.
- [ ] Testing that it continues to work multiple times.

3.4. BYO images

3.4.1. LEDs on display

So far, every time we've used an image, it's been from micro:bit's collection of built-in images — like the duck, the giraffe, the heart, the smiley-face.

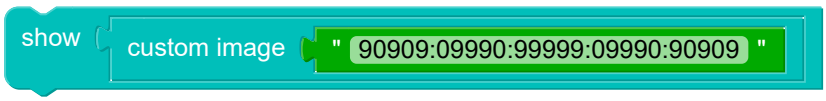
What if you want to show something that isn't in the micro:bit's list of images? What if you want to make your *own* image? 🤔

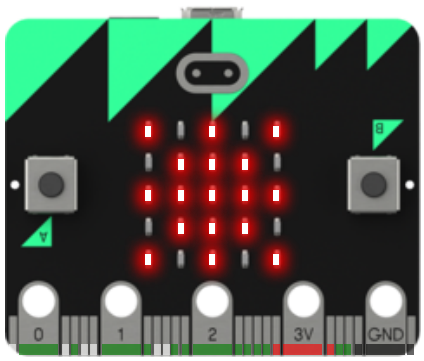


3.4.2. Five by five

It's easy to make your own images for the micro:bit. You just need to tell it which lights on the display to turn on.

The display is a 5-by-5 grid of LEDs, and we can tell the micro:bit to turn them on and off as we like using the custom image block — here's an example, click ▶ to see what the custom image is:





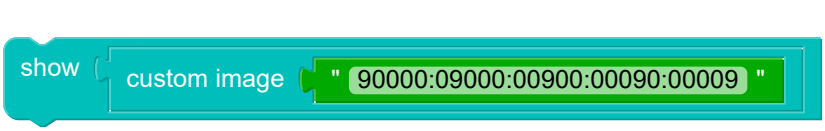
3.4.3. Make your own image

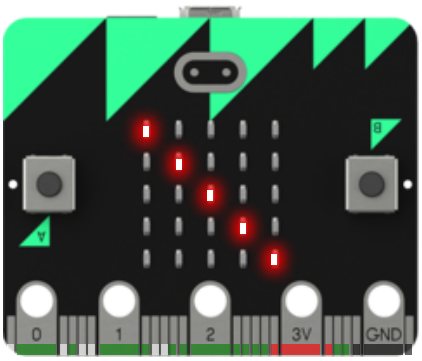
In the custom image block, we use 25 numbers to control the 25 LEDs. Each number tells an LED how bright it should be:

- a **0** means the LED is **off**
- numbers from **1** to **8** make the LED brighter and brighter
- a **9** turns the LED is on as bright as it can go

For each **row** of five LEDs we write five numbers, followed by a colon (:) — and so we end up with five lots of five numbers, like this:
`90000:09000:00900:00090:00009`

Run ▶ the example below to see what image those numbers make:





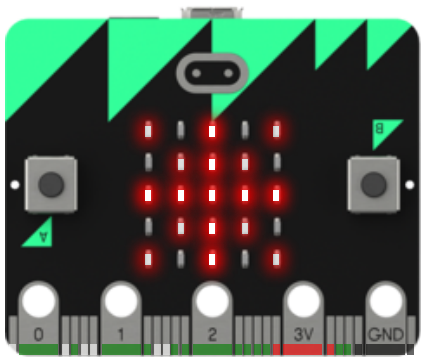
It turns on the first LED in the first row, the second LED in the second row, the third LED in the third row ... you get the idea?

What image do you get if you change the numbers in the example above to:
`11111:33333:55555:77777:99999`?
What about
`00000:09090:00000:90009:09990`?
Copy those into the example and run to see if you guessed right!

3.4.4. Problem: A bit of sunshine

The sun is coming out! 🌞😎

Create your own custom image of the shining sun — click ▶ to see what it looks like:



The image is created from these numbers:
`60906:06960:99999:06960:60906`

1. show a custom image
2. Copy the numbers above into the custom image block

Easy, isn't it?

Testing

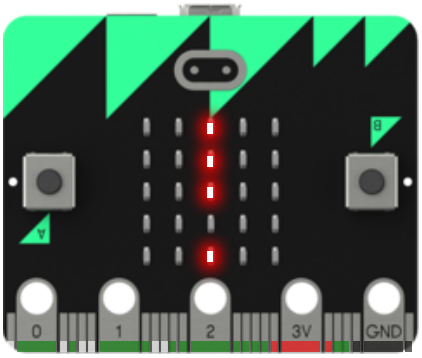
- [ ] Testing that the display is showing the sun image.

3.4.5. Custom Image Playground

Make your own images! (This part is just for messing around to see what you can come up with.)

What do you think these numbers would make in the custom image block?
`00900:00900:00900:00000:00900`

Run! ▶ the example below! to see! if you're right!!!!



Now try out your own images — change the numbers in the example to make any image you like. See if you can make a face, or an animal.

Remember you always need five groups of five numbers, with a colon (:) between them.

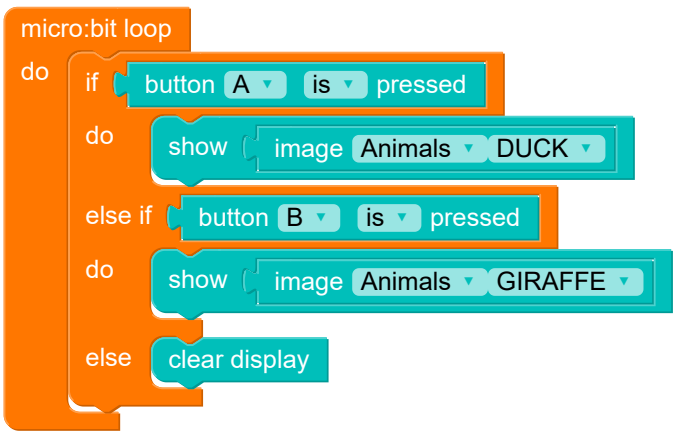
Try different brightnesses — remember **9** is bright, **5** is medium brightness, **1** is very dim, and **0** is off.

3.5. Even more decisions!

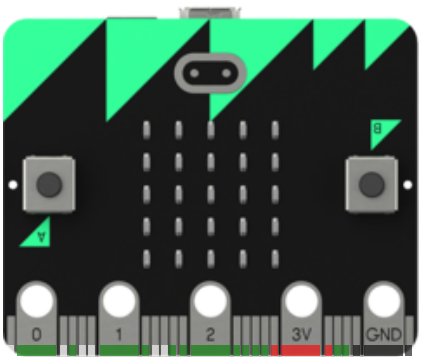
3.5.1. if else if else if else ...

Sometimes, there are lots of different possible decisions. For example, your program could show a *duck* if button A is pressed, a *giraffe* if button B is pressed ... and show nothing at all if *no* button is pressed.

We can program these complex decisions by using `if` and `else if` blocks, like this:



The `if` does button **A** and shows a duck, and `else if` does button **B** and shows a giraffe. The final `else` captures all other decisions — in this case, that's just pushing neither of the buttons! Try it out by clicking ▶ and trying out the two buttons:



3.5.2. Problem: Whatever the weather

Sunshine ☼? Or rain 💧?

Time to update your weather predictor! Write a program that will display:

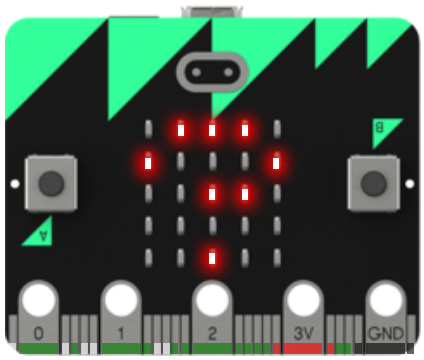
- a custom image of the sun if button **A** is pressed
- a custom image of a raindrop if button **B** is pressed, and
- a custom image of a question mark when **no** button is pressed.

We have given you a start already — your job is to put the blocks in place, and add the custom images.

The custom images are created from these numbers:

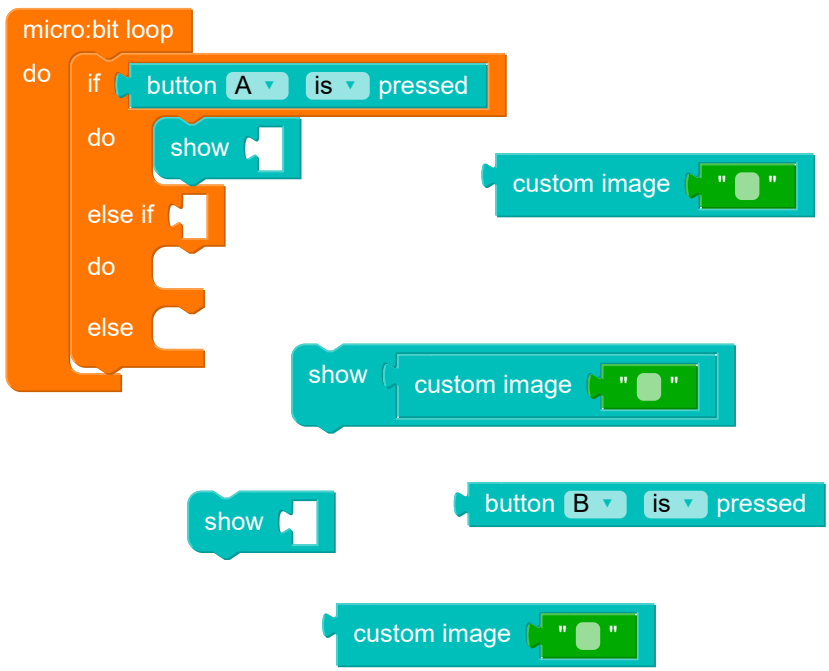
- Sun: `60906:06960:99999:06960:60906`
- Raindrop: `00600:06960:69996:69996:06960`
- Question mark: `09990:90009:00990:00000:00900`

Click ▶ to see what your program should look like — try pressing buttons **A** and **B**:



You'll need

program.blockly



Testing

- [ ] Testing that the display is showing a question mark at start.
- [ ] Testing that the display is showing the sun image when A is tapped.
- [ ] Testing that the display is showing the raindrop image when B is tapped.

3.6. Summary

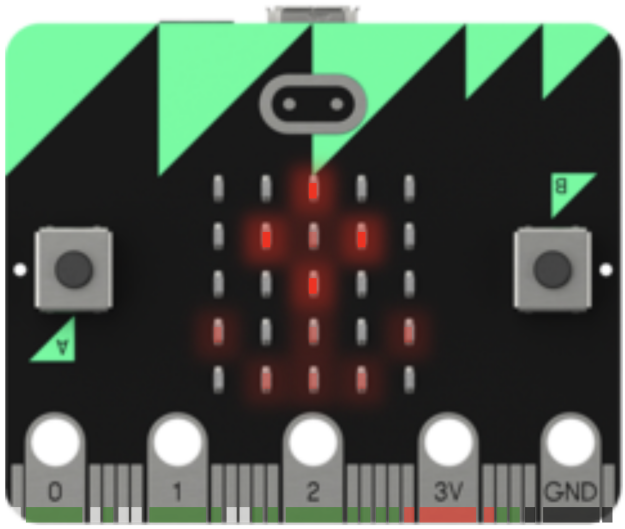
3.6.1. Problem: Tortoise and Hare

Which of the following code snippets will correctly display an image of tortoise whenever the micro:bit's button A is pressed, and an image of a hare (OK, a rabbit then) whenever button B is pressed, but no image at all when no button is pressed?

Testing

- [ ] That's right!

3.6.2. Up Next: Smart Garden!



You're ready!

You've got your temperature sensor and your soil moisture sensor. You know about loops , and making decisions with buttons and if/else blocks. You can make your own custom images .

It's time to put it all together to make your very own Smart Garden.

4

PUTTING IT ALL TOGETHER

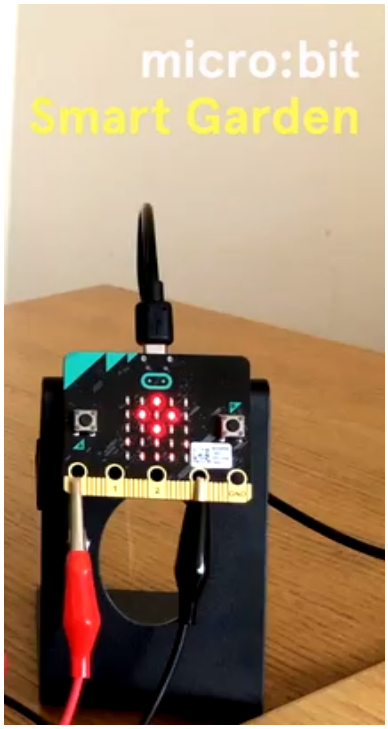
4.1. Let's make a Smart Garden

4.1.1. Piece by piece

You're ready to put together your final Smart Garden project. Here's what you're going to make in this last part of the challenge:

- First, you'll write code to measure the temperature when Button A is pressed, and display the temperature with a custom image.
- Then you'll make a program to measure soil moisture when Button B is pressed, and display it with a happy face if there is enough water, or a sad face if the soil is dry.
- Finally, you'll put it all together, with code to display a custom flower image when no button is pressed.

Here's the video again of the Smart Garden in action:



0:00 / 0:43

Your final project. (https://groklearning-cdn.com/modules/cU5geH65oKwqHZLrVZLEZg/SmartGarden4.mp4).

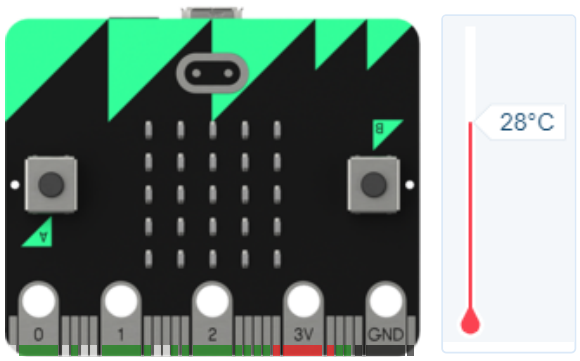
4.2. First, the temperature button

4.2.1. Problem: Temperature on A

First you'll create the temperature sensor. Build a program that, when button **A** is pressed, measures the temperature and displays it nicely with a custom sunshine image like this: `28 deg`

- When no button is pressed, the program should clear the display.
- When button **A** is pressed, it should display a custom image of the sun from the numbers `60906:06960:99999:06960:60906` for one second, then scroll the temperature reading, and then scroll `deg`

Your program should run like this — click ▶ and then press button **A**:



Hint

You're going to need a loop block and an if/else block. You'll also need to remember how to sleep !

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the screen is blank at the start.
- [ ] Testing that an image of the sun is displayed.
- [ ] Testing that the sunshine image image appears for 1 second.
- [ ] Testing that your program scrolls the temperature.
- [ ] Testing that `deg` scrolls past.
- [ ] Testing that the code works for different temperatures.

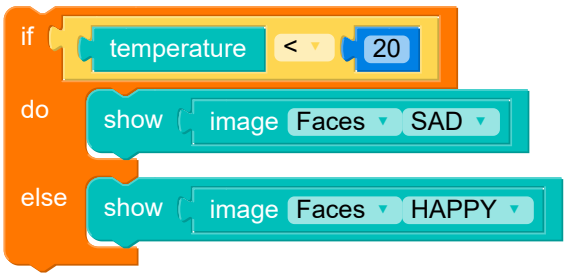
4.3. Then, the soil moisture button

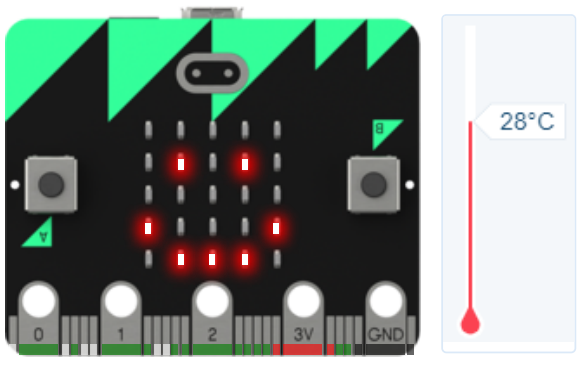
4.3.1. if comparisons

We have seen how to use if blocks using buttons to make decisions. But there are other ways to make a decision.

For example, you could make a decision based on the temperature — if it's less than 20 degrees, you're sad, but if it's 20 degrees or warmer, you're happy!

The way we do this sort of decision is with a comparison block — try out this example:





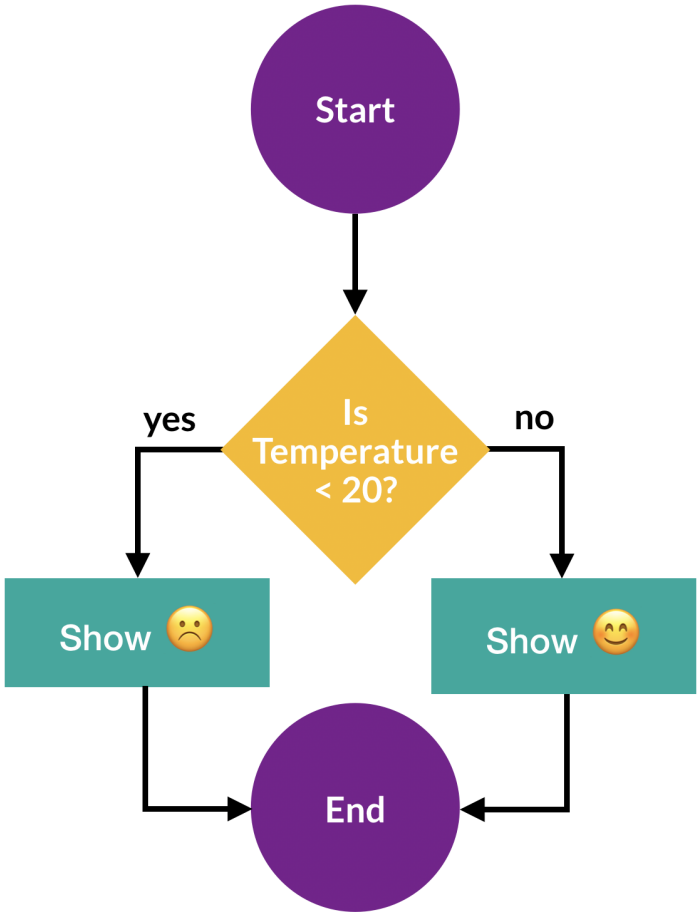
The comparison block can compare two numbers in all sorts of ways — you can make a decision when the first number is:

- greater than > the second number
- greater than or equal to ≥ the second number
- less than < the second number
- less than or equal to ≤ the second number
- equal to = the second number
- **not** equal to ≠ the second number

Try changing the example by clicking on the < and choosing a different comparison. What happens if you change the number 20 to 30 ? Give that a try, and run the example again.

4.3.2. Comparison flowchart

The flowchart for a decision that compares two numbers looks like this:


Start
Is Temperature < 20?
yes
no
Show ☹️
Show 😊
End

4.3.3. Problem: Water me, please!

We want our soil moisture sensor to tell us when the plant needs watering.

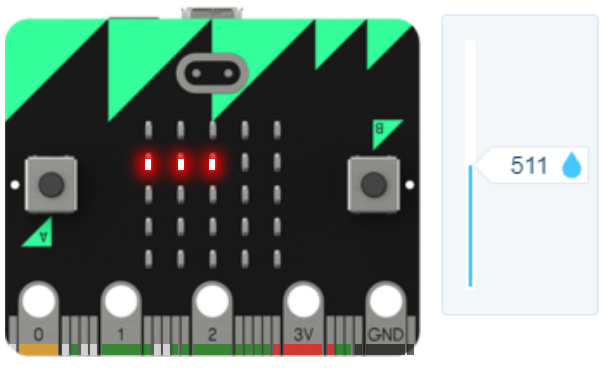
Remember that **low numbers** mean the soil is dry, and **high numbers** mean the soil is moist? Write a program to read in the moisture level from the sensor connected to `pin0`, and then:

- If the reading is `600` **or more**, the program should scroll `I'm OK!`
- **Otherwise** (when the reading is less than 600), the program should scroll `Water me!`

We've given you an `if/else` block and a `comparison` block to get you started. You also need to keep the program running with a `loop`, so you can test it with different moisture levels.

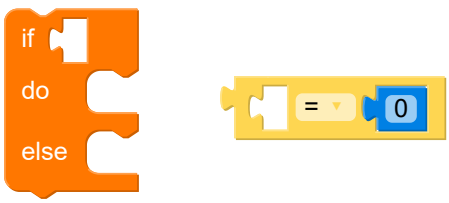
You'll need the right kind of comparison — click on the `=` to choose from the different comparisons.

Your program should run like this — click ▶ to see, and move the moisture slider up and down:



You'll need

program.blockly



Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display scrolls "Water me!" when moisture is 300.
- [ ] Testing that the display scrolls "I'm OK!" when moisture is 900.
- [ ] Testing that the display scrolls "I'm OK!" when moisture is 600.
- [ ] Testing that the animation loops continuously.

4.3.4. Problem: Moisture on B

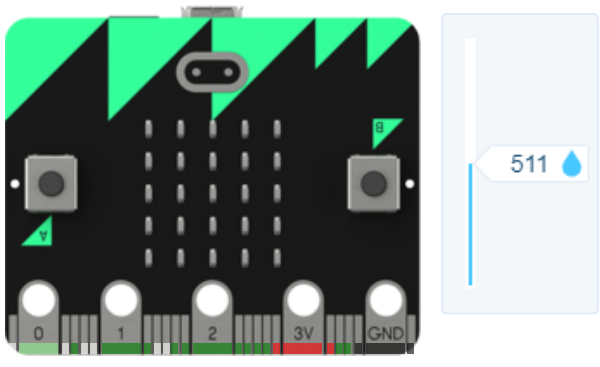
OK, time to write the full moisture sensor program.

Here's how it's going to work — when **button `B` is pressed**, do the following:

- show a custom image of a raindrop for **one second**, using the image numbers `00600:06960:69996:69996:06960`
- Make a decision using an `if/else`, checking the soil moisture reading at **`pin0`**
  - If the soil moisture is `<` `600`, the soil is dry — so show a sad face image for one second
  - Otherwise (if the soil moisture is 600 or greater), it has enough water — so display a happy face image for one second
- finally, scroll the moisture reading

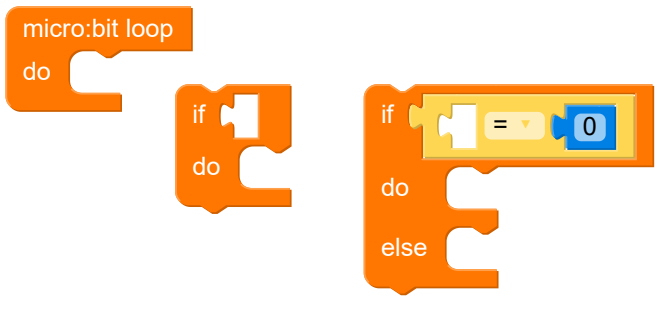
> **Hint!**
>
> This one's tricky — you'll need an `if/else` block *inside* an `if` block! We've given you both of them, your job is to work out where they go, and fill in the gaps.

Your program should run like this — click ▶ and press button `B`:



You'll need

program.blockly



Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the raindrop image image appears.
- [ ] Testing that the raindrop image image appears for 1 second.
- [ ] Testing that the HAPPY face image appears.

- [ ] Testing that the SAD face image appears for low moisture.
- [ ] Testing that the HAPPY face stays for 1 second.
- [ ] Testing that the SAD face stays for 1 second.
- [ ] Testing that your program does everything.

4.4. Hello, Smart Garden!

4.4.1. Problem: The Full Smart Garden!

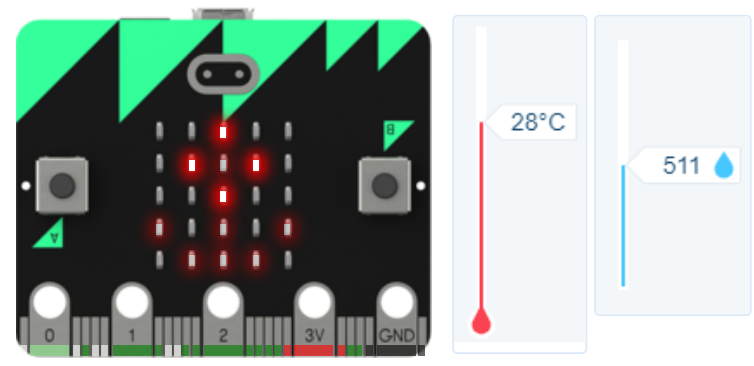
Now it's time to put *everything* together!

Write the full Smart Garden program that does the following:

- If button `A` is pressed:
  - scroll `T:`
  - then scroll the temperature reading
  - then scroll `deg`
- If button `B` is pressed:
  - display a custom image of a raindrop from the numbers `00600:06960:69996:69996:06960` for one second
  - then make a decision about how dry the soil is
    - if the soil moisture reading on `pin0` is < 600, show a sad face image for one second
    - otherwise (if moisture ≥ 600), show a happy face for one second
  - after the sad/happy face, scroll the moisture reading
- When **no button is pressed**, the micro:bit displays a custom image of a flower, using the image numbers `00900:09590:00900:50505:05550`

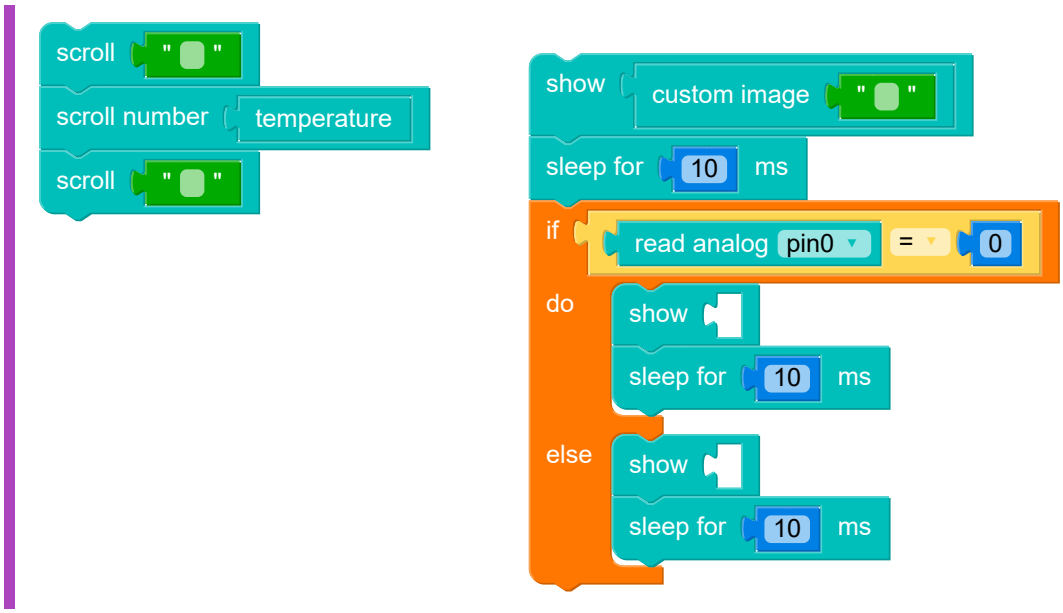
We've given you parts of the solutuions to the previous *"Temperature on A"* and *"Moisture on B"* problems. Your job is to bring everything together into one program that does it all!

Your program should run like this — click ▶, press the buttons and move the sliders:



You'll need

program.blockly

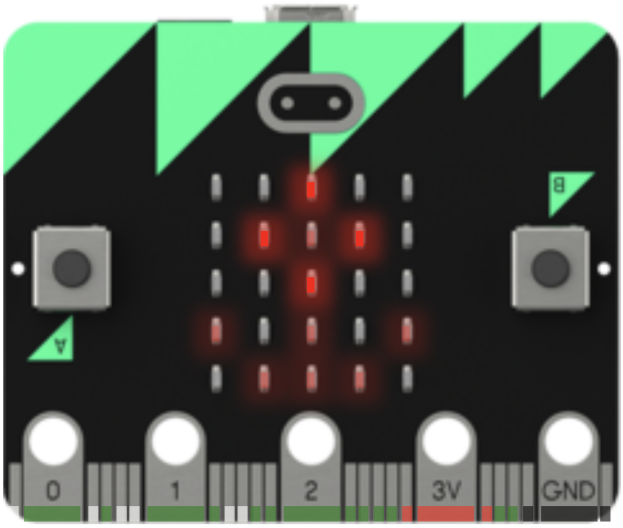


Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts with the flower image.
- [ ] Testing that the flower image stays for 1 second.
- [ ] Testing that the display scrolls T.
- [ ] Testing that the display scrolls the temperature reading.
- [ ] Testing that the display scrolls deg.
- [ ] Testing that the entire temperature reading works.
- [ ] Testing that the raindrop image appears for button B.
- [ ] Testing that the raindrop image appears for 1 second.
- [ ] Testing that the HAPPY face appears for high moisture.
- [ ] Testing that the HAPPY face stays for 1 second.
- [ ] Testing that the SAD face appears for low moisture.
- [ ] Testing that the SAD face stays for 1 second.
- [ ] Testing that the entire moisture part works.
- [ ] Testing that the entire program works.

4.5. You did it! High 5!

4.5.1. Congratulations!



You made it! You've created a Smart Garden to measure temperature and soil moisture, and along the way you have learned:

- how to show images and scroll " text " , and make your own custom images
- how to show the temperature , and connect probes to the micro:bit pins to read soil moisture
- how to use if/else and the micro:bit's buttons to make decisions
- how to compare numbers to help make decisions

Now that you have your Smart Garden, you can use it for some scientific investigating ...

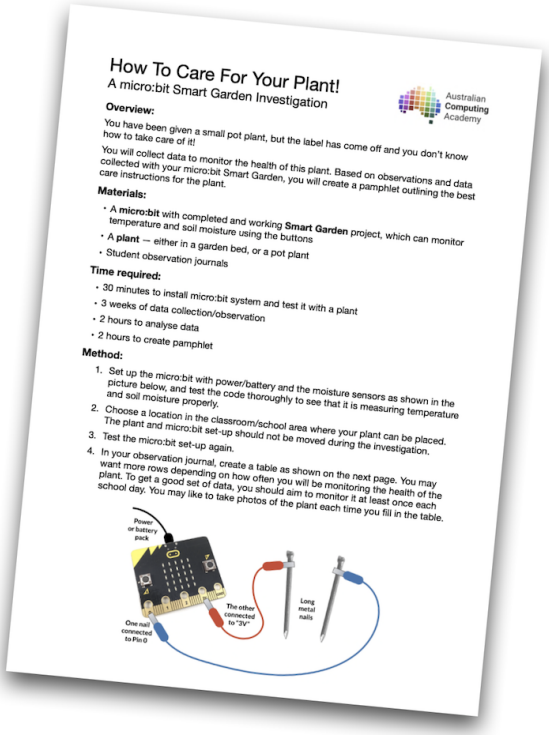
4.5.2. A Smart Garden Investigation

There are lots of different investigations you could do, using your Smart Garden to help look after your plants. You could:

- investigate how plants grow at **different temperatures**
- see whether **different amounts of water** affects how well seeds germinate
- test how changing the **amount of sunshine** that plants get affects their growth

In that last idea, for example, you could use the Smart Garden to make sure the temperature and soil moisture is kept the same for all plants in your investigation — while giving different groups of plants different amounts of sunlight. This helps you to make sure this is a *fair test*.

We have created an investigation that you might like to try out, called "How To Care For Your Plant!". You can [download the investigation worksheet here (https://groklearning-cdn.com/modules/vgxnevP8WvyWyKKQvoc2jV/ACA-SmartGardenInv-56.pdf)](https://groklearning-cdn.com/modules/vgxnevP8WvyWyKKQvoc2jV/ACA-SmartGardenInv-56.pdf).



How To Care For Your Plant!

A micro:bit Smart Garden Investigation

Australian Computing Academy

Overview:

You have been given a small pot plant, but the label has come off and you don't know how to take care of it!

You will collect data to monitor the health of this plant. Based on observations and data collected with your micro:bit Smart Garden, you will create a pamphlet outlining the best care instructions for the plant.

Materials:

- A **micro:bit** with completed and working **Smart Garden** project, which can monitor temperature and soil moisture using the buttons
- A **plant** — either in a garden bed, or a pot plant
- Student observation journals

Time required:

- 30 minutes to install micro:bit system and test it with a plant
- 3 weeks of data collection/observation
- 2 hours to analyse data
- 2 hours to create pamphlet

Method:

1. Set up the micro:bit with power/battery and the moisture sensors as shown in the picture below, and test the code thoroughly to see that it is measuring temperature and soil moisture properly.
2. Choose a location in the classroom/school area where your plant can be placed. The plant and micro:bit set-up should not be moved during the investigation.
3. Test the micro:bit set-up again.
4. In your observation journal, create a table as shown on the next page. You may want more rows depending on how often you will be monitoring the health of the plant. To get a good set of data, you should aim to monitor it at least once each school day. You may like to take photos of the plant each time you fill in the table.

(https://groklearning-

cdn.com/modules/vgxnevP8WvyWyKKQvoc2jV/ACA-SmartGardenInv-56.pdf)

How To Care For Your Plant! (https://groklearning-cdn.com/modules/vgxnevP8WvyWyKKQvoc2jV/ACA-SmartGardenInv-56.pdf)

4.5.3. Problem: Blockly micro:bit Playground

This is the Smart Garden micro:bit playground! Use the blocks to build anything you like.

No marks are awarded here — it's a place where you can just muck around and see what you can build. We've included all of the blocks you've used so far, and a bunch of new ones you may not have seen before. Explore, try out different things, make something new.

> **Save or submit your code!**
>
> There are no points to be earned for this question, so you can submit whatever code you like. Make sure you save programs that you want to keep!

You'll need

components.json

Testing

- [ ] It's a playground --- nothing to mark!