Australian
Computing
Academy

DT Challenge

Sport + micro:bit (Blockly)

1. Getting started with micro:bit
2. Loops, buttons and music
3. Simple decisions and variables
4. Accelerometer and loops
5. Numbers and pixels

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

1

GETTING STARTED WITH MICRO:BIT

1.1. Getting started

1.1.1. Welcome!

Welcome to the Health and Physical Education (HPE for short) micro:bit challenge!

Get ready for a fun challenge that will get you moving and cheering as we explore ways of being healthy and involved in our sports community using the BBC micro:bit.

We will be coding using **Blockly**.

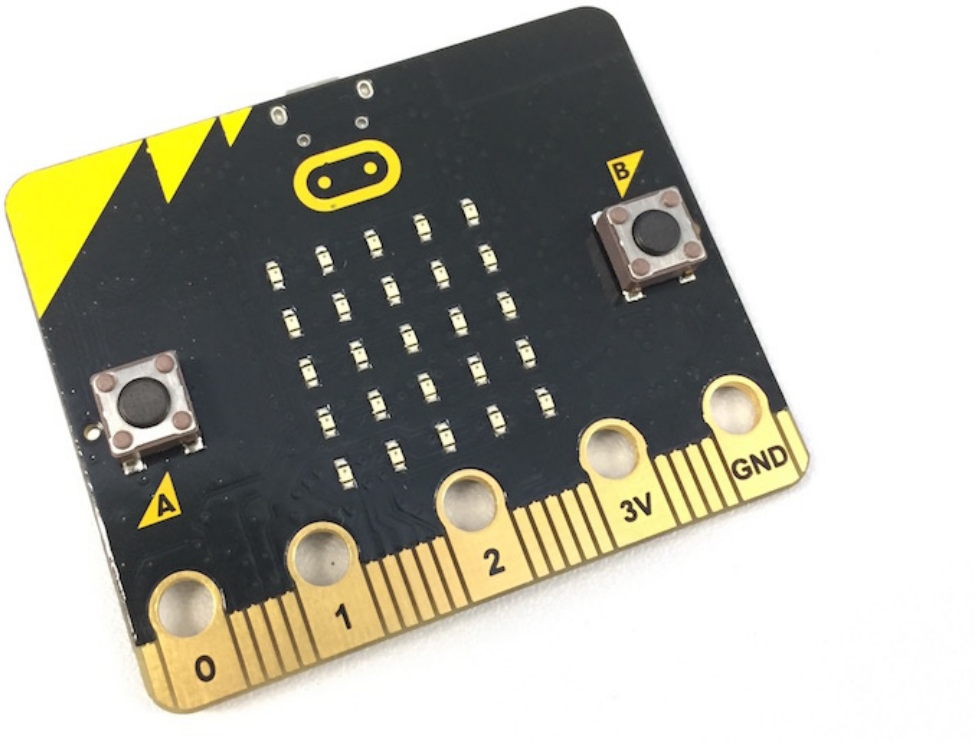
Blockly is a visual coding language, which means you will be able to drag and drop pieces of code to make your projects.

Get ready.... Get set... Go!

1.1.2. BBC micro:bit

You will be learning to code one of these! This is a 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**
- **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...**
>
> You can still do this course! We have a *simulator* which works like a real micro:bit. You will see it as we move along.

1.1.3. The goal

Over the course of this course (😉) you will learn everything you need to use your micro:bit in an egg and spoon race as the egg AND the spoon. 🥚🥄❗

You will need to keep your micro:bit balanced so that the egg doesn't fall off as you race. Here is a video that shows the "egg" (a single LED) moving around the micro:bit as it tilts.

Tilt too far and the egg falls off! 💀

0:00 / 0:07

The micro:bit egg and spoon in action!

Ready to learn some code? Let's go! 😀😀

1.1.4. Hello, micro:bit!

It's time to dive right in. Let's run your first program on the micro:bit!

There will be lots of examples as you work through the challenge.

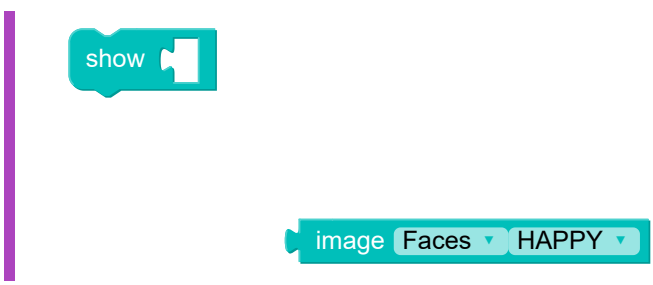
You can run the code in the examples by clicking the ▶ Run button.

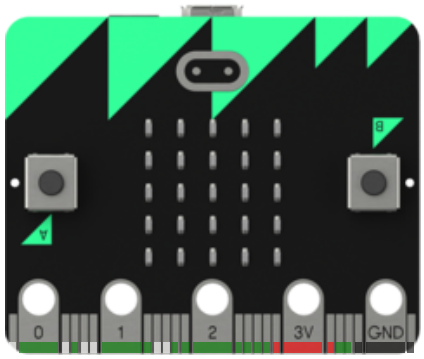
It is in the in the top right hand corner of the example.

If you move the blocks around and want to reset them, you can. Us the ↩ Reset button.

It's time to run your first example! Follow the steps below.

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





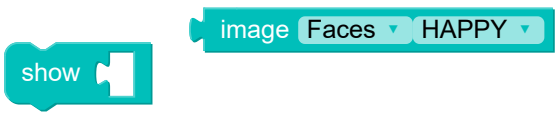
1.1.5. Change the face!

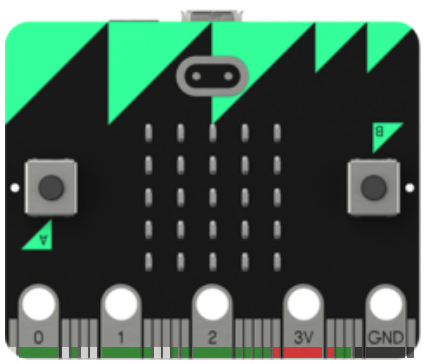
The micro:bit uses 25 Light Emitting Diodes (LEDs) to display images, numbers, and letters.

We can display some different faces and give the micro:bit some more personality.😕😎☹️🙂

Follow the steps below.

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!



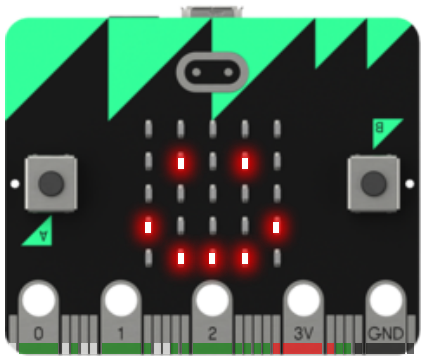


1.1.6. Problem: Happy micro:bit!

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!**

1.1.7. 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 folder in your directory.
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. Writing micro:bit programs

1.2.1. More Images

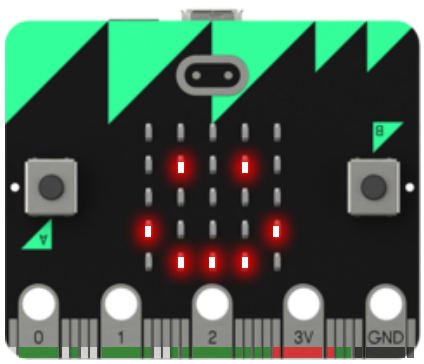
Your micro:bit has loads of images pre-programmed. Not just smiley faces. It can show shapes, music notes, arrows and animals!

You can change the **category** to find more images.

What animals can you find? What symbols can it show? ▲■

You can play with this example to explore the different options.

1. Change the category block below from Faces to anything else.



1.2.2. Block drawers

You will need an extra block to complete the next problem. The blocks are kept in drawers on the left hand side of the building space.

To add blocks to your screen, click on the micro:bit drawer.

Drag the blocks you want from the drawer onto the workspace, and you're done!

Dragging blocks from the drawer

1.2.3. Problem: Team colours

Your micro:bit is a member of their local soccer club **bit:fc** ⚽! They all wear red shirts to games. Use the blocks you have learned to help it get ready.

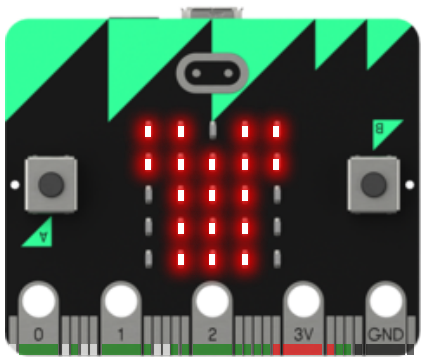
We want to show a TSHIRT for your micro:bit to wear.

1. Click on **micro:bit** and drag the show block into the workspace.
2. Join the blocks together.
3. Change the Animals block to Other
4. Choose the TSHIRT

If you can't see all the options, move your blocks around and try again.

5. ▶ Run and ★ Mark your program.

Your micro:bit t-shirt will look like this (click ▶ to play it):



You'll need

program.blockly

Testing

- [ ] Testing that the display is showing a shirt.
- [ ] **Congratulations, your micro:bit looks great!** 👍💪

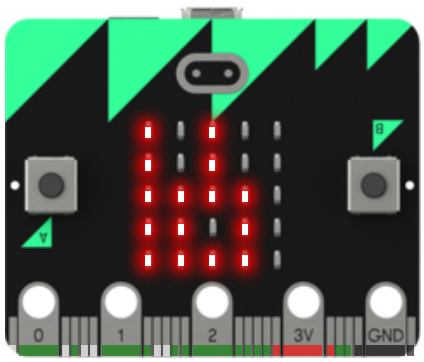
1.2.4. Sports teams

Joining a sports club is a great way to stay active and healthy. And they keep you connected with your community! Your school might have different sports teams. You might even have a local netball, tennis or football club for your suburb.

Being part of a club means being part of a community which is great for your sense of health and wellbeing.

Your micro:bit is a member of bit:fc, the local soccer club.

Their mascot is a rabbit called Bit Bunny 🐰.



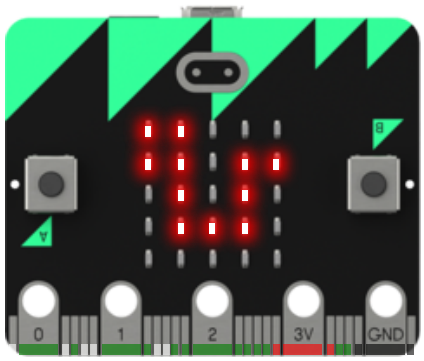
1.2.5. Problem: My micro:bit mascot!

This is Phillie Phanatic. He is the mascot for the Philadelphia Phillies Major League Baseball team in America.

He is here to inspire you because it's your turn to choose a mascot! You can choose **any animal** to be the mascot for your own team!

1. Get the show and image blocks and connect them.
2. Choose your mascot animal. It can be any **animal**.
3. Run your program and mark it!

Here's a snake mascot example:



Testing

- [ ] Testing that the display is showing one of the seven animals.
- [ ] **Well done, you've made your own team mascot!**

1.3. Animation and Sleep

1.3.1. Showing 2 images

If we want to show more than one image using the micro:bit, how do we do that?

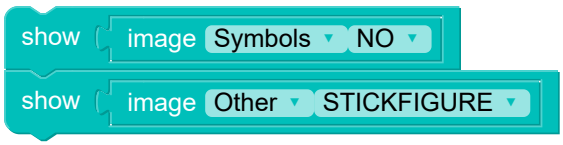
For example, if we want to make a micro:bit pedestrian crossing 🚦, we will need 2 images.

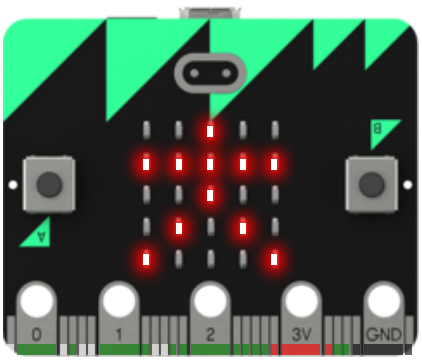
- Symbols NO to tell people to stop 🛑 and wait
- Other STICKFIGURE to tell people they can safely cross 🚶🚶

The example below is trying to do this, but if we run the program we only see the STICKFIGURE.

1. Try it for yourself.

> **It doesn't work!** 🤖
>
> The micro:bit is **really** fast. It does show the NO image but only for a fraction of a second. Much too fast for us to see. We will learn how to fix this in the next slide.



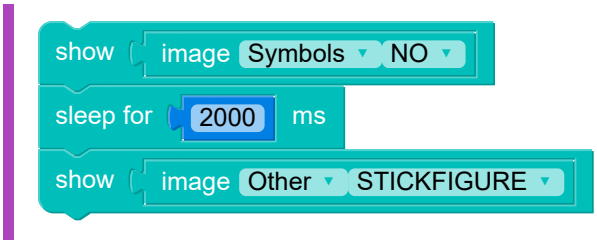


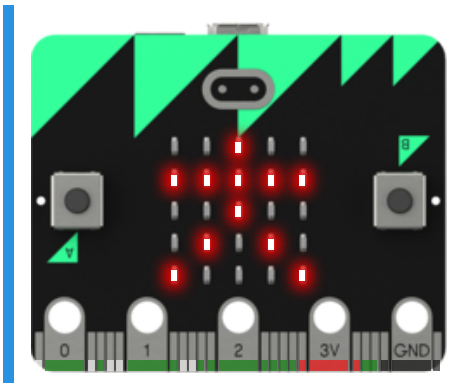
1.3.2. The sleep block

We can stop the micro:bit going too fast using sleep . You set how long the micro:bit will sleep for in **milliseconds (ms).**

There are 1000 milliseconds in 1 second.

1. Click ▶ run.
2. Now, the Symbols NO shows for 2 seconds. Then Other STICKFIGURE shows.


show
image
Symbols
NO
sleep for
2000
ms
show
image
Other
STICKFIGURE


A
B
0
1
2
3V
GND

1.3.3. Problem: Wake up time

It's time to wake up!

Write a program that will:

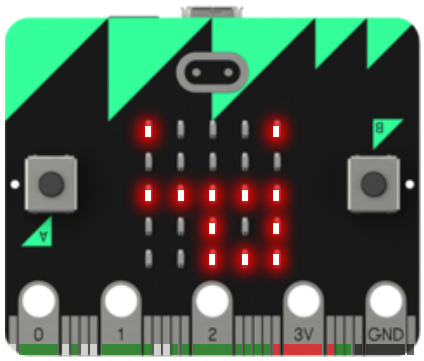
- show an ASLEEP face for **three seconds**
- then show a SILLY face

We have started the code for you. Now you need to follow the steps.

1. Drag the extra blocks you need from the workspace.
2. Join the blocks together.
3. Change the sleep amount from 1000 to 3000 .

   (1 second is 1000 **milliseconds or ms**. That means 3 seconds is 3000 **ms**.)
4. Show the SILLY face after the sleep block.

Here's an example:



You'll need

program.blockly

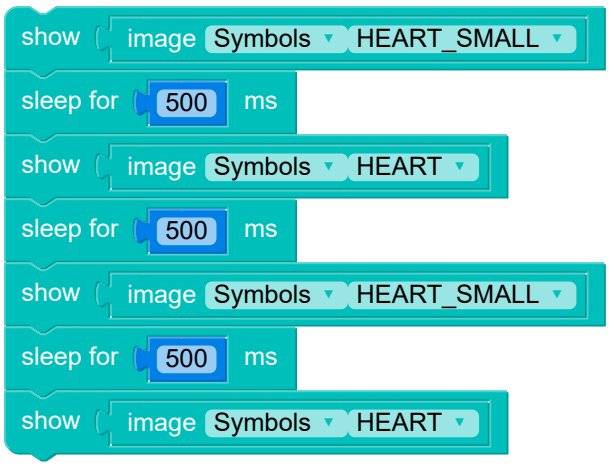
Testing

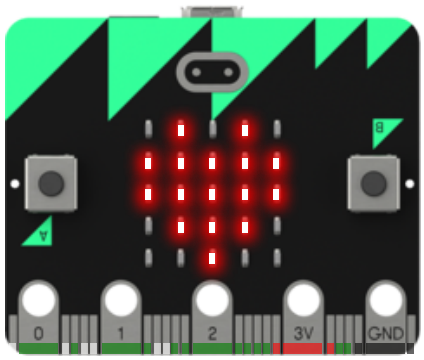
- [ ] Testing that the display starts with an asleep face.
- [ ] Testing that the asleep face is still on the screen less than 1 second later.
- [ ] Testing that the display changes to a silly face after 3 seconds.
- [ ] Testing that the silly face stays on the display for 1.5 seconds.
- [ ] **Congratulations! Your micro:bit is awake.** 😊

1.3.4. Healthy hearts

We all want a healthy heart beat, and we can make a heart animation using the micro:bit. We use the `show` and `sleep` blocks we have learned about.

Run the example below.





You can see how fast the micro:bit's heart is beating! 😍

Measuring how fast your heart is beating (your heart rate) is a good way to keep track of your health. A healthy resting heart rate is between 60 and 100 beats per minute. ♡♡

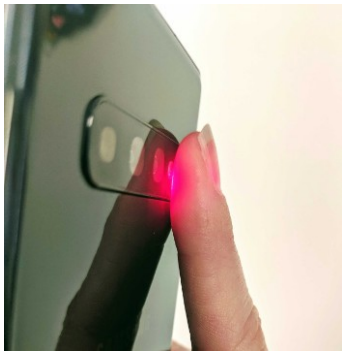
1.3.5. Heart rate

You can measure your heart rate too! And you don't need a micro:bit to do it. 😉

Place two fingers on the side of your neck, under your ear or under your jaw. You should feel your heart beat (pulse). Count how many times your heart beats in one minute. ⌚

Some smart devices measure heart rates using photoplethysmography (PPG for short). It is used in hospitals but you can also find it on smart watches and some smartphones.

Your heart rate is a great way to check how hard your body is working while you exercise. It will also tell you when you need to slow down! 🎵Listen to your heeaaart!🎵🎶



PPG measuring a heart rate on a smartphone. Source: Australian Computing Academy

PPG works by flashing light onto your skin. It measures how much light reflects back. When your heart is beating fast 🤍🤍, less light comes back. When your heart is beating slowly, more light comes back.

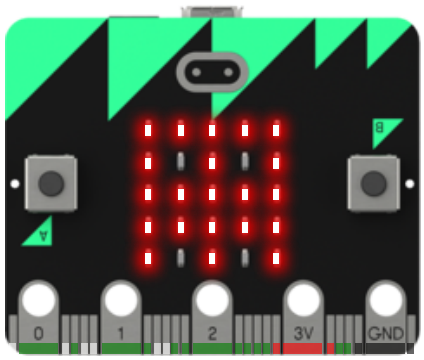
1.3.6. Problem: Duck, duck, ghost!

Let's get our hearts pumping with a bit of excercise! We are going to use the micro:bit to play Duck, Duck, Ghost! It's like Duck, Duck, Goose... only *spooookyyy*.

Use repeating `show` and `sleep` to play the game.

- First show a `Animals` `DUCK` for **4 seconds**.
- Then show a `Other` `GHOST` for **1 second**.

1. Drag in the right number of `show` and `sleep` blocks into the workspace.
2. Drag in your `image` blocks.
3. Select the correct images. If you can't see the ghost option, move your blocks down the screen and try to select it again.
4. Set the `sleep` time so the delay between the images is correct.
5. Make sure all your blocks are connected.



Activity!

When you are finished, download your program to your micro:bit. Sit everyone in a circle with one person standing on the outside.

That person is the and holds the micro:bit. While it shows a , they walk around the circle, tapping people to be ducks. When it changes to , tap the next person and say "BOO!".

The person must jump up and try and catch the before the takes the spot in the circle. If they're too slow, the person becomes the . **Use the reset button on the back of the micro:bit to restart the round.**

Sleep for milliseconds

To sleep for a number of seconds just add three zeros!
For example:

`sleep` `2000` `ms` will sleep for **2 seconds**

`sleep` `6000` `ms` will sleep for **6 seconds**.

`sleep` `13000` `ms` will sleep for **13 seconds**.

Testing

- [ ] Testing that the display starts with a duck.
- [ ] Testing that the duck is still on the screen than 2 seconds later.
- [ ] Testing that the display is still a duck after 4 seconds.
- [ ] Testing that the display changes to a ghost after 4 seconds.
- [ ] **Congratulations! Let's play!**

1.4. Letters and words

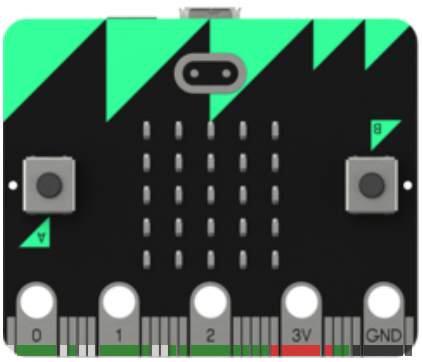
1.4.1. Scrolling letters and words

Now that you know all about showing images with your micro:bit, we can start to show words.

We use scroll to show letters on the micro:bit.

> **Strings**
>
> In programming, words and letters are called **strings**. " I'm a string "
>
> This is why you will find your green " string " blocks under the **Strings** drawer.

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!



1.4.2. Problem: I ♥ soccer

Time to combine the sleep and " String " blocks.

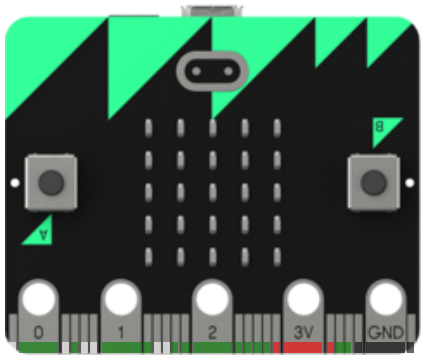
Do you love soccer? Your micro:bit sure does!

Program the micro:bit to show everyone how much you love soccer. Make `I ♡ soccer!` scroll on the LEDs.

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

Remember to ★ Mark your program!

Your program should look like this:



💡 Eeeeeeeek! 😱 My e looks too big!

Don't panic! It's not **eeee**vil!😈 Your micro:bit displays the **e** taller than the other letters because of how the LEDs are arranged.

Have a play with the scroll using **e** and **o** and **c**. Try a **y**! The display does it's best but they don't always end up the same height.

You can run your program as many times as you like before you mark. Make some fun signs! Show the world what you ♡!

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 an " s " scrolls past.
- [ ] Testing that " soccer! " scrolls past.
- [ ] **Great work! I love soccer too!!!** ⚽🥅😀🏆🎉

1.5. Summary

1.5.1. Congratulations!

You finished Part 1!

We learned about:

- the parts of the BBC micro:bit
- how to `show` an image on the micro:bit
- making the micro:bit wait with `sleep`
- scrolling `"strings"` on the micro:bit

Click » to learn about making decisions with buttons and gestures.

Next step we start to get a bit loopy...

2

LOOPS, BUTTONS AND MUSIC

2.1. Looping forever

2.1.1. Introducing loops

So far, our programs run each step one time.

Think back to the heart beat we played with before. It beat for a few seconds before it reached the end of the program and stopped. We want our heart beat to go on ♡ and on ♡ and on ♡ and on ♡♡♡...

We can use a micro:bit loop to repeat the heart beat **forever**!

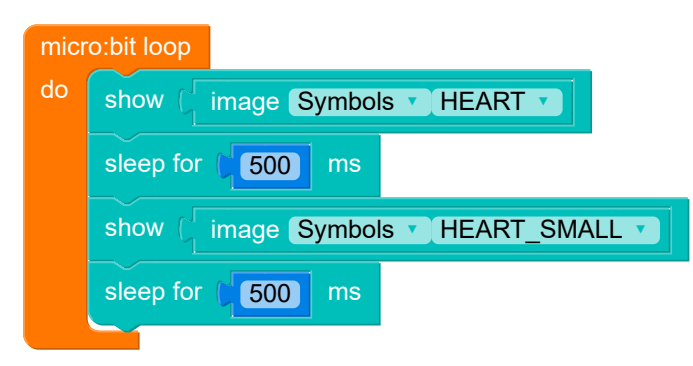
1. Click ▶ Run run below.

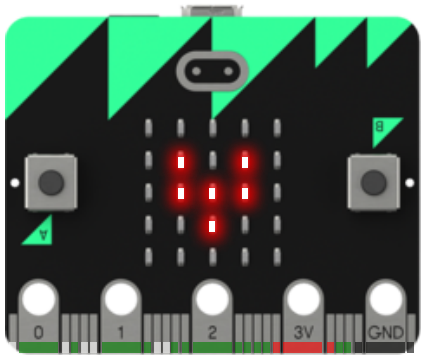
The heart can go forever!

2. Click the ■ Stop button.

3. If we go for a bike ride 🚴 or play some basketball ⛹ our heart rate gets faster! Make the heart beat faster by reducing the sleep time.

4. Run the program again!




B
A
0
1
2
3V
GND

2.1.2. Problem: Up, down, up, down

Have you ever had to do burpies in sports lessons? They are a good warm up exercise that help you stretch safely to get you ready to play!

- Jump in the air
- Touch the ground
- Stand up
- Repeat again, and again, and again...

We will use our new micro:bit loop block to do as many burpies as we can.

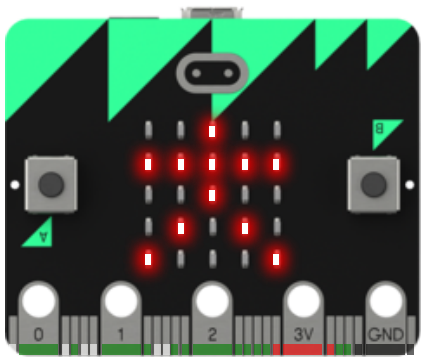
Your program should:

1. Show ARROW_N for **1 second**.
2. Show ARROW_S for **1 second**.
3. Show STICKFIGURE for **1 second**.
4. Loop this forever!

> **Use the loop**
>
> You will need to add blocks inside the micro:bit loop to complete the problem.

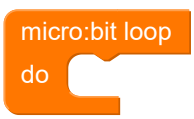
Check out the example below. It will look like that, but will **loop** forever!

> **Activity!**
>
> Try doing burpies in time with your micro:bit! If it's going too fast you can increase the sleep time. If you want a really tough challenge you can **decrease** the sleep time. Good luck!



You'll need

program.blockly



Testing

- [ ] Testing that the display starts with an up arrow.
- [ ] Testing that the display is still showing an up arrow after less than a second.
- [ ] Testing that the display shows a down arrow for a second.
- [ ] Testing that the display shows a stick figure for a second.
- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display goes back to an up arrow for a second.
- [ ] Testing that the animation loops continuously.
- [ ] **Congratulations! That should get the heart rate up.** ♡

2.1.3. Problem: Mascot parade ⌨

At the end of the annual micro:bit Athletics Carnival there is a parade of the winning mascots! 🎖🏆🎉

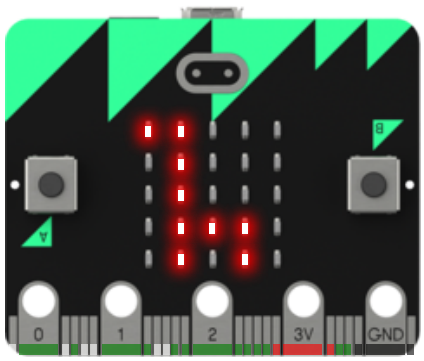
Practice using your loop block again to create a mascot parade.

Program your micro:bit to show the 1st, 2nd, 3rd and 4th place mascots **in order**. Make it loop forever!

1. show each Animal image for **1 second**. Make sure you show them in the correct order:
    1. BUTTERFLY
    2. COW
    3. DUCK
    4. GIRAFFE
2. And loop forever!

> **Don't forget**
>
> Your micro:bit should sleep for 1 second after the last animal to make sure you can see 👀 it.

Like this example (but your will loop forever):



You'll need

program.py

Testing

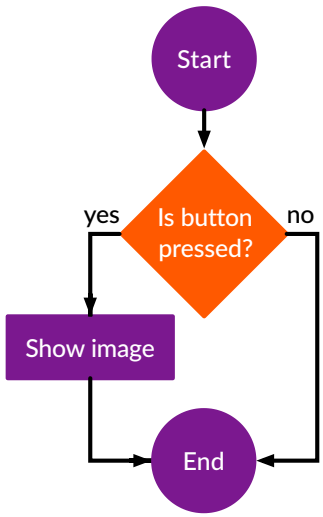
- [ ] Testing that the display starts with a butterfly.
- [ ] Testing that the display is still showing a butterfly after less than a second.
- [ ] Testing that the display shows a cow for a second.
- [ ] Testing that the display shows a duck for a second.
- [ ] Testing that the display shows a giraffe for a second.
- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display goes back to a butterfly for a second.
- [ ] Testing that the animation loops continuously.
- [ ] **Well done! You can loop forever!**

2.2. Making decisions with buttons

2.2.1. Making decisions

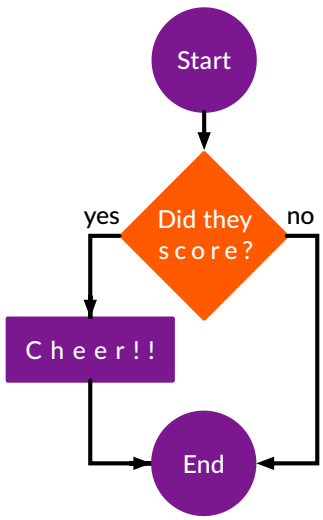
So far our programs only do one thing. A micro:bit has buttons that we can use to change that. When we press a button we are giving the program **user input** which can change what the program does.

For example, in this flowchart. An image is shown *only* if the button is pressed.



This is how we **make decisions** in a program.

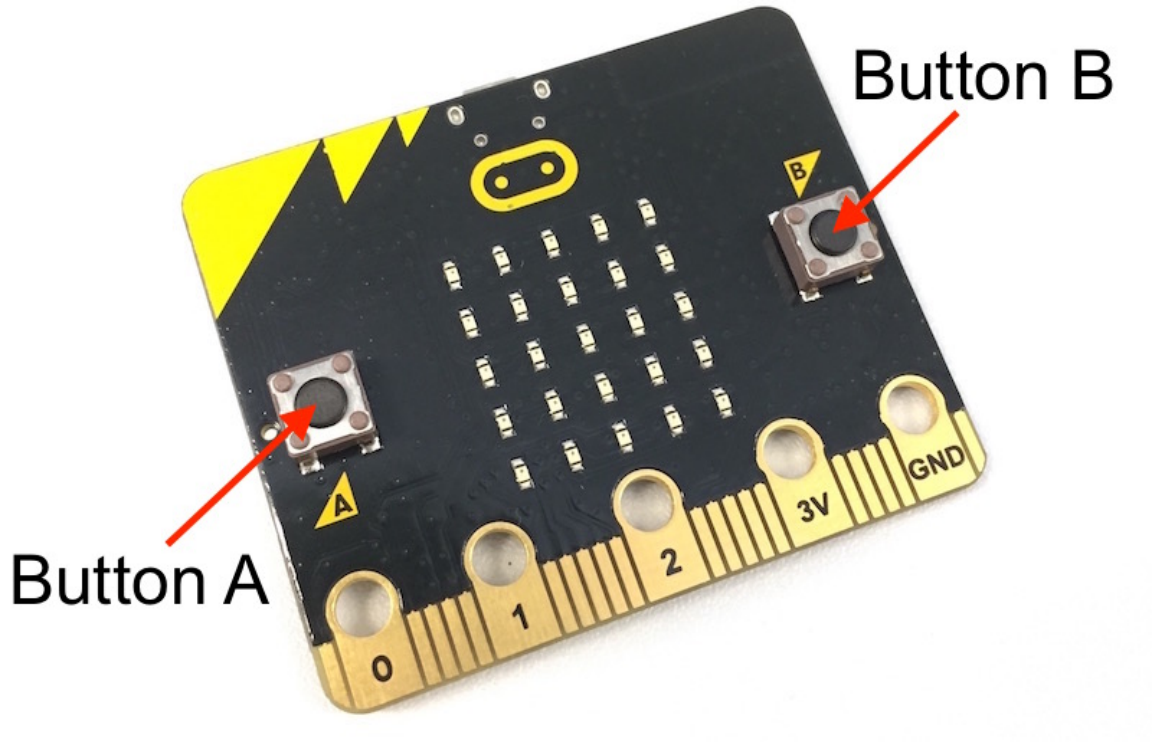
You make decisions all the time in sports games. He is an example you might know.



2.2.2. Button A and Button B

The BBC micro:bit has two buttons.

One is **A**. The other one is **B**.



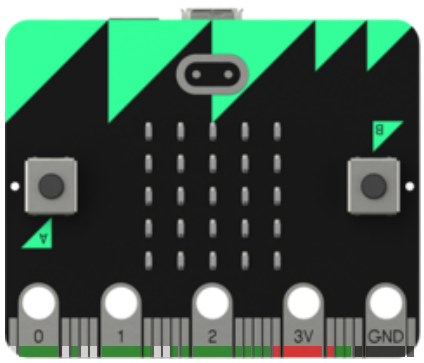
We can use button A is pressed and if to make decisions.

2.2.3. The if block

This is an if block.

1. ▶ run the example below.

If you try and press button A on the micro:bit **it doesn't work on its own!** 😱😱

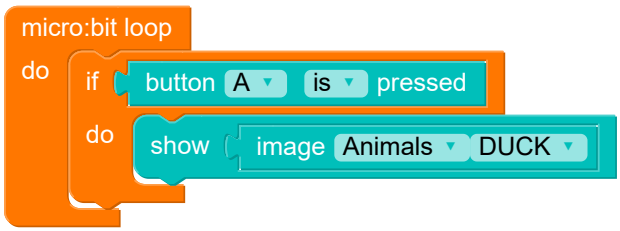


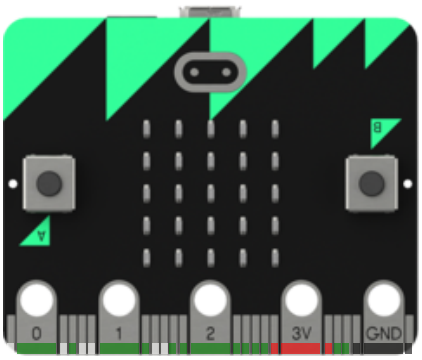
The New Loop

We need to put the if inside of the micro:bit loop .

2.

- run the second example below.





3. Press the ■ button.

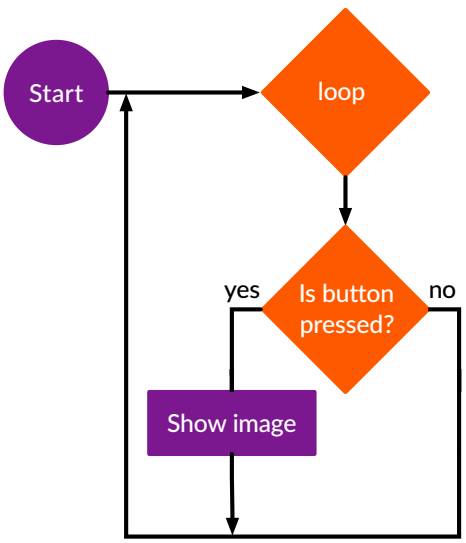
> **Use your mouse or keyboard**
>
> Press the buttons in the examples by clicking with your mouse or pressing A or B on your keyboard.

2.2.4. Making decisions inside a loop

Without a loop, your micro:bit will only check if the button was pressed once, then stop the program.

When you add a loop, the micro:bit will check if the button was pressed over and over and over again forever!

Here's a decision in a loop as a flowchart:



Follow this loop with your finger. You start at the purple circle. Then you enter the loop. If the button is not pressed, follow the loop back to the start. Then you start again.

If the button **is** pressed you show the image. After you show the image, you go back into the loop. And start again!

It goes on and on and on and on... Like our infinite loops before!

2.2.5. Problem: 3... 2... 1... GO!

On your marks, get set, GO!

Use your `if` block to make a count down timer that **starts if you press button A.**

Make sure you have an **infinite loop** checking if the button is pressed.

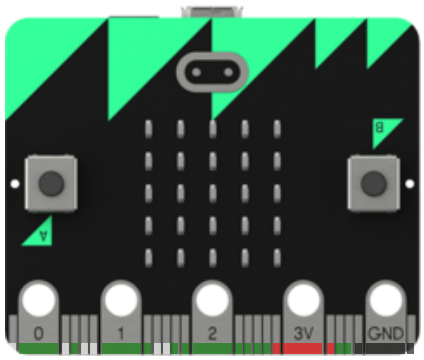
Your program should:

1. If `button A` is pressed.
    - Scroll `" 3 2 1 GO! "`
2. Loop forever!

We have given you some blocks to start with. You will need to add your loop and check if the button is pressed.

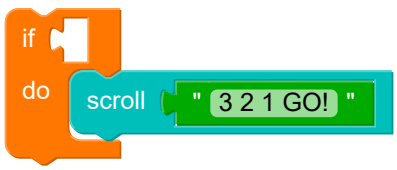
Your micro:bit should run like this.

Run the code ▶ and **press button A.** (You can also press A on your keyboard.)



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 counts 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.
- [ ] **Congratulations! You are getting loopy!** 😳😳🎉

2.2.6. Problem: The fastest freezer

The micro:bit has 2 buttons, A and B. **We can use both with 2 if blocks!**

Let's use both buttons to make a micro:bit game called "The fastest freezer".

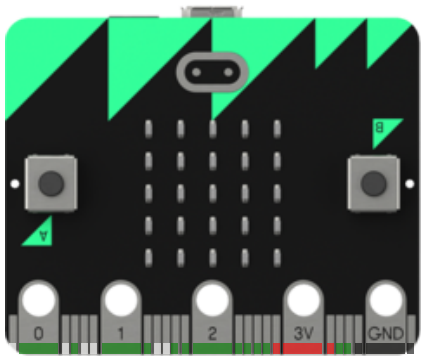
Your program should:

- show a YES ✔ when button A is pressed
- show a NO ✕ when button B is pressed.

YES and NO are found in the Symbols category.

Here is an example to help get you started. It will show a YES when A is pressed and a NO when B is pressed.

Try running it. Don't forget to press the A and B buttons!



Activity!

When you are finished, download the code to your micro:bit. Choose someone to be in charge of the micro:bit, everyone else is a dancer.

Dancers start dancing when the micro:bit is on "Go" ✔! When you see the micro:bit change to "Stop" ✕ you have to freeze!

The last dancer to freeze is out, if you lose balance you are also out. Keep going until you have one winner! The fastest freezer!

You'll need

program.blockly

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off blank.
- [ ] Testing that YES symbol ✔ shows when the A button is pressed.
- [ ] Testing that a NO symbol ✕ shows when the B button is pressed.
- [ ] Testing pressing A then pressing B work correctly.
- [ ] Testing that it continues to work multiple times.
- [ ] **Yes! ✔ ✔ ✔ Nice work!**

2.3. Making Music

2.3.1. Connecting headphones

The micro:bit has lots of built-in components (another name for parts) like the LEDs. It can play music through headphones, but you will need some extra parts.

> **Hack your headphones!**
>
> If you have a real micro:bit, follow these instructions (https://makecode.microbit.org/projects/hack-your-headphones) to play sound through your headphones!
> You will need a battery pack!

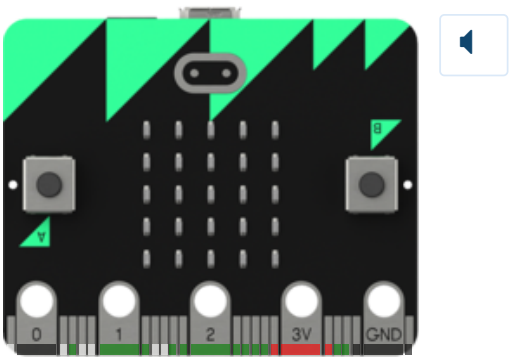
From now on, when you see this button next to the micro:bit in our simulator, it means that the headphones (or speaker) are connected and playing.

If your computer has speakers it will play the music. You can plug your headphones in and listen too.

Don't forget to turn on your computer's sound!

2.3.2. Playing music

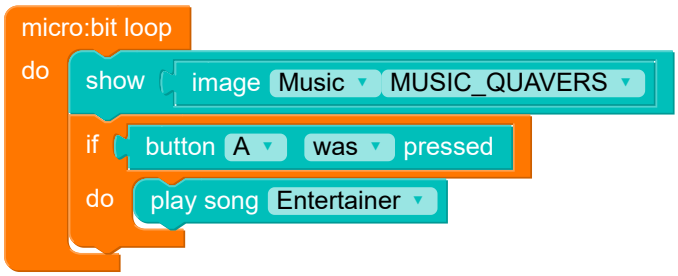
It's time to play your first song! Run the example below!

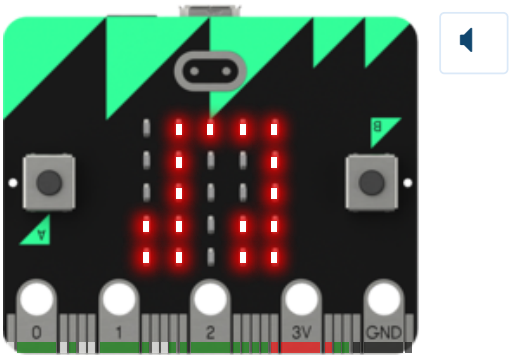


If you can't hear anything, check that your sound is on!

Here is another example. This time you have to press button A to play the song.

Try the different songs and sounds! Click on Entertainer so see more options.

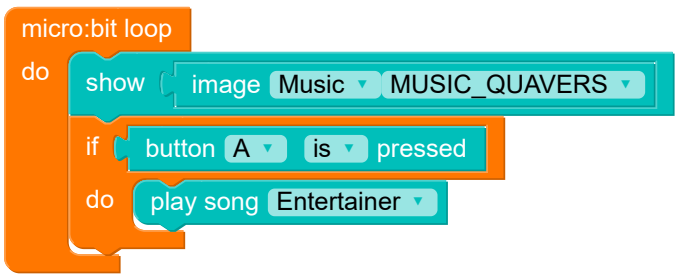


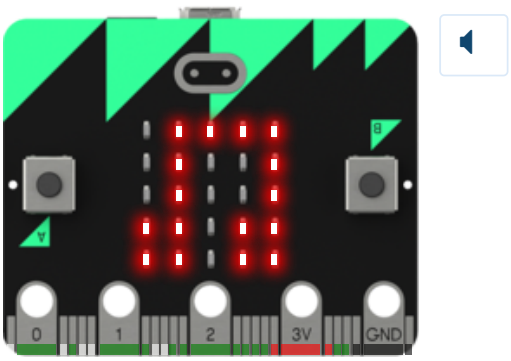


2.3.3. Is pressed or was pressed?

So far you have only used `button A is pressed` to work your buttons. But it won't always give us the best result.

For example, if we just want to play a song **one time through** no matter how long we hold down the button. Try running these blocks. **But make sure you hold down button A**



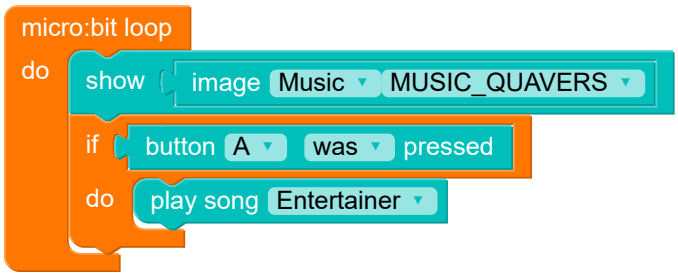


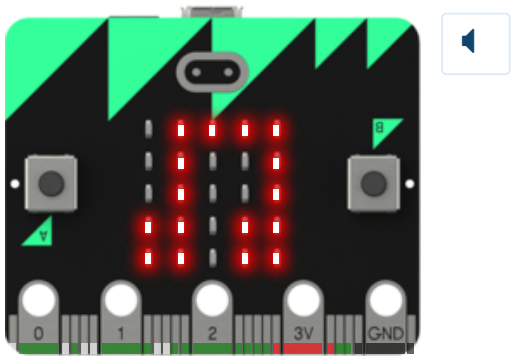
The music plays again and again and again and again... until you let go of the button. 😫😫

This is when we use the `was pressed` option! Changing the `is` to `was`.

It means that the song will only play **once**, even if you hold down the button! 😁

Try this example, and hold down button A.





The song only plays once! Great! Let's put that into action.👍

2.3.4. Problem: Toe touch tester

Time to get everybody touching their toes! We will build a Musical Toe Touch Tester!

When you are finished, download your code to your micro:bit. Put it face up on the floor by your feet.

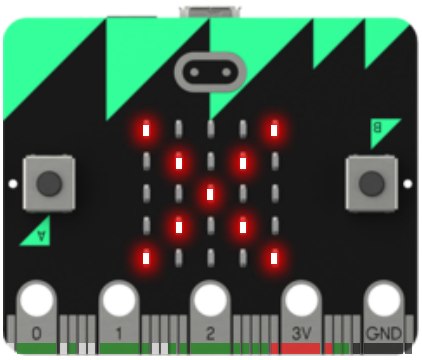
Every time you bend and touch button B your micro:bit will give you a smile and sing you a song! It's a good way to stretch!

Your program will need to:

- start by showing a NO ✕ on the micro:bit
- if button B was pressed show a FABULOUS face **then** play Power Up
- use an infinite loop to go on forever

> If you aren't sure where to start, check the examples on the previous slide.

Here is how your program should work



> **Sound on a real micro:bit**
>
> Instructions on how to hack your headphones can be found on our micro:bit cheatsheet (https://aca.edu.au/resources/microbit-cheatsheet-poster/microbit-cheatsheet.pdf)!
>
> Here's a video if you want to see a real micro:bit with a speaker attached
>
> 0:00 / 0:03
>
> 
>
> How to use your toe touch tester.

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the micro:bit is initially not playing any sound.
- [ ] Testing that the micro:bit starts by showing a NO.
- [ ] Testing that pressing Button B shows `Fabulous` .
- [ ] Testing that pressing Button B starts playing `Power Up` .
- [ ] Testing that you used `was pressed` .
- [ ] **You can touch your toes!**

2.4. Summary

2.4.1. Congratulations!

Fantastic work! You made a 🎶Musical Toe Touch Tester!🎶

We learned about:

- visualising decisions as flowcharts
- a micro:bit loop that repeats forever
- buttons on the micro:bit
- simple decisions with if blocks
- is pressed and was pressed ! You can do all this and more with your micro:bit!

Have fun! See you in the next module! 🤘😃

3

SIMPLE DECISIONS AND VARIABLES

3.1. Decisions with 2 options

3.1.1. Decisions with two options

When we make a decision, we have only been working on the 'yes' answers.

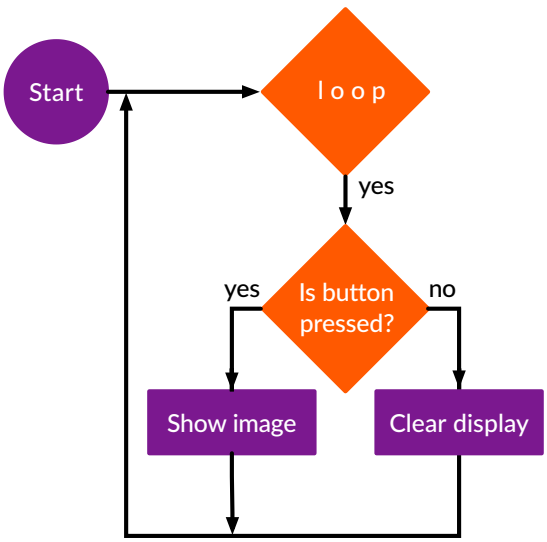
Was the button pressed? Yes. Do something.

What if we want to do something if the button is **not** pressed?

If we ask "Is the button pressed?" we could:

- Show an image if the answer is yes.
- Hide the image if the answer is no.

Like in this flowchart:



3.1.2. The `if/else` block

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

It lets us say what to do if something is pressed **and** if something is **not** pressed.

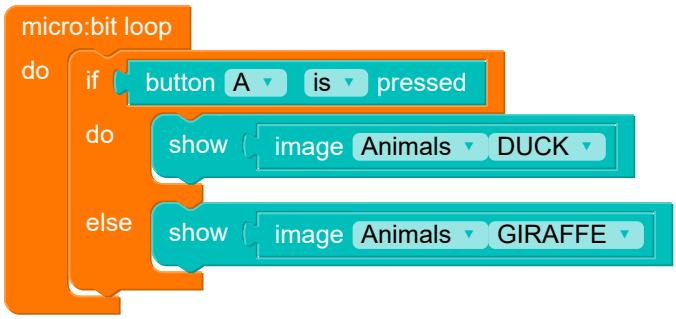
In the example below:

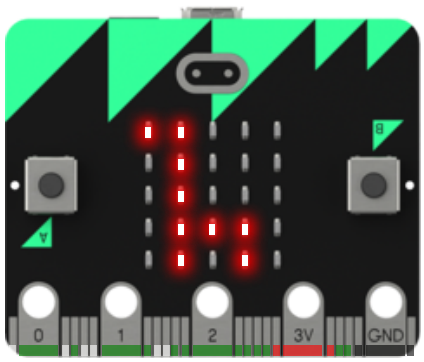
- If button A is pressed, we will show a duck

- if button A is **not** pressed, we will show a giraffe

You can see how this works in the example below.

1. ▶ run the example below.





3.1.3. Problem: Smile for the camera!

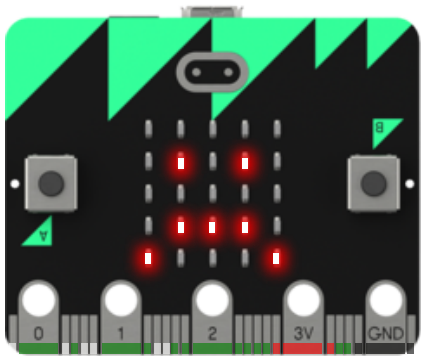
Smile for the camera!

Write a program to show a happy face if button A is pressed, but otherwise show a sad face.

1. If button A is pressed
    - Show a HAPPY face ☺
2. Else
    - Show a SAD face ☹

You will need to use the if/else block that you learned about in the last slide.

Here is what your micro:bit should do:



You will need to press button A **!**

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off showing a sad face.
- [ ] Testing that it becomes happy when the button is pressed.
- [ ] Testing that it went back to a sad face after the button was released.
- [ ] Testing that holding down the button keeps the happy face on the screen.
- [ ] Testing that it continues to work multiple times.
- [ ] **Nice work! Ready for your close up!** 📷😃🎉

3.2. More Complex Decisions

3.2.1. Use both buttons!

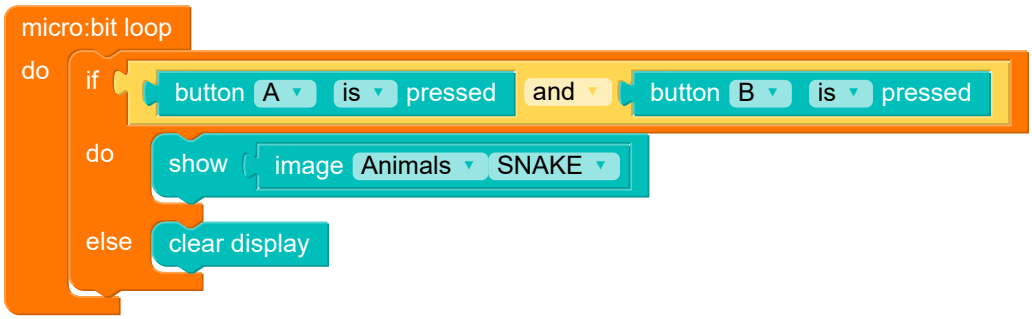
Play the example!

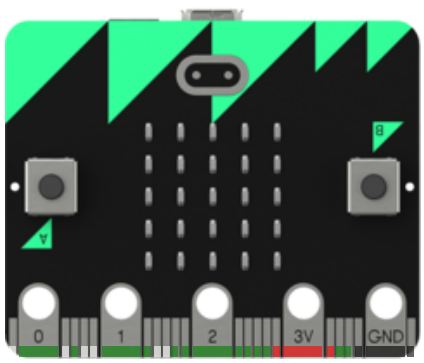
Press button A... nothing happens.

Try pressing button B... Nothing

Press both A and B at the same time? **Snake!!**

This is the and block!



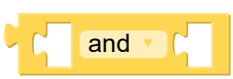


Clear display?

The clear display block turns off all of the LEDs. It is a good way to reset your micro:bit while buttons are not pressed.

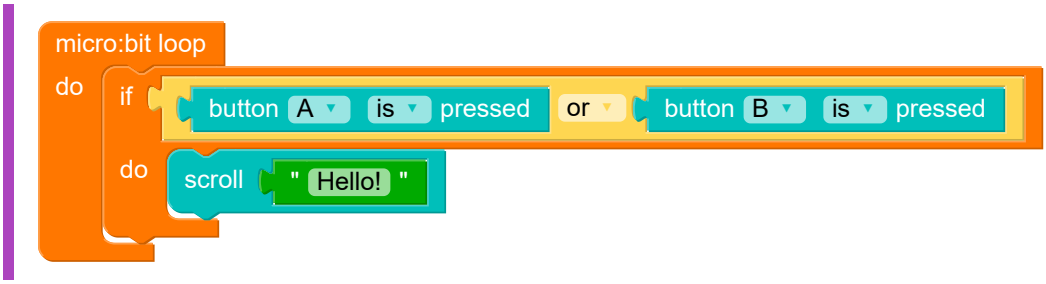
3.2.2. The and block

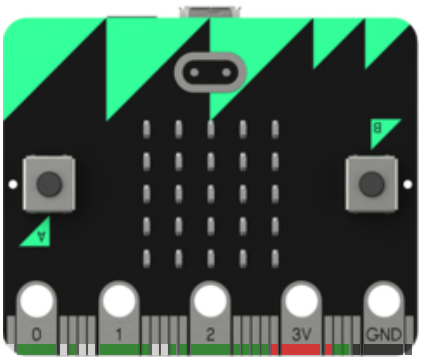
The and block lets you define what happens when both buttons are pressed at the same time.



You can also select the or option from the drop down! Using or you can define what happens when either of the buttons are pressed.

For example, if button A or button B are pressed, I want to scroll " Hello! " .


micro:bit loop
do
if
button A
is
pressed
or
button B
is
pressed
do
scroll
" Hello! "


0
1
2
3V
GND

3.2.3. Problem: AND Smile for the camera!

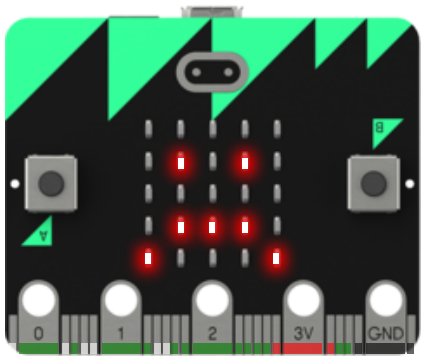
Smile for the camera! Again!

Write a program to show a happy face if button A and button B are pressed at the same time. Otherwise show a sad face.

1. If button A and button B are pressed
    - Show a **HAPPY** face ☺
2. Else
    - Show a **SAD** face ☹

You will need to use the if/else block and your and block.

Run the micro:bit below to see an example interaction:



Use your keyboard to press button A and B at the same time.

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off showing a sad face.
- [ ] Testing that it becomes happy when both buttons are pressed.
- [ ] Testing that it went back to a sad face after the buttons are released.
- [ ] Testing that holding down the buttons keeps the happy face on the screen.
- [ ] Testing that it continues to work multiple times.
- [ ] **Fantastic work! What a photo finish!** 📷😉

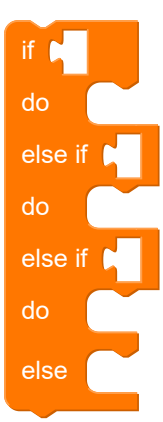
3.2.4. The elif block

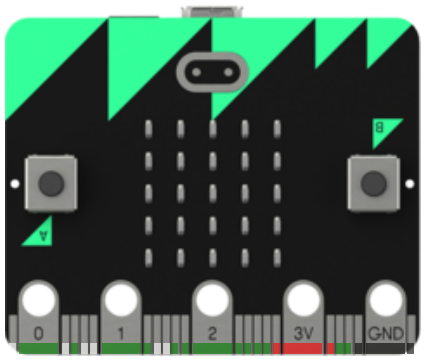
If we want, we can make the micro:bit do something if:

- nothing is pressed
- Button B is pressed
- Button A is pressed
- Both buttons are pressed

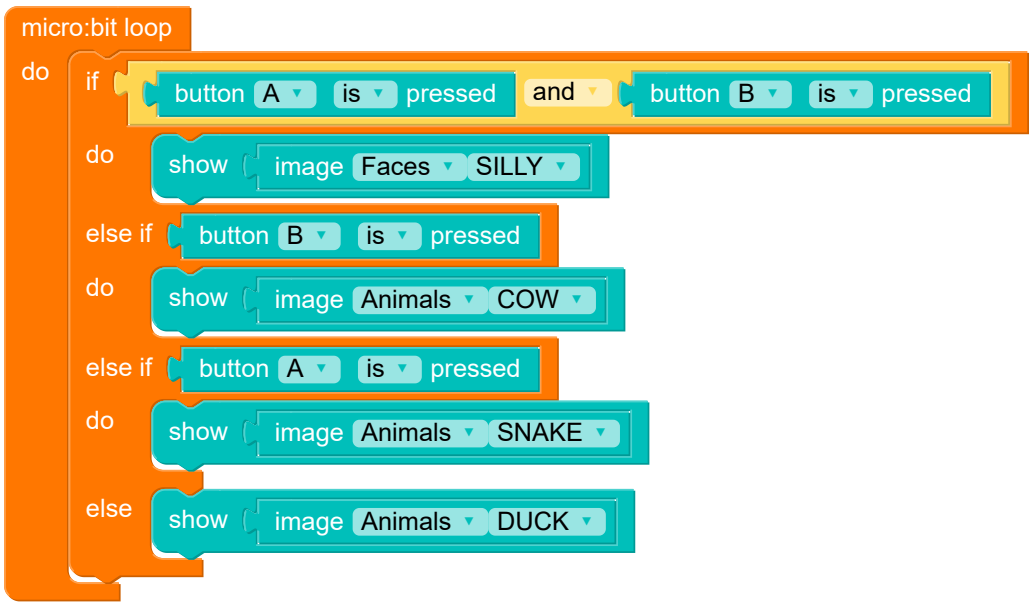
So many options! 😱

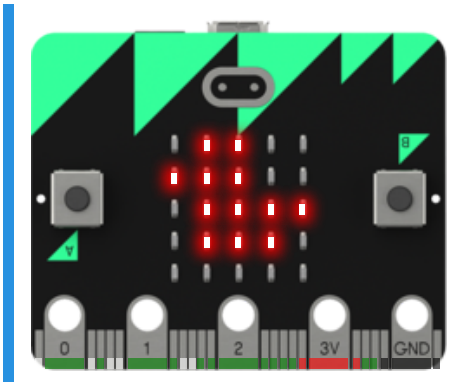
We need another block if we want to make that many decisions. We can use else if





Here it is showing all sorts of animals. Can you make the face display?





3.2.5. Elif Decisions

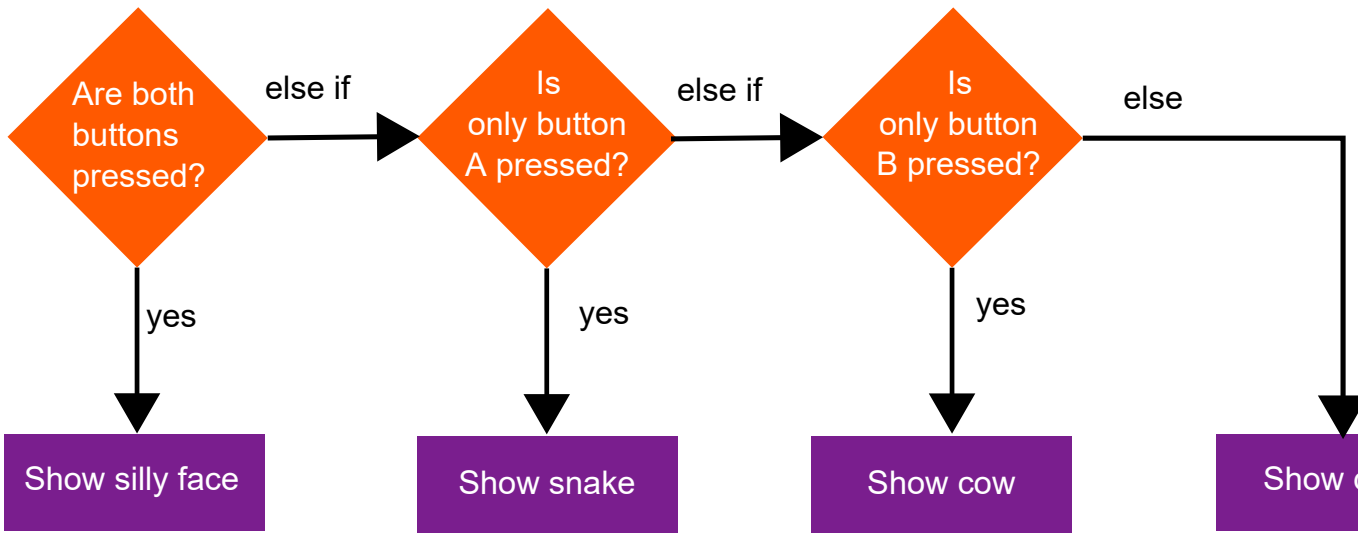
Let's break down that example with the flow charts we have used before.

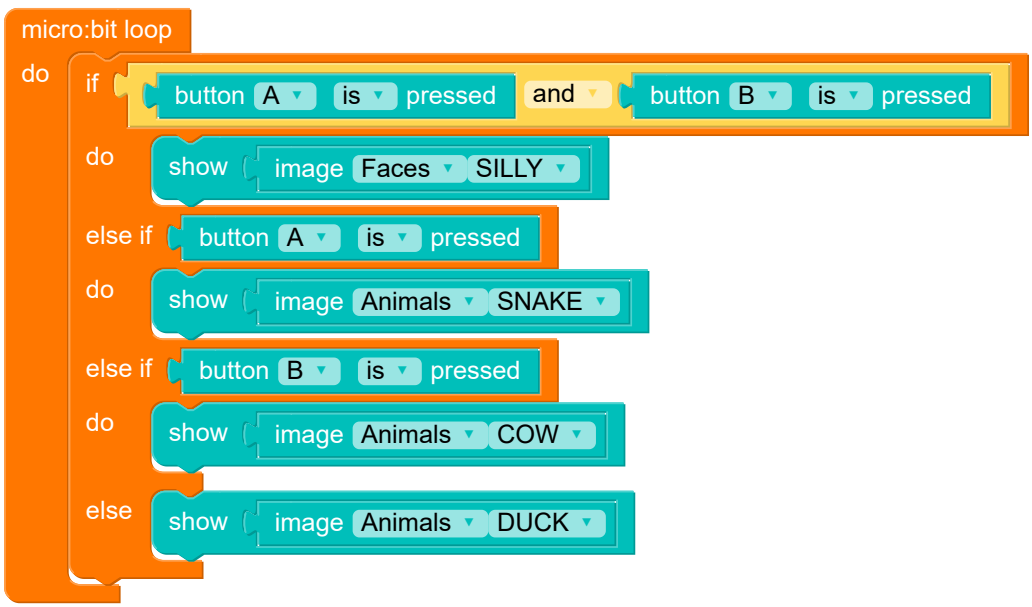
Follow the flow chart and the code together.

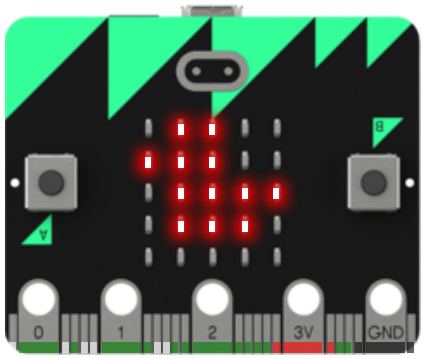
Order Matters!

What happens if you swap the order of the if statements? Try swapping the block that check for Button A AND button B for the statement that just checks button A.

Does it still work?






0
1
2
3V
GND

3.2.6. Problem: Dance Dance Revolution

Have you heard of Dance Dance Revolution (https://en.wikipedia.org/wiki/Dance_Dance_Revolution) (DDR)? We are going to build a DDR micro:bit.

Write a program to draw an arrow on the display indicating the direction you should step. The display should be blank when not moving.

Your program should:

- If button A and button B are pressed, show a forward arrow
- else if button B is pressed show a right arrow
- else if button A is pressed show a left arrow
- else clear display

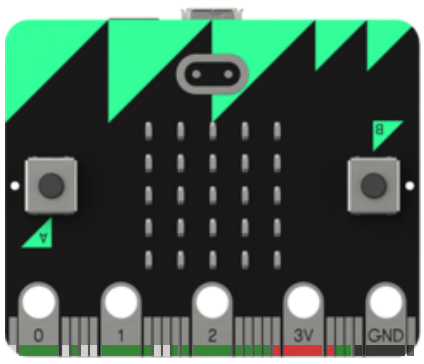
Here are the arrow images:

Name	Direction	Image
image Arrows ARROW_N	forward	
image Arrows ARROW_E	right	
image Arrows ARROW_W	left	

Order Matters!

Be careful what order you check the buttons. The order of your **if statements** matter. Check the previous slide if you get stuck.

Here is your example micro:bit.



Activity!

Dancing is a great way to stay fit! You can use it to get your friends to dance to your favorite song when you are finished. Every time they see an arrow they have to move the correct way, **then step back to the center.**

Testing

- [ ] Testing that the display starts blank.
- [ ] Testing that the display shows the up arrow when both buttons are pressed.
- [ ] Testing that the display goes blank again when the buttons are released.

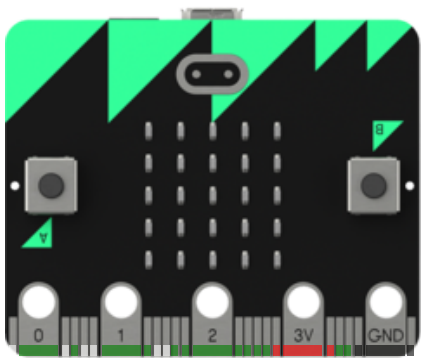
- [ ] Testing that the display shows the left arrow when the A button is pressed.
- [ ] Testing that the display goes blank again when A button is released.
- [ ] Testing that the display shows the right arrow when the B button is pressed.
- [ ] Testing that the display goes blank again when the B button is released.
- [ ] Testing none → A → A+B → B → none.
- [ ] **Well done!! You did it! Code Code revolution!** 🕺😇🕺😇🕺😇🕺

3.3. Variables and time

3.3.1. Variables

If you wanted your micro:bit to scroll a cheer for your favourite team, how would you do it?

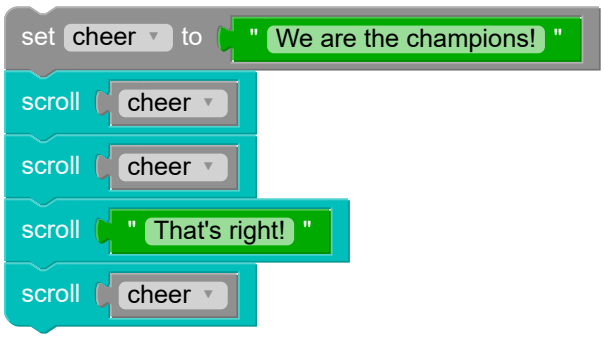
Something like this?

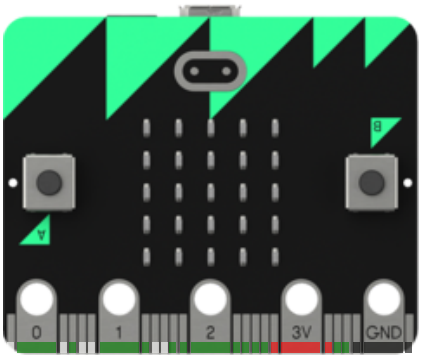


That's a lot of typing!

We can save time and typing by using a variable to save your cheer. That way we can use it again and again without having to type it out.

This code scrolls the same messages but using a variable called cheer.





3.3.2. Team variable!

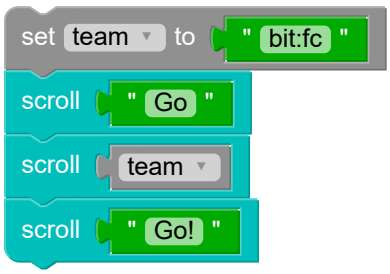
Variable variables

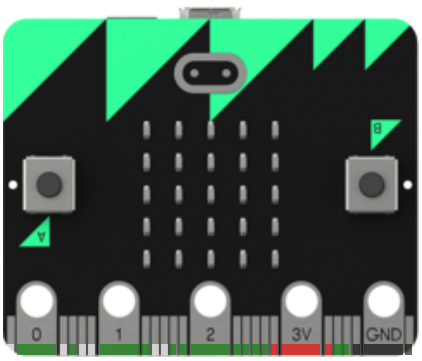
Variables are called variables because the value inside them can change!

In this example, the variable is " bit:fc " .

If you change the team name stored in the variable, you will see the message change too.

1. Run the example below.
2. Change " bit:fc " to your favourite sports team.
3. Run the code again to see the new team name.





3.3.3. Problem: Which way is the goal?

When teams are playing on fields, they usually swap goals after half time. This is to help keep the game fair.

We can build a micro:bit program to help the team remember which way the goal is. Just in case they forget! 😳😁 We will need to use our new variables.

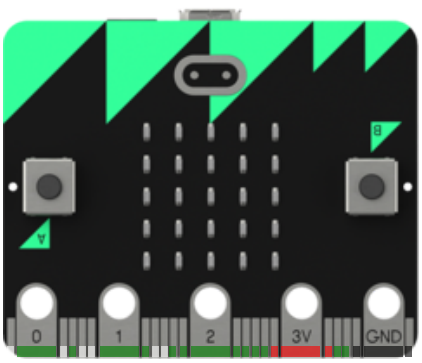
Your program should:

- Set the variable called `goal` to `ARROW_W`
- When you hold down `button A`
  - show the variable `goal`
  - Sleep for 500ms
  - then `clear display`
- if `button B` `was pressed` , it's half-time and you need to change goals!
  - Set the variable called `goal` to `ARROW_E`
  - now, when you press A you should see the new arrow.
- loop forever

We have built your first variable. You will need to set the goal image and complete the problem.

Building Variables

To create a variable, open the variables tab and click `create variable` . You will need to give your variable a name, then you will get blocks to use.



You'll need

program.blockly

Testing

- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that your code starts blank.

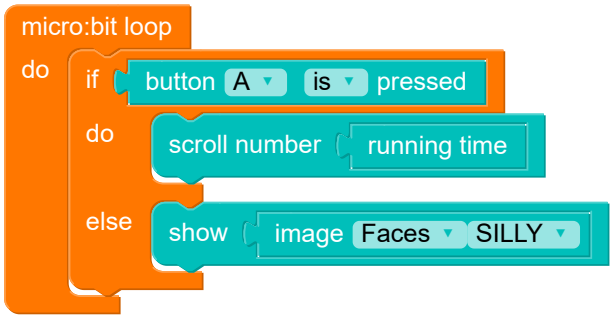
- [ ] Testing that you show a West arrow when button A is pressed.
- [ ] Testing that you used `is pressed for button A .`
- [ ] Testing that your code changes to an East Arrow when button B was pressed.
- [ ] Testing that your new arrow shows every time you press Button A.
- [ ] **Congratulations!🎉 You can use variables! 📣⏹⏹**

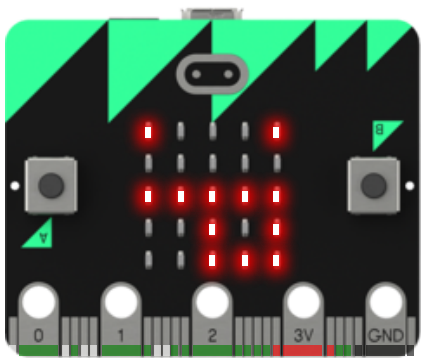
3.3.4. Micro:bit time

The micro:bit doesn't have a clock, so it can't tell the time of day.

However, you can find out how long it's been running since it was last switched on or restarted by calling `running time`. This function returns how long it's been running in **milliseconds.**

Try running the example below and waiting a few seconds. Then press A. The number that scrolls is the time your micro:bit has been running! Press A again, the number will have increased.





Look carefully at the example! Did you notice the new `scroll number` block? You need it to scroll the running time.

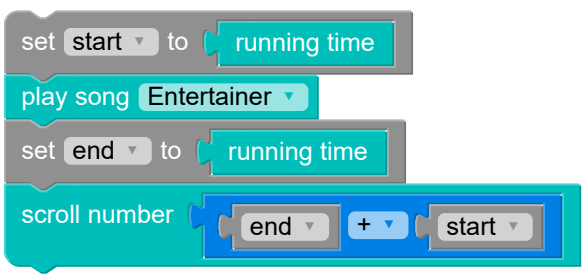
3.3.5. Measuring time

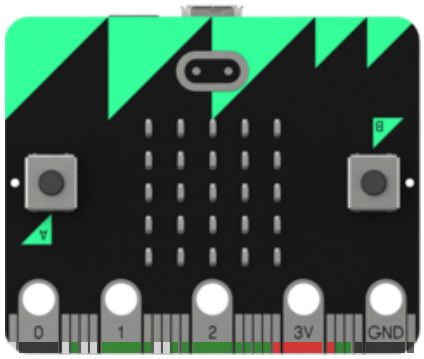
We can use the running time block to measure how long things take.

For example, we can measure how many milliseconds long the Entertainer song goes for.

We save the running time at the start of the song. As the song plays the running time keeps counting up. Then we save what the running time is when the song finishes. Subtract the finish time from the start time and you get the length of the song!

Run the code below to see the time:




0
1
2
3V
GND

3.3.6. Problem: Reaction time tester

How fast are your reaction times? ? We will build a reaction time tester using the variable and running time blocks you just learned about.

Press button A to start the game. Your micro:bit waits 3 seconds then shows you a diamond. As soon as you see the diamond, press button B! As fast as you can!

Your micro:bit will then show your reaction time in milliseconds. The smaller the number, the faster you are!

To create a variable, open the Variables drawer and click on `create variable`. You will need to name the variable and click ok. After that it will appear in your variables drawer.

When `button` `A` is pressed:

- Show the `NO` symbol
- Sleep for `3000` milliseconds
- Set the `running time` in a variable called `start`
- Then, show `Symbols` `DIAMOND`

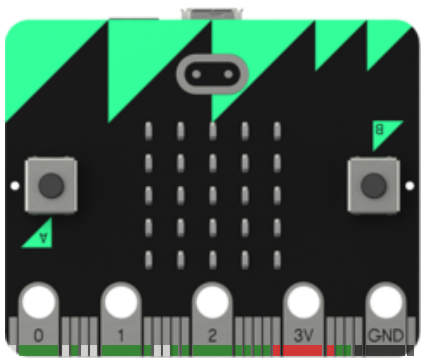
When `button` `B` is pressed:

- Set the `running time` in a variable called `end`
- scroll the number you get from `end` - `start`

We have put out some of the blocks you will need.

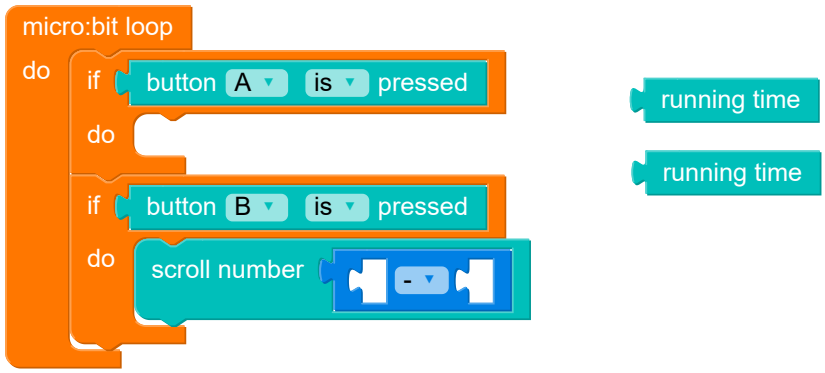
> **Pressing B**
>
> If you play the example and press B straight away, what happens? Your micro:bit scrolls an error!
>
> This is because you program the micro:bit to scroll a variable when B is pressed. If you press too early, the variable doesn't exist yet! If this happens, just reset the micro:bit.
>
> So no cheating!

You can play the reaction time game with this example:



You'll need

program.blockly



Testing

- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that the display starts blank.
- [ ] Testing that your display shows a NO after button A has been pressed.
- [ ] Testing that it holds NO for 3 seconds.
- [ ] Testing that your program shows a diamond.
- [ ] Testing that your code shows a diamond for a long time.
- [ ] Testing that pressing B stops the timer.
- [ ] Testing if you scroll the reaction time.
- [ ] **You're a time lord!** ⌛

3.4. Summary

3.4.1. Congratulations!

Congratulations! You have made your micro:bit into a reaction time tester.⏱

We learned about:

- Using the `if/else` block
- Using the `elif` block to make lots of decisions
- All about `Variables`
- Using the micro:bit to measure time

Nice work!

4

ACCELEROMETER AND LOOPS

4.1. The Accelerometer

4.1.1. Welcome to Module 4

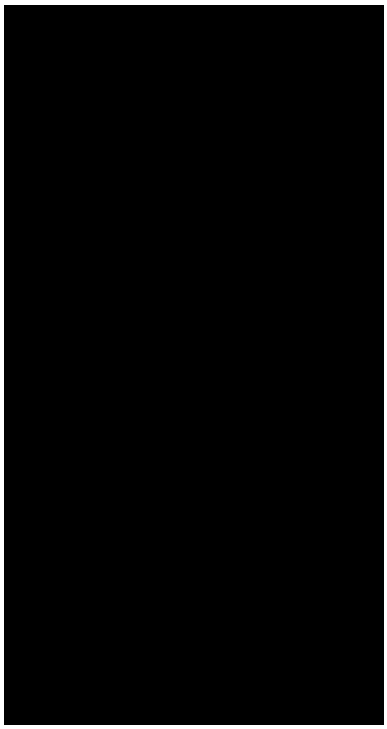
In this module we will start to bring together what we are learning into projects.

You will learn how to use the Accelerometer, some more loops, playing music on your micro:bit and more!

Let's get started!!

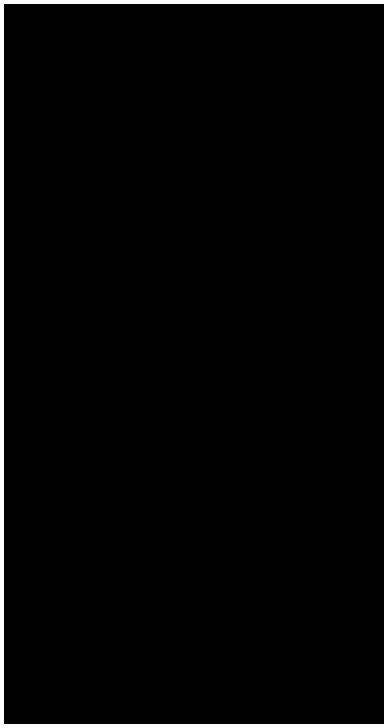
The project at the end of this module is the Milk Bottle Challenge!

The goal of the challenge is to hold up a milk bottle as long as your muscles can. You will program the micro:bit so that you can put it on the bottle and it will time how long you can hold it up for. If you shake too much or put the bottle down, it will beep at you and show you your time.



The milk bottle challenge micro:bit.

This is an up close look at what your micro:bit will be doing. You press A and the program starts to check if you shake, and shows a stickfigure. When the bottle is shaken (because you have put it down or maybe laughed too much 😛) then it scrolls your time.



0:00 / 0:09

The milk bottle challenge up close.

4.1.2. Accelerometer

The micro:bit has a built-in accelerometer (https://en.wikipedia.org/wiki/Accelerometer) that measures *acceleration*.

Acceleration is changes in speed (speeding up and slowing down) and direction (curving). It's not just going faster!

Lots of other devices contain accelerometers, including smartphones, fitness trackers, and some game controllers.

Using an accelerometer, you can detect which way the device is facing (e.g. screen orientation on your phone).

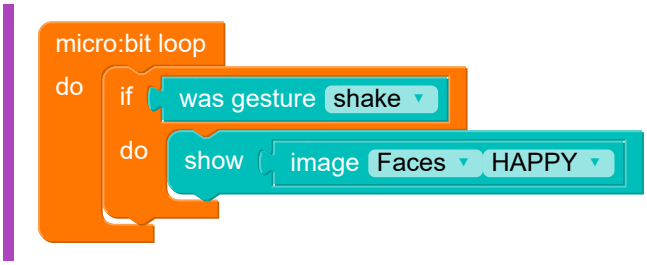
You can also detect movement where acceleration changes (such as shakes and falls).

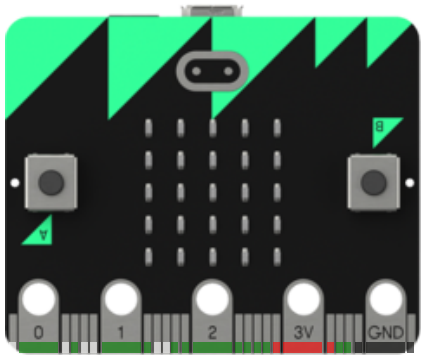
4.1.3. Detecting Shakes

The Micro:bit detects different types of movement using the `was gesture` block. You can select which gesture you would like to detect using the drop-down list.

This is an example of a program that uses the accelerometer block. Run it and click the **shake** button.

You can use the accelerometer to make decisions. Use it inside of an `if` statement, in a similar way to the buttons blocks.





Shake

4.1.4. Problem: Try not to shake!

Time to try the `was gesture` block out!

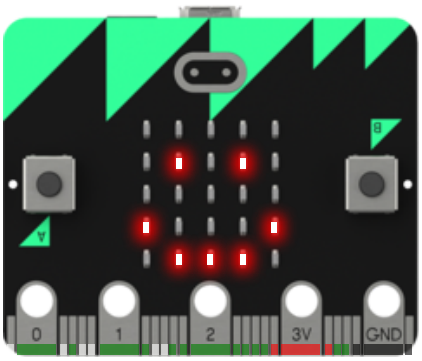
Do you think you are balanced enough to walk 5 steps without shaking your micro:bit? Let's write a program that you can use to check.

You will need to choose a the correct gesture option from the `was gesture` drop-down list.

Your program should:

- Show a happy face on the microbit 🙂
- If the micro:bit `was gesture` `shake` , show a skull 💀 for 1 second
- Don't forget to loop forever

Here's an example for you to try. Shake it about!



Shake

Activity!

When you are finished, you can download your code to a real micro:bit. See if you can walk 5 steps holding your micro:bit without triggering the skull! 💀 After you have practiced, race your friends and see if you can make it to the finish line without shaking!

You'll need

components.json

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that your code starts showing a happy face.
- [ ] Testing that your code stays happy if you do not shake.
- [ ] Testing that your micro:bit detects a shake.
- [ ] Testing that the display goes from happy to skull when the micro:bit detects a shake.
- [ ] Testing that you display the skull for 1 second
- [ ] **Congratulations! Good luck balancing your Micro:bit!**

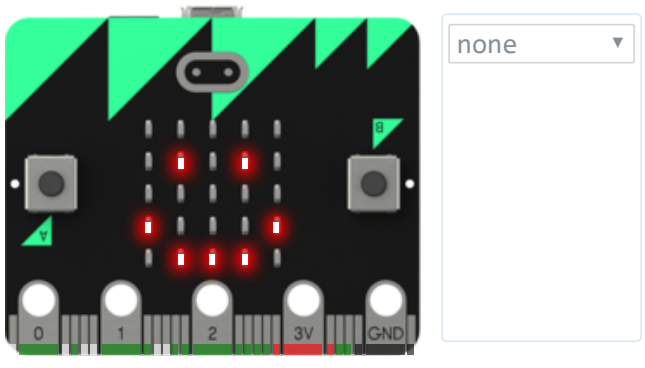
4.1.5. Problem: Look up!

Is your micro:bit scared of heights? Write a program that shows a surprised face on your microbit when it goes up in the air!

This time we will use the was gesture block and if do else block.

Your program should show a surprised face 😳 for 3 seconds after the micro:bit detects it went up. Otherwise, show a happy face on the microbit 🙂

Here's an example for you to try. You will need to select 'up' from the drop-down. Try not to scare your micro:bit too much. 😱😱



When you are finished, you can download your code to a real micro:bit.

You'll need

components.json

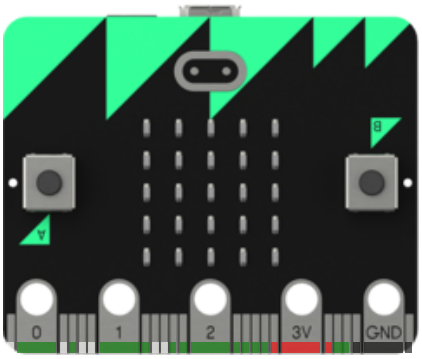
Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that your code starts showing a happy face.
- [ ] Testing that your code stays happy if you do not move the micro:bit upwards.
- [ ] Testing that the display goes from happy to surprised when the micro:bit detects an upward movement.
- [ ] Testing that the display shows a surprised face for 3 seconds.
- [ ] Testing that your micro:bit goes back to a happy face.
- [ ] **Congratulations! Good luck balancing your Micro:bit!**

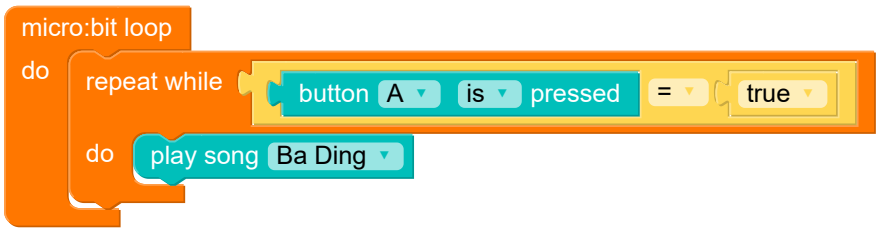
4.2. Loops within Loops

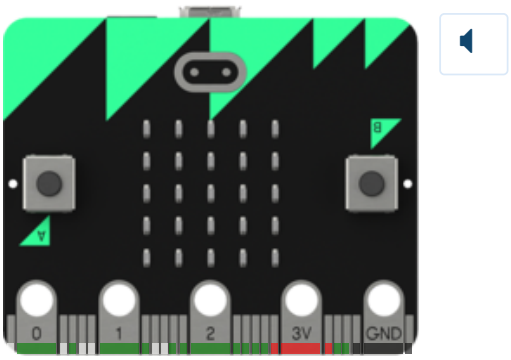
4.2.1. The repeat while loop

This is the repeat while loop.



It will repeat the **do** space while the **condition** is true. For example, this program will play the song "ba ding" while it is true that button A is pressed.





Let's take a closer look at the condition in this while loop.

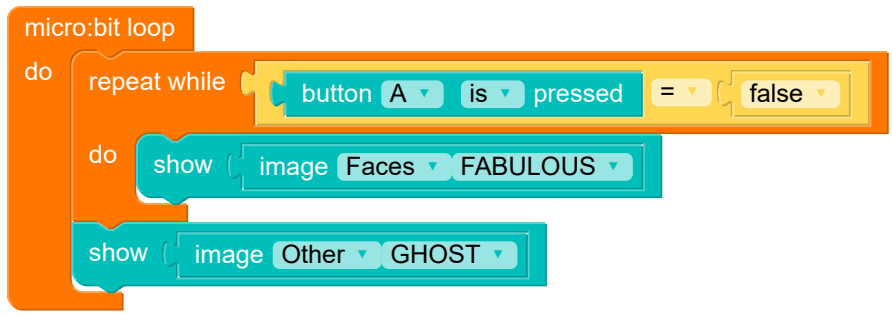
The condition in this example is that **it is true that Button A is pressed.**

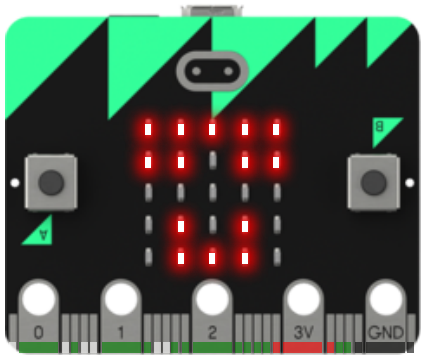
We will look at more examples in the next slide.

4.2.2. Conditions

Here are some more examples of conditions. **Notice that they don't always use the and block.**

repeat while button A is NOT pressed



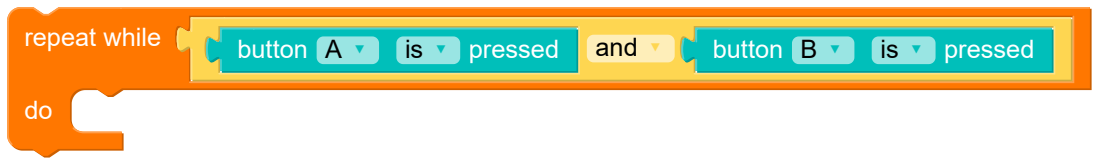


If it is **true** that button A is **not** pressed, the micro:bit will show a fabulous face 😎

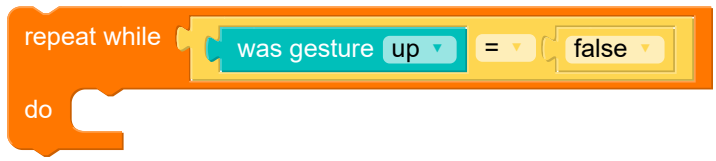
If it is **false** that button A is not pressed, the computer will skip the blocks inside the do space and show a ghost 👻

Here are other examples of condition blocks by themselves.

repeat while button A and button B are pressed

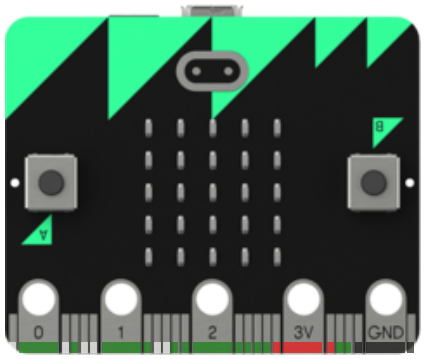


repeat while the micro:bit has not been shaken



4.2.3. Conditions continued

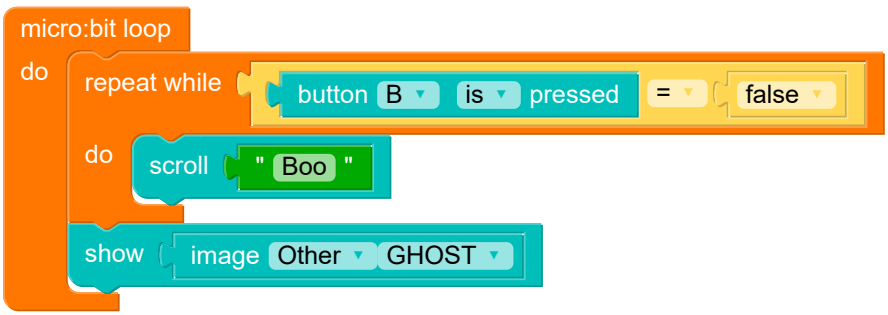
Here is a list of some of the symbols from the = block and what they mean:



Symbol	Meaning
=	equals
≠	does not equal
<	less than
≤	less than or equal to
>	Greater than
≥	greater than or equal to

4.2.4. Problem: What is the condition?

What condition must be true for Boo to scroll?



- ○ Button B is pressed
- ○ Button B is NOT be pressed
- ○ A ghost must be showing
- ○ No buttons must be pressed

Testing

- ☐ That's right!

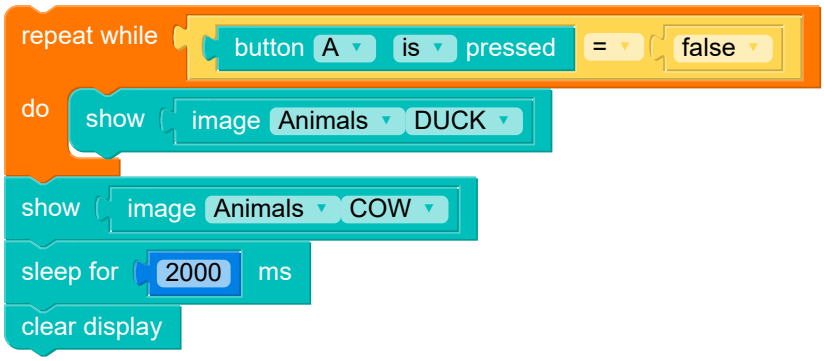
4.2.5. When the loop ends

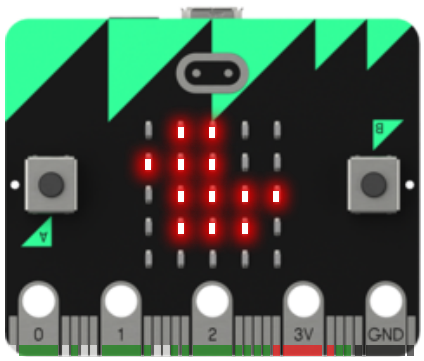
When a loop ends, the program moves on to the next blocks in the sequence.

This example will loop and show a duck while button A is not pressed.

When button A is pressed, the loop will end and the micro:bit will move onto the next blocks.

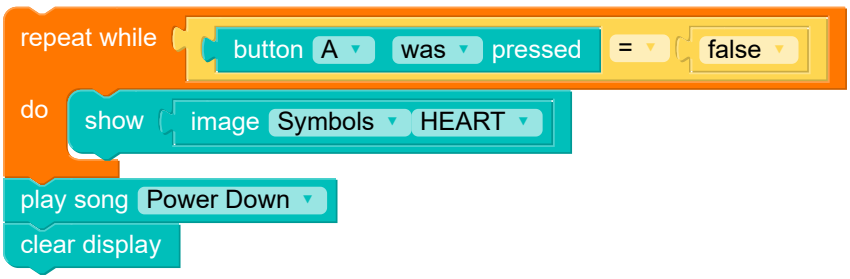
It will show a cow for 2 seconds and then the screen clears.





4.2.6. Problem: What is checked?

What will happen when button B is pressed?



- Nothing will happen.
- Power Down will play.
- Power Down will play and then the screen will clear.
- The screen will clear.

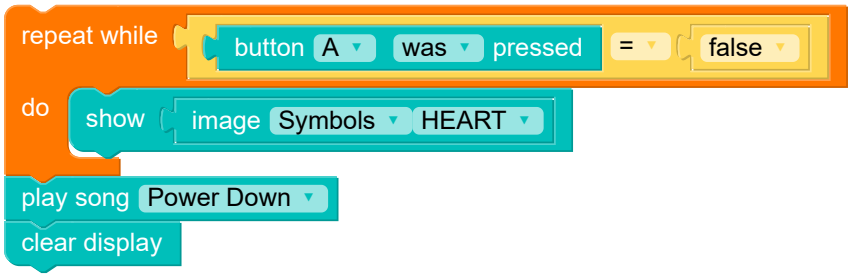
Testing

- That's right!

4.2.7. Problem: What about now?

What will happen when button A is pressed?

Check the blocks and make sure you read the question carefully.



- ○ Nothing will happen.
- ○ Power Down will play.
- ○ Power Down will play and then the screen will clear.
- ○ The screen will clear.

Testing

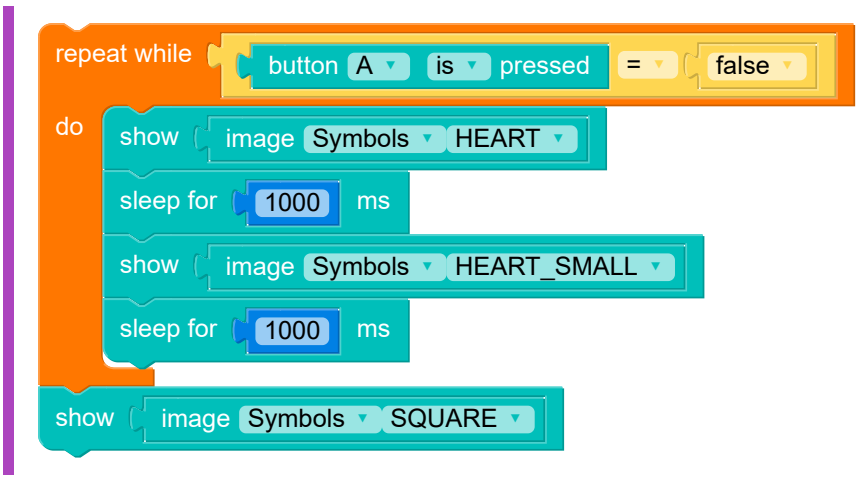
- ☐ That's right!

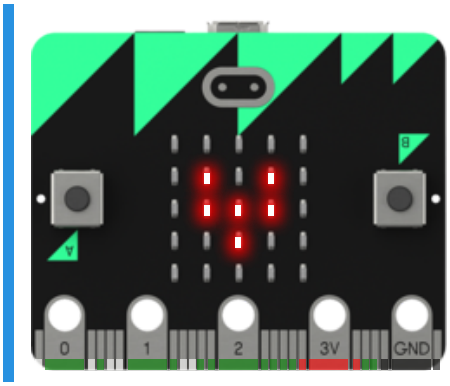
4.2.8. Is pressed? Was pressed?

Do you remember the difference between is pressed and was pressed ?

If you want to make sure your condition is checked at the start of each loop, you must use was pressed .

This is because is pressed only checks if the condition is met for a split second! **If you aren't pressing A in that tiny moment, your program is going to miss it!**





Try tapping button A in this example. **Sometimes the program notices. Sometimes it does not.**

Change is pressed to was pressed and try again.

Now your program remembers if the button was pressed !

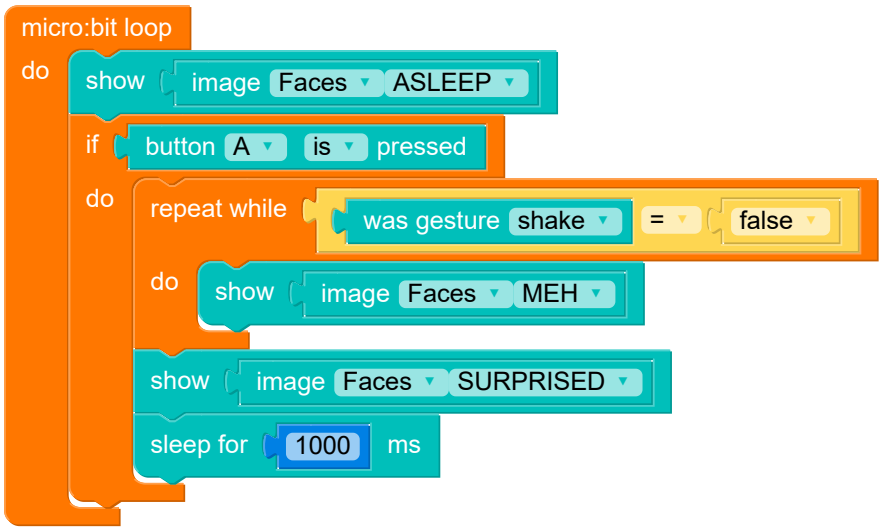
4.2.9. Before and after the loop

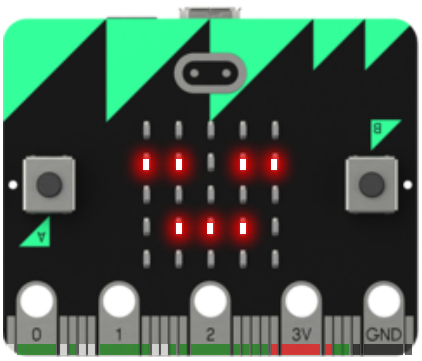
You can have blocks before and after your repeat while blocks if you want.

The program will run the blocks in order from to pto bottom (as long as it isn't going around in a loop). In the example below, we are waking up a sleepy micro:bit.

Once you press button A to wake the micro:bit up, it will show a MEH face until you shake it!

Then it will show the SURPRISED face for 1 second, before falling to sleep again.


micro:bit loop
do
show image Faces ASLEEP
if button A is pressed
do
repeat while was gesture shake = false
do
show image Faces MEH
show image Faces SURPRISED
sleep for 1000 ms


0
1
2
3V
GND

Shake

4.2.10. Problem: Try not to shake on repeat!

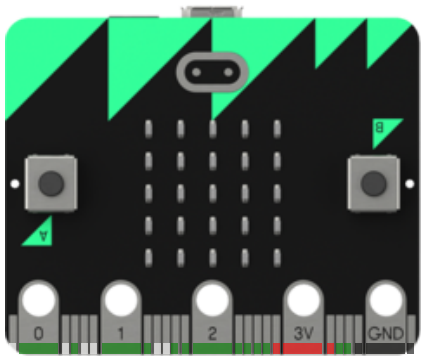
We are going to give our "Try not to shake" game an upgrade! You will program your micro:bit to detect if you shake while you try and run a race!

Press button A to start the game, and when it's started you should show a FABULOUS face.

After was gesture shake happens, show a SKULL 💀, **then** play Punchline .

Press button A to start the game again.

Here's an example for you to try. Press button A to start the game and to reset the micro:bit if you shake.



Shake

When you are finished, you can download your code to a real micro:bit and test your balance.

> **Hint!**
>
> The repeat while block could be useful if you want to wait until something happens.

You'll need

components.json

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that your code starts blank.
- [ ] Testing that the display does nothing if button B is pressed.
- [ ] Testing that your code shows a fabulous face when you press A.
- [ ] Testing that your micro:bit still a fabulous face if no button is pressed.
- [ ] Testing that your micro:bit shows a different face after being shaken.
- [ ] Testing that your micro:bit shows a skull when shaken.
- [ ] Testing that your micro:bit plays punchline when shaken.
- [ ] **Nice work!!** 🏵️🎉😃

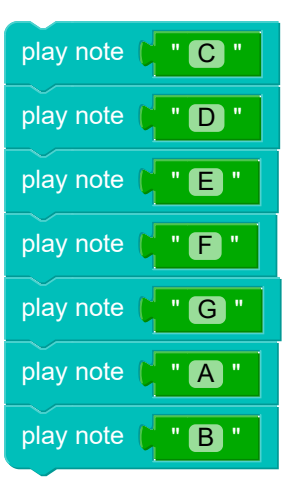
4.3. Musical Notes

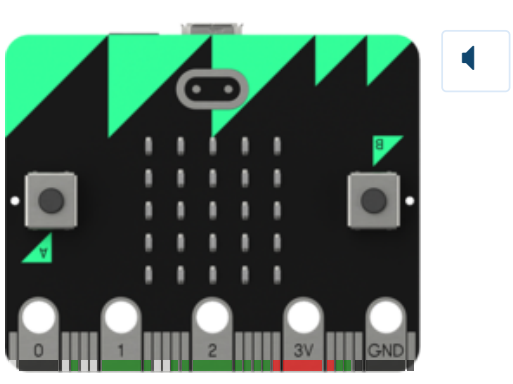
4.3.1. Musical Notes

We have already learned how to play songs. The micro:bit can also play individual notes.

The micro:bit can play all twelve notes of the chromatic scale (C, C# or Db, D, D# or Eb, E, F, F# or Gb, G, G# or Ab, A, A# or Bb).

Try this example:





4.3.2. Detailed musical notes

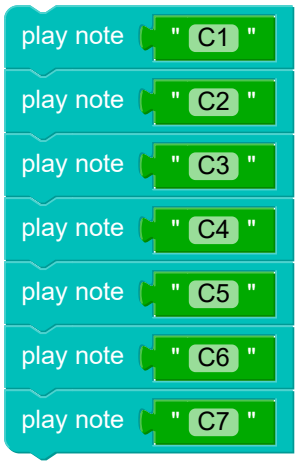
Each micro:bit note also has an **octave**. The octave of a note is the pitch, the higher the number means the higher the pitch.

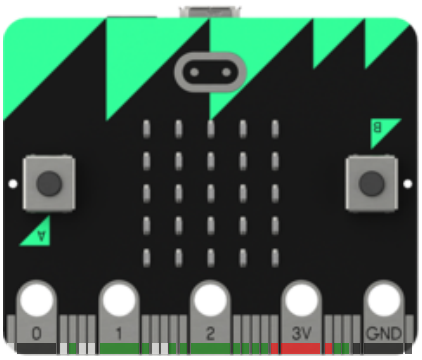
You can program what octave the note is played in by typing " (note)(octave) " .

For example, to play the note D, in the 5th octave: " D5 " .

The micro:bit default octave is 4. That means that any note without a specific number next to it will automatically be played like it has a 4. For example, " F# " will play the same sound as " F#4 " .

You can hear the difference that octaves make in this example:





Guidelines

Octave values can be from 0-9

The notes of the chromatic scale (that you rmicro:bit can play) are: C, C# or Db, D, D# or Eb, E, F, F# or Gb, G, G# or Ab, A, A# or Bb.

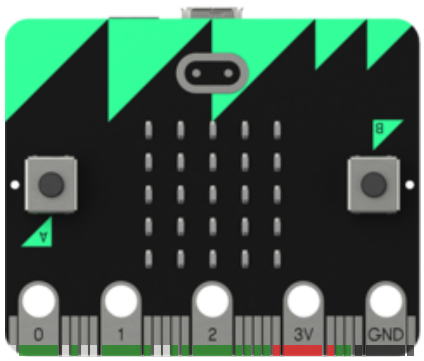
4.3.3. Problem: Musical legs

We are going to use music notes to build a game to test your reflexes and your balance using the micro:bit!

While you're pressing a different button, you should play a different note according to the following table:

Button	Note
button A	C in the 4th octave
button B	F in the 6th octave

Here is an example to try:



Activity!

In pairs, one person will have control of the micro:bit, the other will stand in front of them, they are the balancer.

To start the game, hold one micro:bit button down. The balancer must lift one foot off the ground as long as a note is playing. When the note changes, the balancer must swap feet!

The goal is to try and surprise the balancer with swaps and see how long they can hold their balance. 😆

When the balancing partner loses balance and has to put both feet on the ground, swap roles!

You'll need

components.json

Testing

- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that your micro:bit isn't playing sound when it starts.
- [ ] Testing that your micro:bit plays a sound when button A is pressed.
- [ ] Testing that your micro:bit plays C4 when button A is pressed.
- [ ] Testing that your micro:bit plays a sound when button B is pressed.
- [ ] Testing that your micro:bit plays F6 when button B is pressed.
- [ ] Testing that your micro:bit changes notes when different buttons are pressed.

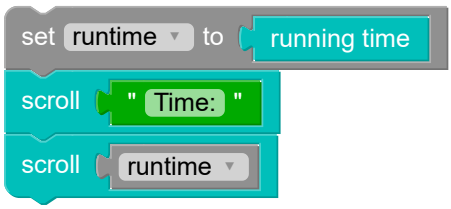
- [ ] Testing that you used is pressed .
- [ ] **Congratulations! You did it! Enjoy balancing** 

4.4. Text and numbers in output

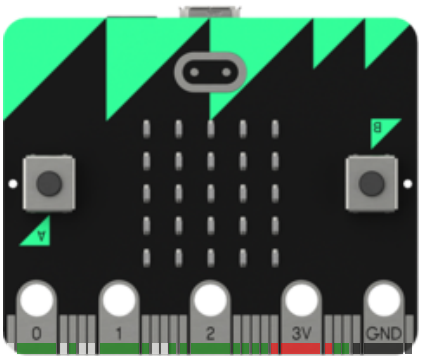
4.4.1. Scrolling text and numbers

Until now we have only scrolled text **or** numbers on the micro:bit. What if we want to scroll both? 👧👧

The example below looks right, but if you run it you will get an error! 🤦




```
Traceback (most recent call last):
  File "__main__", line 6, in <module>
TypeError: can't convert 'int' object to str implicitly
MicroPython v1.7-9-gbe020eb on 2016-09-14; micro:bit with nRF51822
Type "help()" for more information.
>>>
soft reboot
```



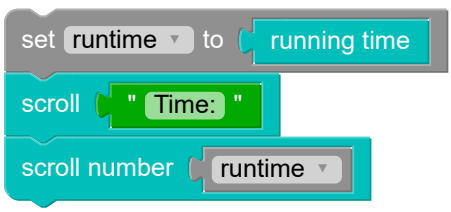
The error says "Line 6 TypeError: can't convert 'int' object to str implicitly"

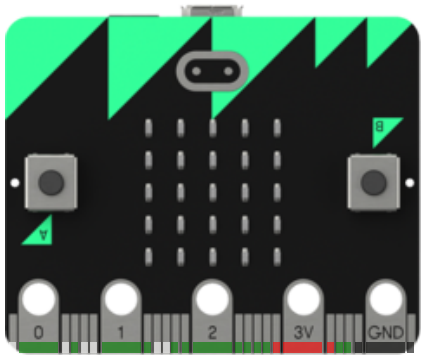
The micro:bit is trying to scroll numbers (the int) in the same way as it scrolls letters (the str) but it doesn't know how ☹.

You have to tell it to use different blocks to scroll different things.

> Remember, numbers and strings are different things to computers (your micro:bit is a tiny computer 💻). You can't tell it to scroll a string and a number with the same blocks.
>
> You will need to use scroll for strings and scroll number for numbers.

Run the example below, just to make sure it works! 😉




B
A
0
1
2
3V
GND

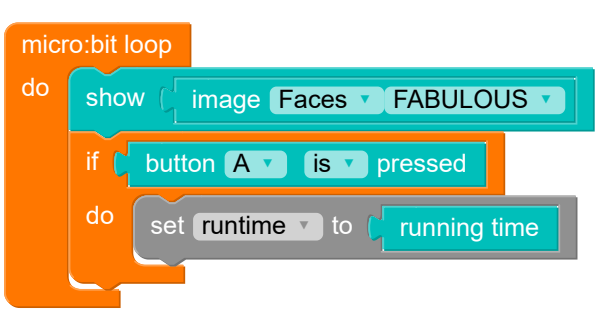
4.4.2. Problem: How to scroll both?

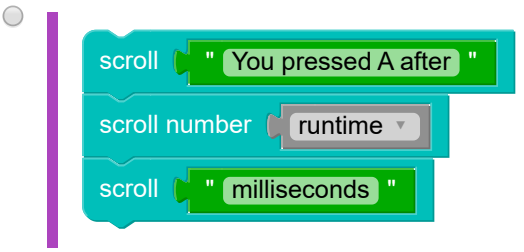
This program is almost complete. It is missing the blocks needed to scroll the final message.

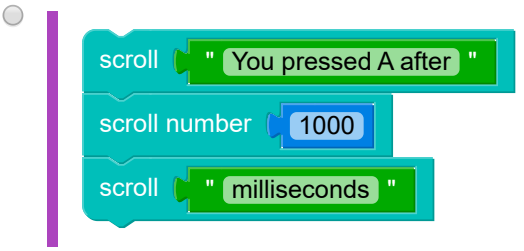
You need to scroll:

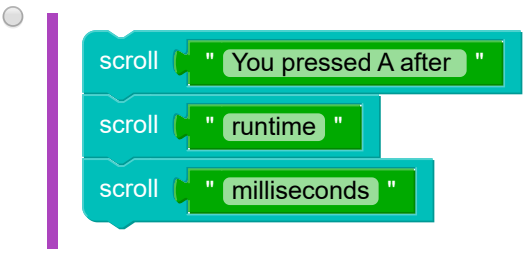
- "You pressed A after"
- then the runtime variable (which is a number)
- "milliseconds".

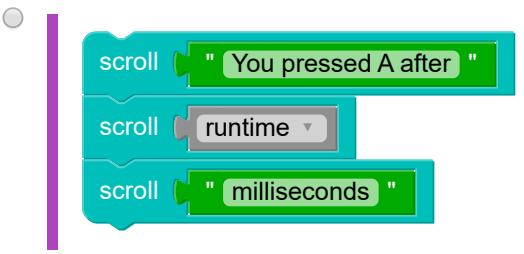
What blocks to you need to add inside the `do` section?











Testing

- [ ] That's right!

4.5. Project Time!

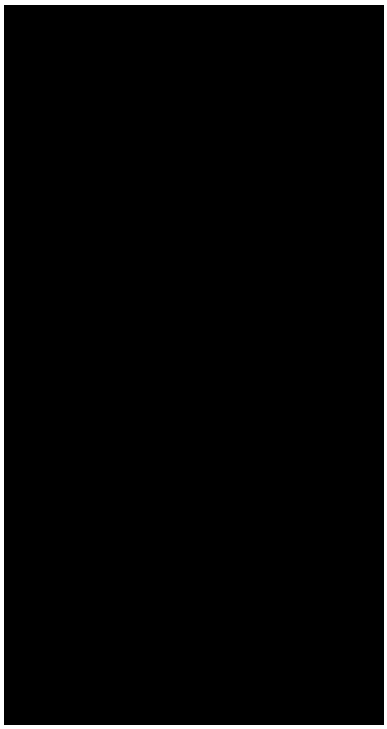
4.5.1. Challenge time!

It's time for the milk bottle challenge!

We are going to put your arms to the test! 💪💪

We want to see how long you can hold up a milk bottle...🥛...Without shaking! 😳😳

This is what the challenge will look like if you have a real micro:bit to use.

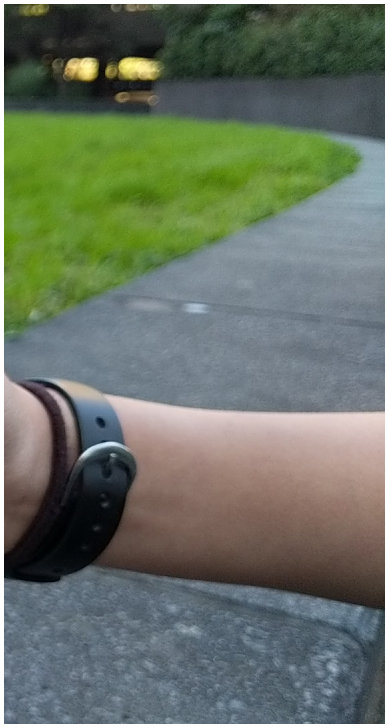


0:00 / 0:07

The Milk bottle challenge in action!

You hold up the milk bottle and press A. The micro:bit will show you a stickfigure symbol while you hold up the bottle for as long as you can. When you can't hold it anymore, the micro:bit will detect you shaking or if you put the bottle down. It will scroll how long you managed to hold up the bottle for.

like this:



0:00 / 0:09

The micro:bit up close.

Let's do it!

4.5.2. Problem: Milk bottle challenge!

We are going to write a program to test your strength! It is called the milk bottle challenge, but you can use it on lots of diffferent objects.

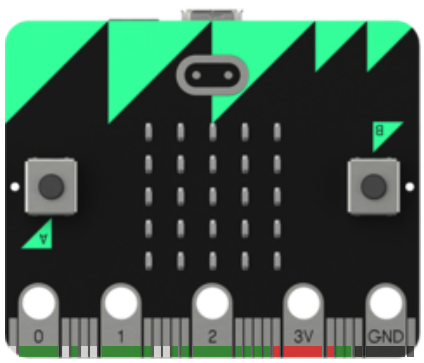
The idea is that you attach a micro:bit to a milk carton full of water, lift it up, then start the timer. Hold it up for as long as you can! When you shake or put it down the micro:bit will stop timing and scroll how long you held up the bottle. It's a test of strength!

Start the timer when button A is pressed, then show the STICKFIGURE image until the shake gesture occurs.

After a shake is detected, save the running time in a variable, play note " A4 ", then scroll number the **elapsed time**, followed by scrolling " ms " .

There are some blocks to get you started, you will need to add more.

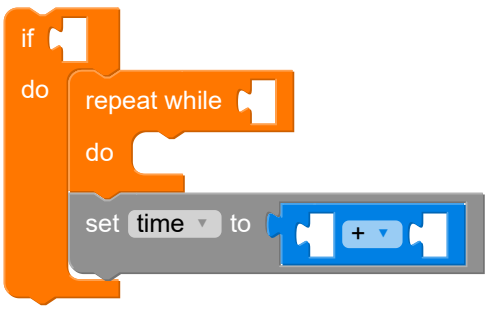
Here's an example for you to try. You will need to press button A to start the program.



Shake

You'll need

program.blockly



components.json

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that your micro:bit starts blank.
- [ ] Testing that your code shows a stickfigure when you press A.

- [ ] Testing that your micro:bit keeps showing a stickfigure as long as you don't shake.
- [ ] Testing that the micro:bit plays the note " A4 " when it shakes.
- [ ] Testing that you scroll the reaction time when the micro:bit shakes.
- [ ] Testing that you scroll ms after the reaction time.
- [ ] **Congratulations! Such strong brain muscles!** 🧠💪🧠💪

4.6. Congratulations!

4.6.1. Congratulations!

You did it! 🍾🍾 You can do the Milk Bottle Challenge!💪

You also learned about:

- The Accelerometer
- The repeat while block and its loops
- Conditions
- Playing musical notes 🎵🎶
- Scrolling words and numbers

Wow! 😃

Great work! Only one more module to go!

In the last module we are going to learn the final pieces we need to use a micro:bit in an egg and spoon race! 🥚🥚🥚🍳

5

NUMBERS AND PIXELS

5.1. Micro:bit calculator!

5.1.1. The final module!

You have made it to the final module! Well done!🎉😀😎

In this module we will be learning the last pieces we need to build the egg and spoon race game.

We will work through some variable loops and learn to make the single LED 'egg'🥚 move around the micro:bit. As well as how accelerometers can tell which way your micro:bit is facing.

Let's go❗❗

5.1.2. The maths Blocks

You have already done some maths using the running time of your micro:bit. Did you know your micro:bit is also a calculator! It can do lots of maths using the blocks in your numbers tab.

These are the mathematical **operators** you can use in your blocks and what they do:

+ - addition

- - subtraction

÷ - division

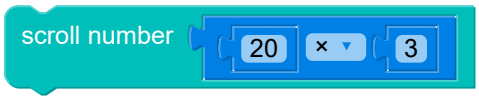
x - multiplication

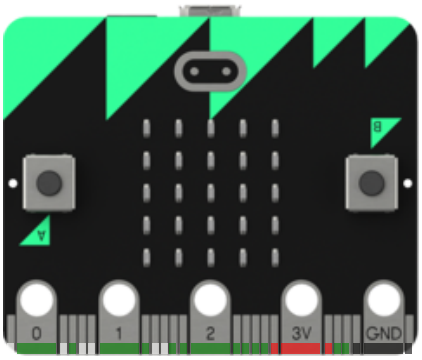
^ - to the power of

5.1.3. Using maths blocks

> 💡 **Don't forget!**
>
> Computers scroll numbers and letters differently so you must use the scroll number block to scroll the answers to equations.

Try this example think of a sum that you might need to do . **You should change the numbers and mathematical operators** and see which numbers print out.





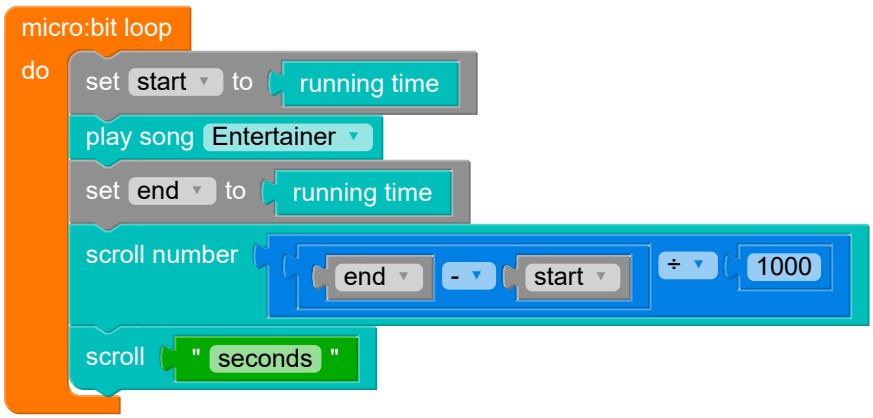
5.1.4. From milliseconds to seconds

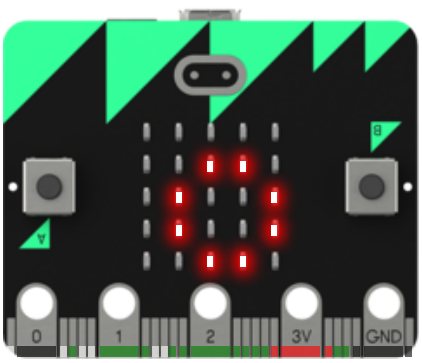
You can use the mathematical operators to scroll milliseconds as seconds.

For example, rather than scroll 1500 milliseconds, we can scroll 1.5 seconds.

To do this, we need to divide the milliseconds by 1000.

The example below shows you how long the song Entertainer runs for in seconds. You can run the example to see the run time.





5.2. Updating variables

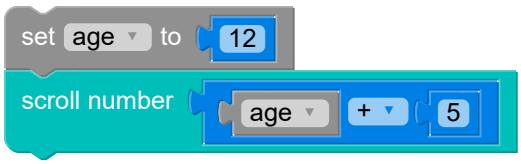
5.2.1. Using variables to do maths

Let's use some mathematical operators to work out how old we will be in 5, 10, 15 and 20 years.

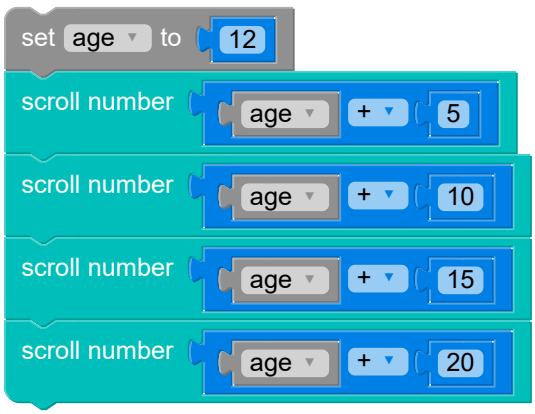
You could type out your age over and over (😫), or you could store it in a variable and use it again and again. Here is how:

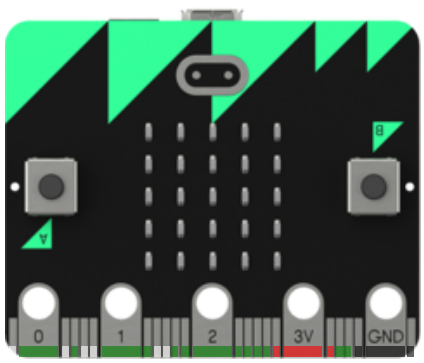
Save your age in a variable (We have used 12 as an example, but you can change it if you want!):

Use that variable in an equation:



The long way to check every 5 years is to build it over and over like this:





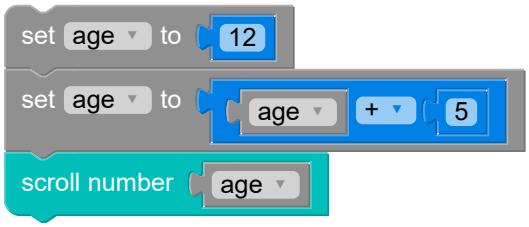
It works. But it's long and boring! And it's going to take *so many* blocks to get to 100... 😫😫😫

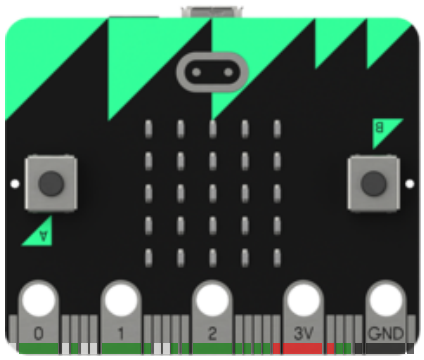
5.2.2. Updating variables

Instead, we can **update the variable** to the new number each time.

For an example age of 12. You store it in the variable `age`. Then we add 5 years and save the variable again. Now, `age` will scroll as 17.

Here it is in blocks:





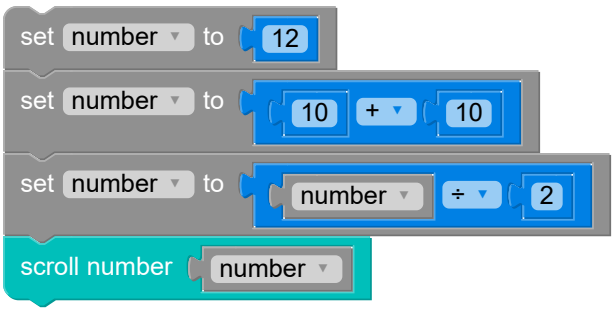
This shows you the value of the variable as each block runs:

Step	Blocks	Value of age
Store the original age in the age variable	set age to 12	12
Add 5 years and save the variable again.	set age to age + 5	age = (12 +5)
The new value of age is 17.	age	17

5.2.3. Problem: Value of number

What is the value of the variable `number` at the start of the code?

What is the value of `number` at the end of the program?



- Start: 12

  End: 10

- Start: 12

  End: 12

- Start: 10

  End: 10

- Start: 0

  End: 6

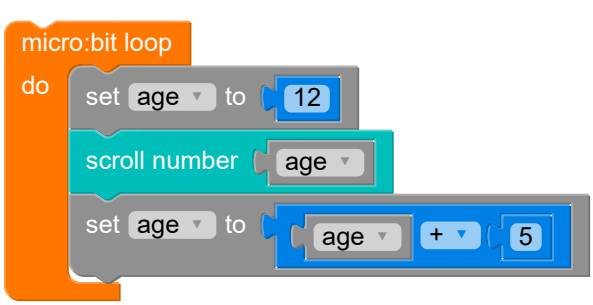
Testing

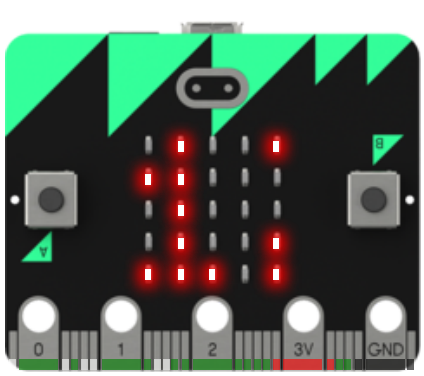
- [ ] That's right!

5.2.4. Looping updates

If we want to keep adding 5 to `age` and scroll the number over and over, how should we do it? 🤔🤔

We could try and put a a loop around the variable update. **Do you think it will work?**





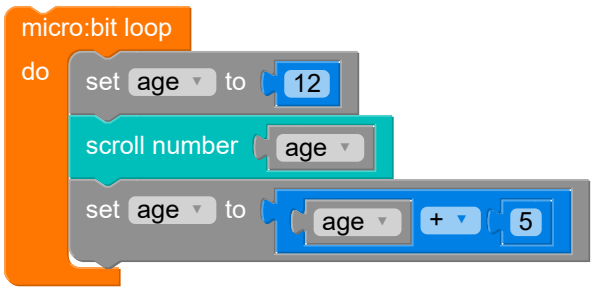
12 12 12 12 ... wait a minute! It's not going up! 😠

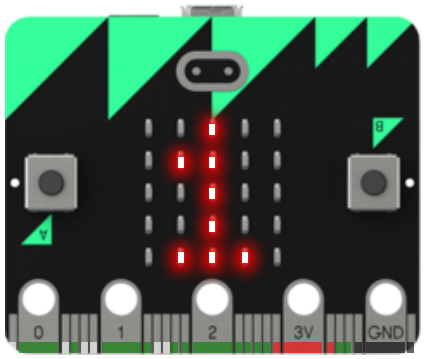
Let's figure out why in the next slide.

5.2.5. Variables out of loops

The problem is that we set `age` to 12 **every time we start the loop again.** So whatever updates we have made get reset.

Follow the code with the table below to see where the reset happens.





Step	arithmetic	Value of age
Set the original age	set age to 12	12
scroll number age	age	12
Add 5 years and save the variable again.	set age to age + 5	(12 +5) = 17
Start the loop again	set age to 12	12!! 😱😠

Let's fix it!

Move the first age block outside of the main micro:bit loop. We have added a notch into the loop for it to fit into.

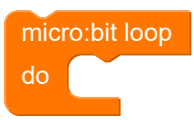
Run the code to see if that fixes our bug 🐛. (A **bug** is a word for a problem in the code)

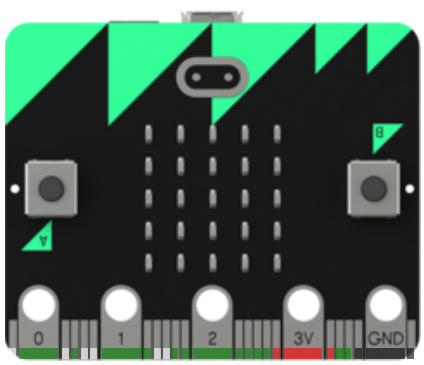
> **Stop the reset!**
>
> To stop the age variable from resetting with each loop, you must start it outside of the main micro:bit loop.

5.2.6. The new notch

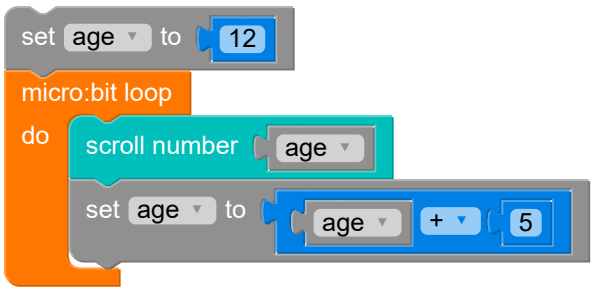
To fix this we needed to move our very first variable **outside of the micro:bit loop**. This will stop the variable from resetting every time the loop runs.

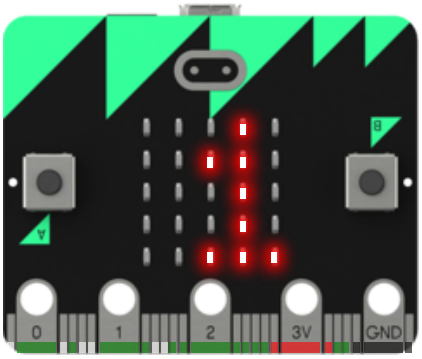
We have upgraded your micro:bit loop so that you can attach variables to the outside. We added a special notch for them!





Now this example is working! 😁 Give it a try!





5.2.7. Problem: 10 Push-ups please

We are going to use the looping method we just learned about to do 10 push-ups!

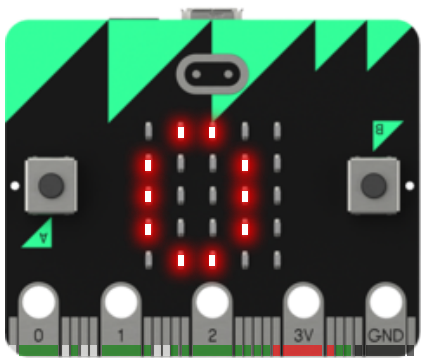
Every time you do a push-up, press button A . When you have done 10 , the micro:bit will congratulate you with a song! (Then you can do 10 more! 😉)

Your program should keep count of how many push ups you've done, and display the count using the show number block. If the count reaches 10 the micro:bit should play the Power Up song, then reset the count.

> **Show not scroll!**
>
> We have given you a new block! The show number block means the number doesn't move. Check out the difference in the example below.

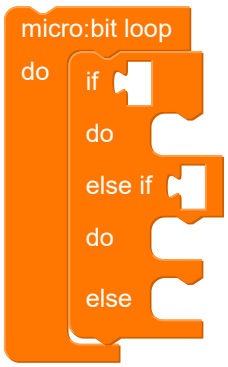
> **No 10?**
>
> Why don't you see the number 10 when you get to it? We are using the show number and it only has space to fit one **digit**. 10 is made of 2 digits, "1" and "0". If we display it the micro:bit will have to flash both. It's not very pretty, so we have left it out. You can always try it out in our code to see what we mean. Try running show number 10 and seeing what the micro:bit does.

Here is the example micro:bit for this question.



You'll need

program.blockly



components.json

Testing

- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that your micro:bit starts at 0.
- [ ] Testing that your micro:bit adds 1 on button A.
- [ ] Testing that your micro:bit can count higher than 2.
- [ ] Testing that your micro:bit sings after reaching 10.
- [ ] Testing that your variable resets to 0 after you reach 10.
- [ ] **Congratulations! You did it!! Now drop and give me 20!** 💪💪

5.3. Individual pixels

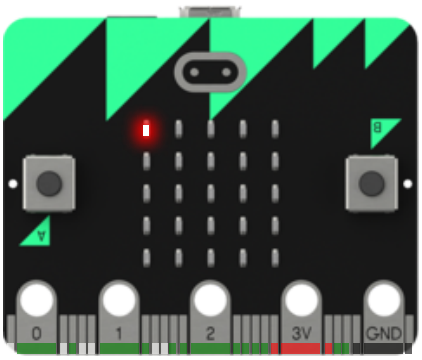
5.3.1. Using individual pixels

We have already seen lots of the pre-programmed images and symbols on your micro:bit. Those images are made by turning specific LEDs on at the same time.

We are going to give you a new block so that you can turn LEDs on one by one.

Tah Dah!📣📣 Here it is:

What do you think it will do when you run it...? What will happen if you change the first 2 numbers to 1 ...?



5.3.2. Pixel parameters

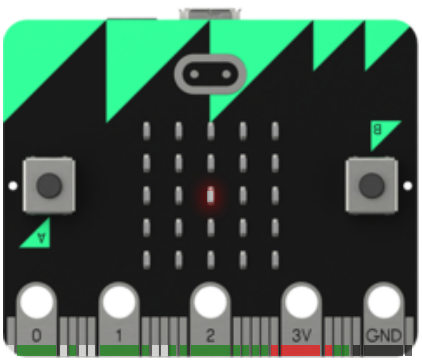
Just like the music notes have different **parameters** (the octave of the note, for example), LEDs have **parameters** too.

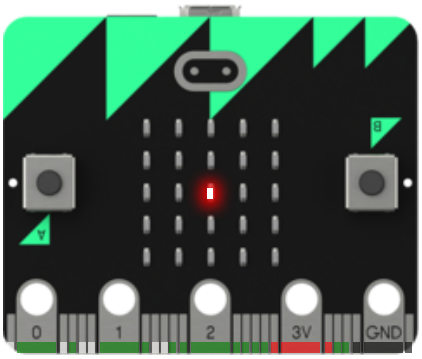
The first 2 numbers tell the micro:bit which LED to light up (more on that in the next slide).

The third number is the brightness level.

0 is off and 9 is the on at the brightest setting.

Run both of these examples. Can you see the difference in the brightness?





5.3.3. Pixel layout

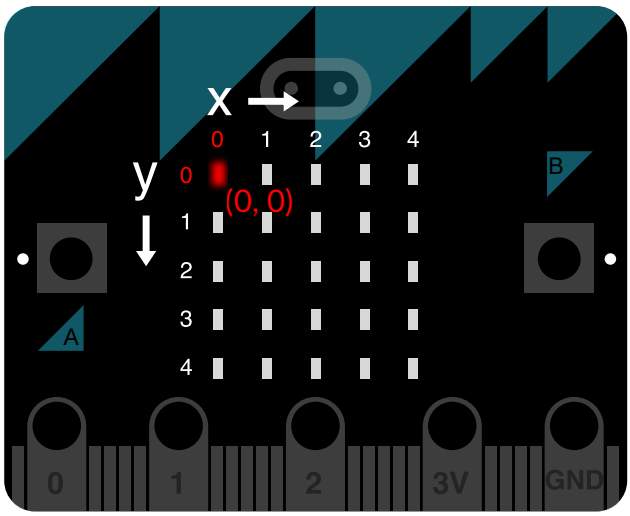
The first 2 numbers are the location **coordinates** of the pixel you want to turn on or off. The first number is the **x coordinate**. The second number is the **y coordinate.**

This is a diagram that shows you the **coordinates** of each pixel. The top, left pixel is in position (0,0). The bottom right pixel is in position (4,4).

Click on the LEDs on the micro:bit below. They will turn on and you will see what their coordinates are!

Count from zero!

Computer scientists start counting from zero! That means that the coordinates of a pixel tell us how far it is from the top-left pixel.

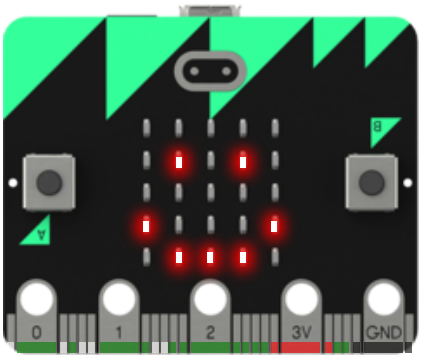



```
display.set_pixel(0, 0, 9)
```

5.3.4. Pixel pictures!

Want to give your happy face a nose?

First, find which pixel should be the nose. Run this example so you can see the happy face. The nose should go in the middle of the face.



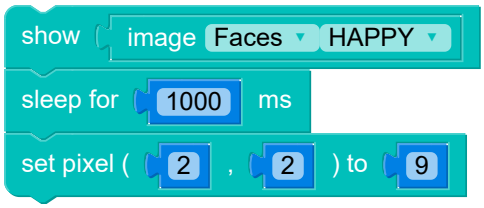
Work out what coordinates the nose should be (x,y). Count x coordinate from the left hand side. Count the y coordinate from the top. **Start at 0**

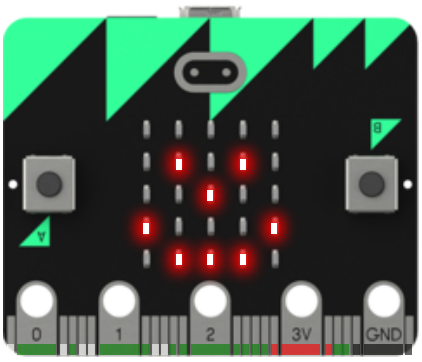
You can also hover over the LEDs with your mouse and the coordinates will appear. Once you have your coordinates, here are the blocks you need:

You show a happy face for 1 second, then turn on the nose, brightness level 9!💡

Try it! Run the code

Did you see the nose appear? 👀





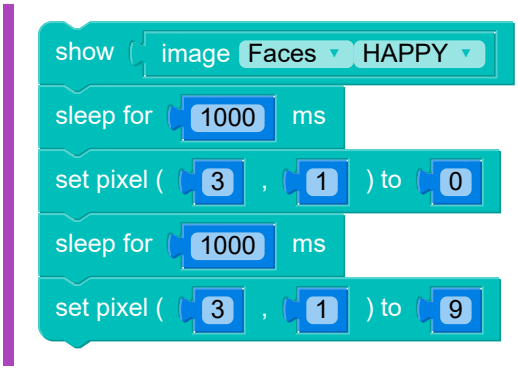
5.3.5. Pixel pictures 2

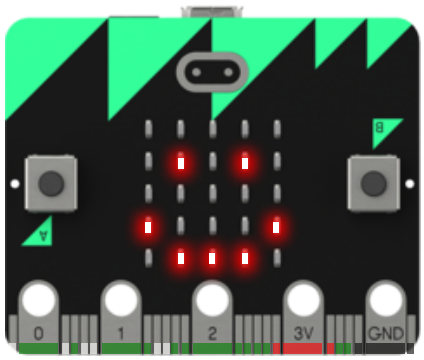
Now let's try and make it wink at us. 😉 **You have to turn a pixel off**

Show the happy face for 1 second. Then, set the pixel that is the right eye 👁 to brightness 0.

Sleep for 1 second and turn the pixel back on (back to brightness 9) again!

Run the code and watch for the wink.


show
image
Faces
HAPPY
sleep for
1000
ms
set pixel (
3
,
1
) to
0
sleep for
1000
ms
set pixel (
3
,
1
) to
9


0
1
2
3V
GND

☺😉☺

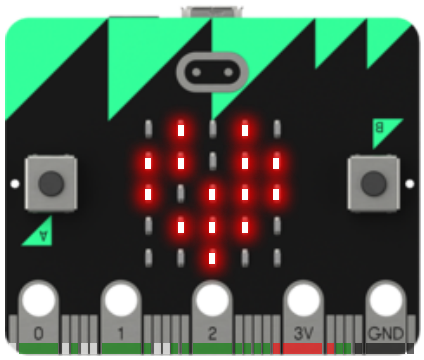
5.3.6. Problem: Broken heart

Oh no! The micro:bit loves you ♡ but you broke its heart! 💔😢

Write a program to show its broken heart.

A broken heart looks like a normal heart missing 2 pixels. Run the example below to see which 2 pixels you need to turn off.

You will need to find the x and y coordinates of those 2 pixels, then remove them from a normal `HEART` image.



Hint: set the brightness to zero

Start with the ♡ picture and turn off two pixels by setting them to brightness `0` .

You'll need

program.blockly

Testing

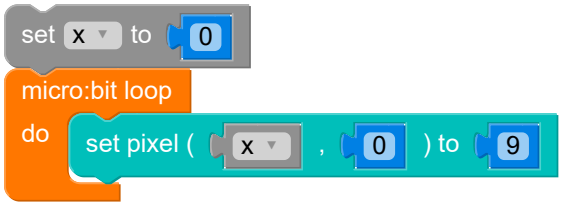
- [ ] Testing that the display is showing a heart.
- [ ] Testing that the pixel at coordinate `(2, 1)` is set to `0`.
- [ ] Testing that the pixel at coordinate `(1, 2)` is set to `0`.
- [ ] Well done! Hopefully the micro:bit can put it back together! 💔💔

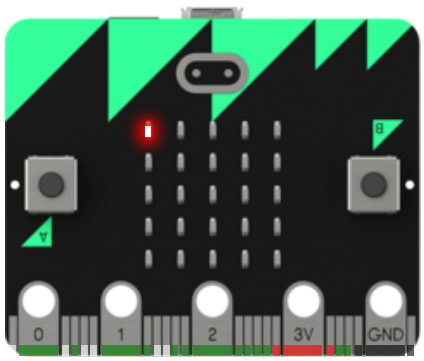
5.3.7. Store pixels in variables

We can use variables to move pixels using the buttons. Let's walk through this example.

First, we need a variable to save the **x coordinate** of the pixel we want to move. Set a variable called x to be 0 (outside the loop).

Now we can use that variable block instead of a number block when we turn on the pixel.

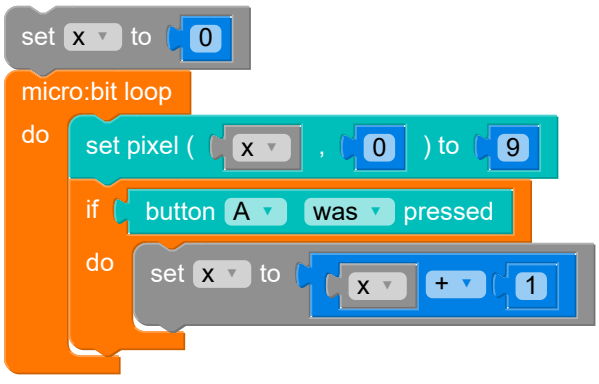


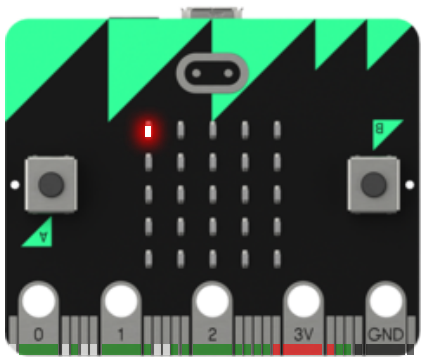


Now we want to make the pixel move. We need to add the button press.

When button A was pressed, we should update the x variable to add 1.

Now our code turns on the pixel with the x coordinate. If button A was pressed, it adds 1 to the x coordinate. Then the loop restarts and we show the new pixel.





Not quite there yet!

If you run that code, each new pixel lights up and the old pixel stays on. We don't want to draw a line. We want to move 1 pixel. We need to turn the old pixel off!

We do that in the next slide. 🛠

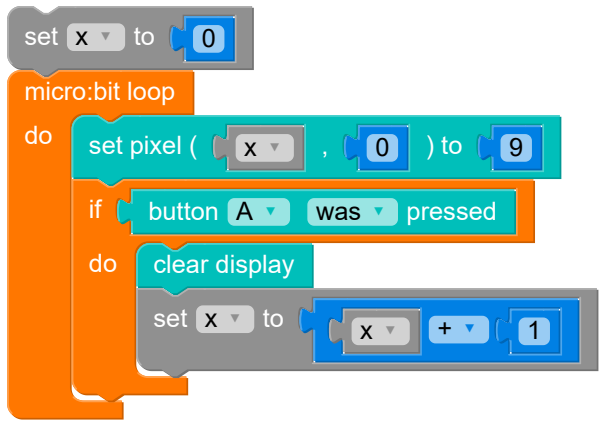
5.3.8. Store pixels in variables 2

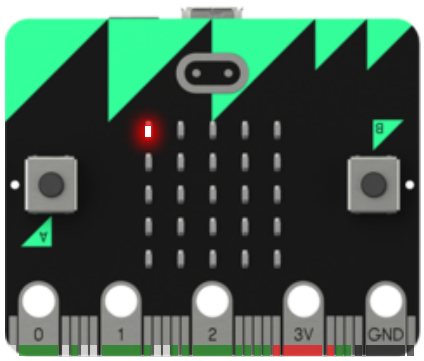
The first thing we want to do if button A was pressed is clear the old pixel. Use the clear display block.

Now our code stores the x coordinate variable. Then it turns on the correct pixel. If button A was pressed it clears the display. Then it adds 1 to our x coordinate, getting ready to show the new pixel.

The loop restarts and the updated pixel coordinates are used to turn the new pixel.

Run this example and press A to move the pixel across the micro:bit.





A new problem!

What happens if you press button A until the pixel goes over the edge of the micro:bit? You get an error that tells you that the micro:bit can't display!

Let's fix that in the next slide. 🛠

5.3.9. Don't go over the edge

Our final step is to stop the pixel from going over the edge of the micro:bit.

In this example, we will make the pixel jump back to position 0 when it gets to the edge.

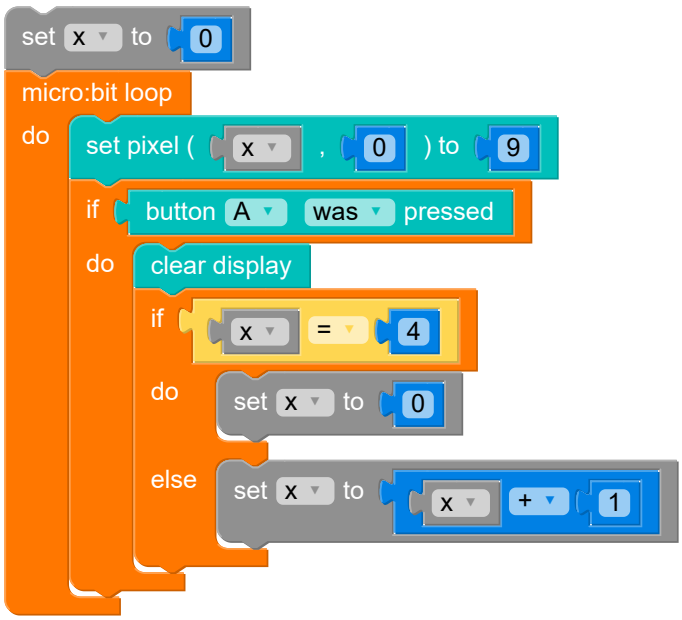
You need to add an if/ else block.

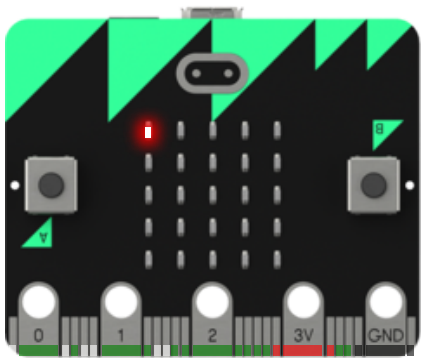
It will check if the pixel goes over the edge. If the pixel does go over, reset the x variable to 0. It is similar to how we counted up to 10 push-ups then reset the count variable back to 0.

The `if/ else` block checks the pixel position using a **condition**. Remember those? This condition is `x` = `4` (the last valid pixel position).

If x = 4, the pixel is going to go over the edge next time we loop! We reset x to 0 to send it back to the start of the micro:bit.

Else, the pixel isn't going to fall off and we can add 1 to the x value.





Woohoo! We did it! The pixel is zooming across the micro:bit!

5.3.10. Problem: Take a pixel for a walk

Write a program that moves a pixel around your micro:bit as you press button A and button B .

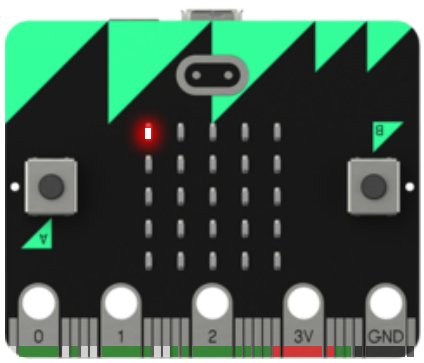
It's an upgrade to moving the pixel across the micro:bit in the previous slides.

The pixel should start in the top left hand corner of the screen. If you press button A , the pixel should move to the right. If you press the button B it should move down. If it's about to go off the screen it should loop back to the opposite edge! You can use the previous slides as a guide to help.

Make sure the pixel is at maximum (9) brightness!

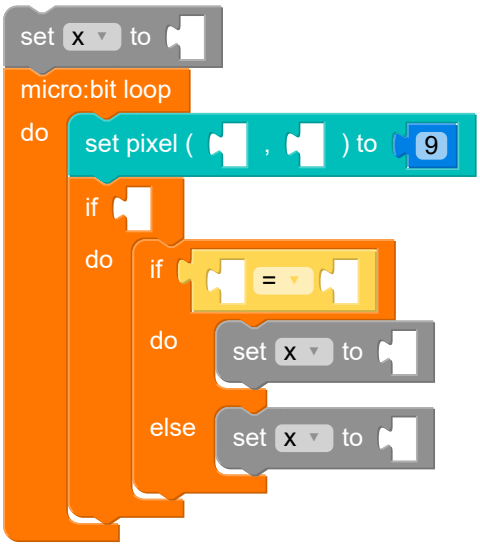
> **Hint**
>
> In the last slide you needed to use one variable to track the x coordinate. How will you track both the x and y coordinates?

Here is an example, press button A and button B to move the pixel around.



You'll need

program.blockly



components.json

Testing

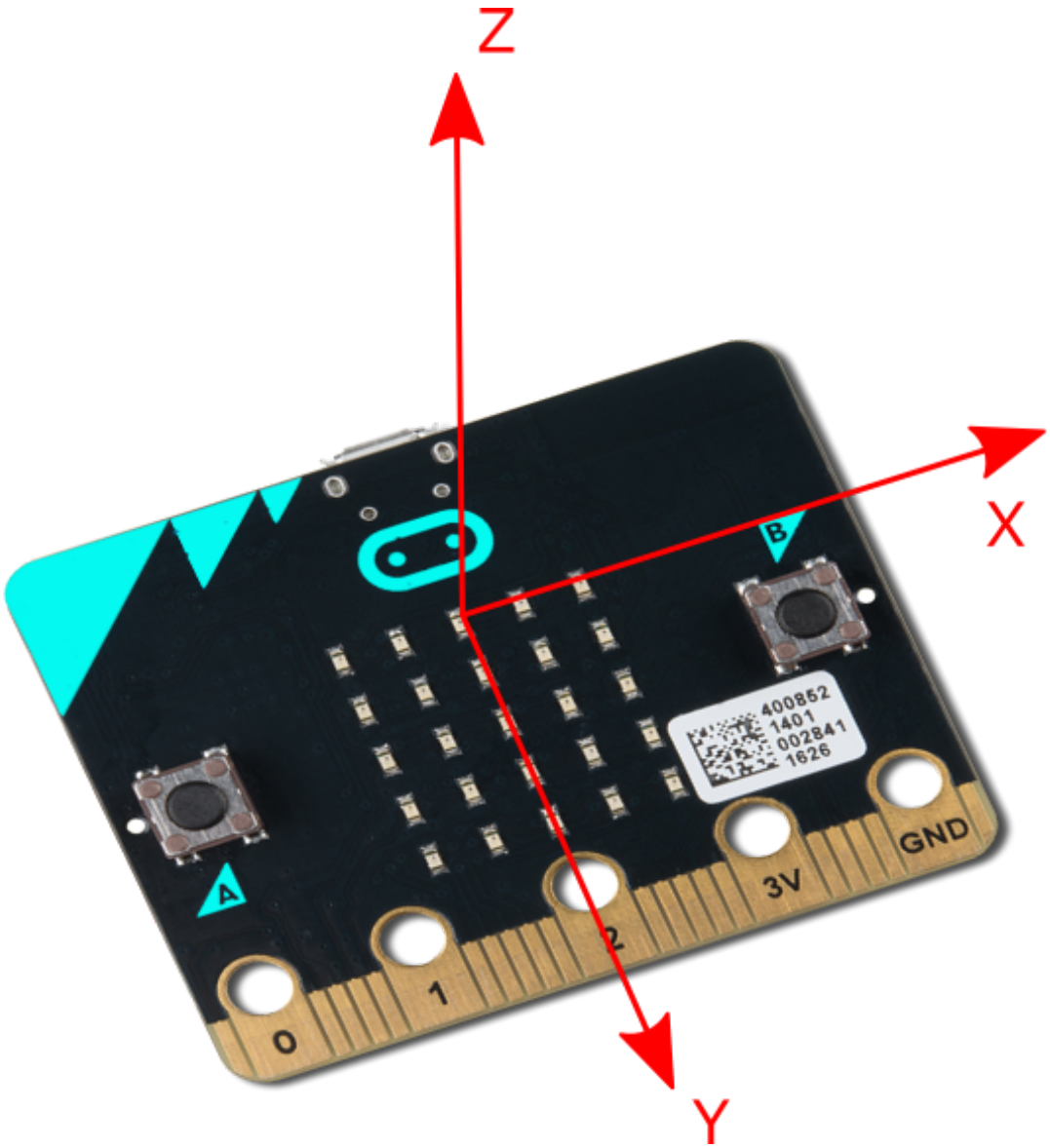
- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that your pixel starts at the (0,0) position.
- [ ] checks for both is pressed and clearing the display in one. Also catches out of bounds value errors if they've used 'is pressed'
- [ ] Testing that your micro:bit moves to the second correct x position.
- [ ] Testing that your micro:bit resets the x axis.
- [ ] Testing that your micro:bit moves to the correct y position.
- [ ] Testing that your micro:bit moves to the second correct y position.
- [ ] Testing that your micro:bit resets the y axis.
- [ ] **Congratulations!! You did it!** 🕷🍕🕷🍕

5.4. Accelerometer X and Y

5.4.1. Accelerometer X and Y (and Z!)

You have already used the accelerometer in your micro:bit. Remember checking for shakes? The accelerometer works with X and Y coordinates too. It also has a Z coordinate! It uses them to see how far the micro:bit has tilted in any direction.

In this diagram, you can see which axis is measuring which movements.



The different accelerometer axes.

For example, when you hold your micro:bit up to your face to see the LED's, it will register the Y axis getting larger. As you tilt it flat onto a desk, the Y axis gets smaller.

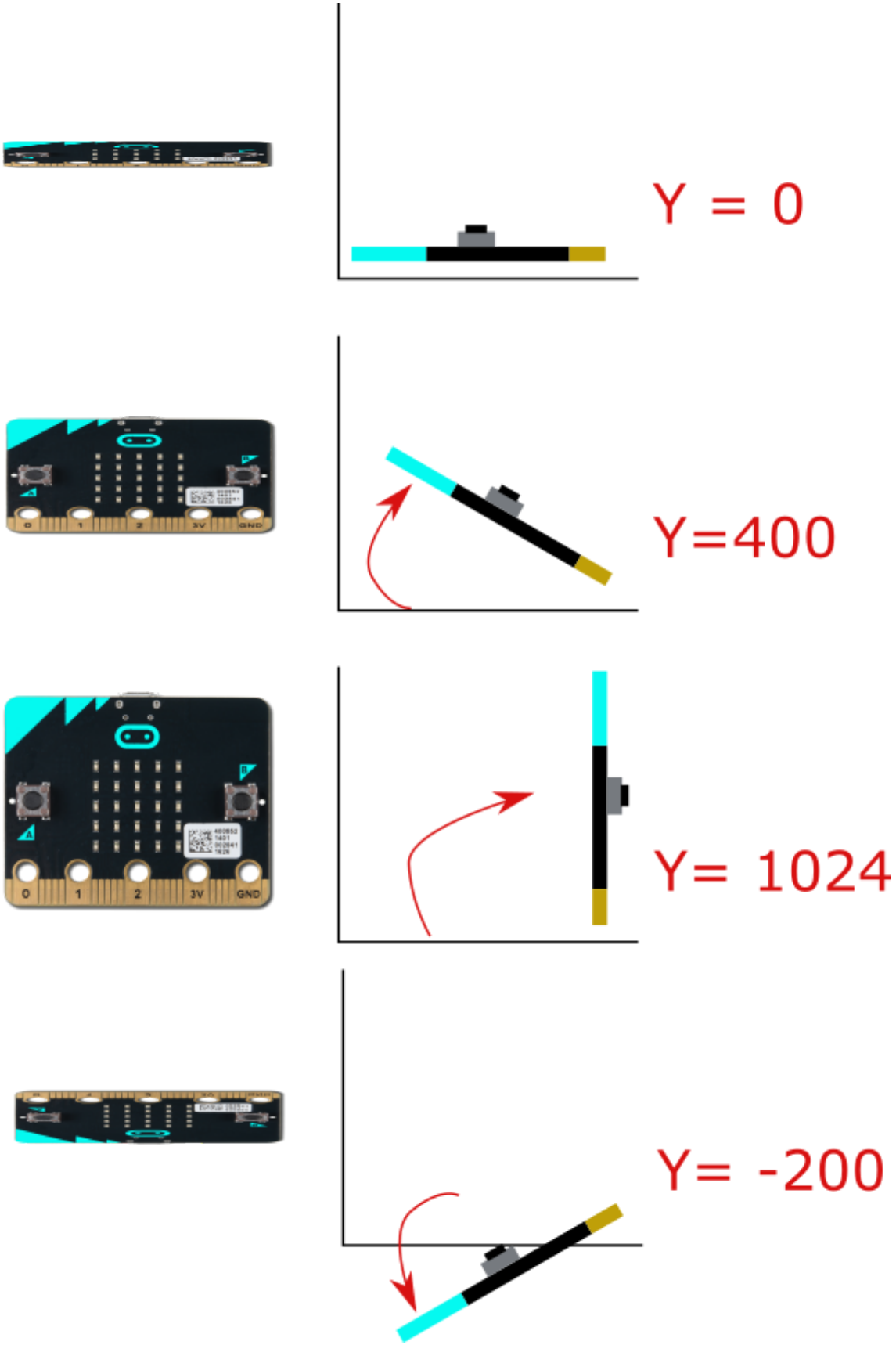
5.4.2. How accelerometers work

The axes register as 0 when the micro:bit is laid completely flat, and not moving. (except for Z, it's a special case but you don't need to know much about it))

As the micro:bit tilts one way or another, axis values go up and down. For example, as you tilt the micro:bit up to look at it, the Y axis values go up. If you tilt the micro:bit back down, the numbers go down.

The numbers can go into the negatives once they go past 0!

The diagram below will help you to get an idea. The numbers are an approximate guess for what your micro:bit might read.

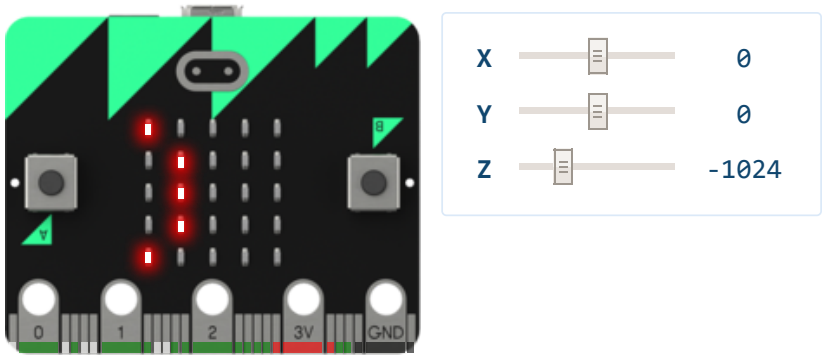


The Y axis in action.

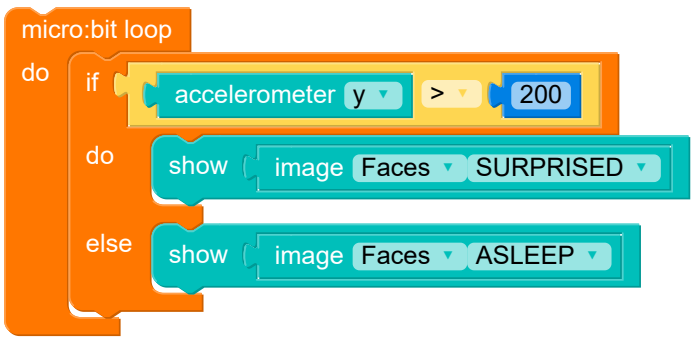
5.4.3. How to use them

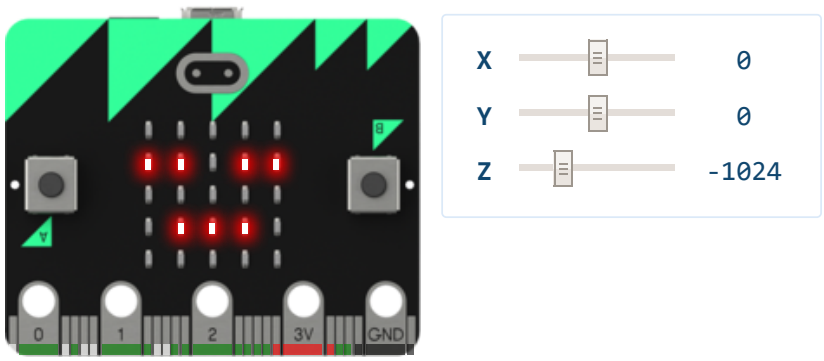
You can get the specific measurements of each accelerometer axis by using this block and selecting the axis you want.

The micro:bit below will scroll the value of the axis you choose. It is currently measuring the y axis. You can change the angle of the micro:bit using the sliders and see how the numbers increase or decrease.



You can also use the accelerometer to make something happen when your micro:bit tilts a certian way. **This micro:bit will show a surprised face when it tilts too far back.**





5.4.4. Problem: Tilt

Let's use the accelerometer axis to make a balancing game! **We will only measure the Y axis.**

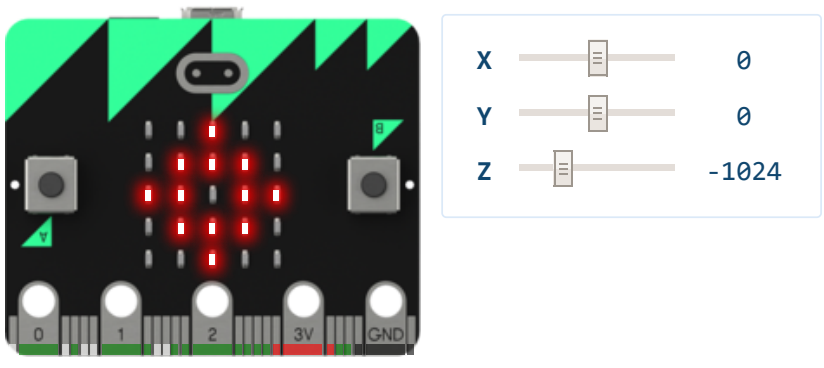
You should hold your micro:bit steady, if you tilt too far forward, it will use arrows to tell you to tilt back. If you tilt too far backward, it will use arrows tell you to tilt up.

This should work according to the following table:

y value	Image	LED output
> 500	ARROW_S	
< -500	ARROW_N	
No tilt	TARGET	

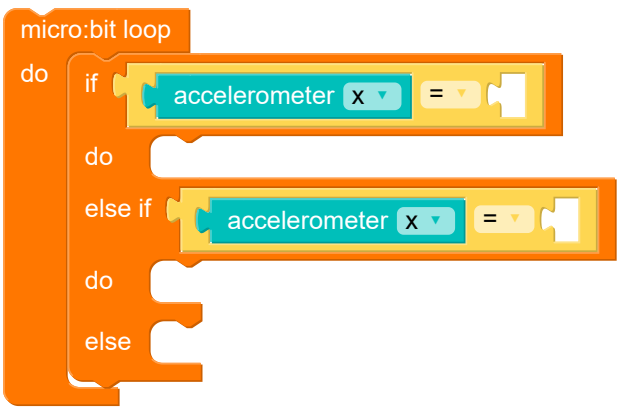
Don't forget to loop forever!

Here is your example to try. Move the sliders about to tilt the micro:bit.



You'll need

program.blockly



components.json

Testing

- [ ] Checking that your code contains an infinite loop.

- [ ] Testing that your code shows a target when y=0 and x=0.
- [ ] Testing that you program shows a South arrow if y is set to 501.
- [ ] Testing that you program shows a North arrow if y is set to -501.
- [ ] Testing a target is shown when y is set to 500.
- [ ] Testing a target is shown when y is set to -500.
- [ ] Testing that your micro:bit shows a South arrow when y > 500.
- [ ] Testing that your micro:bit shows a North arrow when y < -500.
- [ ] Testing many different arrows.
- [ ] **Hooray!! Good working using conditions!** 🎉 👍

5.4.5. Problem: Tilt both ways

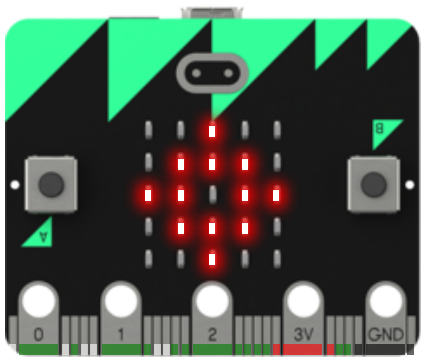
Upgrade your balancing game! Now you want to measure if the micro:bit tilts on the y axis **and** the x axis.

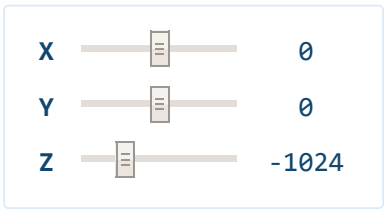
Your program should use both x and y accelerometers measurements and show the correct arrow, or the target if there is no tilting.

accelerometer value	Image	LED output
y > 500	ARROW_S	
y < -500	ARROW_N	
x > 500	ARROW_E	
x < -500	ARROW_W	
No tilt	Target	

Don't forget to loop forever!

Here is your example to try:





Tilting all around

Can you think of a way to tilt your micro:bit so that it is showing the wrong arrow?

If you tilt the micro:bit backwards AND to the left, it can only show one arrow at a time. How would you fix this?

You can try to build a fix in the blockly playground at the end of this course!

You'll need

program.blockly

components.json

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that your code shows a target when y=0 and x=0.
- [ ] Testing that you program shows a South arrow if y is set to 501.
- [ ] Testing a target is shown when y is set to 500.
- [ ] Testing that you program shows a North arrow if y is set to -501.
- [ ] Testing a target is shown when y is set to -500.
- [ ] Testing that you program shows a East arrow if x is set to 501.
- [ ] Testing a target is shown when x is set to 500.
- [ ] Testing that you program shows a West arrow if x is set to 501.
- [ ] Testing a target is shown when x is set to -500.
- [ ] Testing many different values.
- [ ] **Great work! Tilting every which way like a pro!** 💪🔼🔽⬅️➡️👍

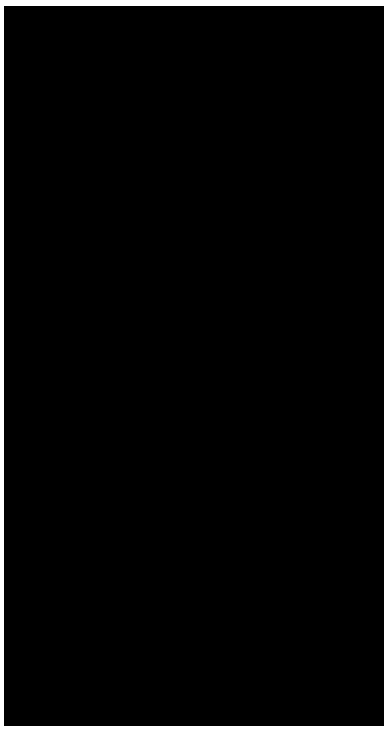
5.5. Final project

5.5.1. Final project

Congratulations! You have made it to the final project!! 🎉🎉🎉🎉🎉🎉🎉🎉🎉🎉🎉🎉

It's time to bring it all together and build an egg and spoon race micro:bit!

This is what you are building:



0:00 / 0:07

Egg and spoon micro:bit up close.

When you are finished, you will be able to use the micro:bit to run an egg and spoon race with your friends.

Have you ever played an egg and spoon race? 🥚🥄

The goal is to run the race while holding an egg on a spoon. If you drop the egg 😬, you must start again. It's a game of balance and speed. We will turn the mciro:bit into the egg and spoon. 🥚🥄If you tilt the micro:bit too far you "egg" pixel will fall off and you will have to start again!

Ready? Let's go!

5.5.2. Problem: Egg and spoon race

You micro:bit will show you the "egg" using a pixel, as you tilt the micro:bit the "egg" will move around. Tilt it too far and it will show a skull 💀 and you will have to start again!

We have given you some blocks to help you get started. **We are bringing together lots of what you have done before. You can do it!** 💪🤩

Start by saving the x and y coordinates of your pixel "egg". The egg should start in the middle of the micro:bit (2) and your variables should be outside of the main loop so that they don't reset.

Now it's time to work inside the micro:bit loop. In one big condition at the top, check that the egg doesn't go off any of the micro:bit sides. You should check that the coordinate variables are not out of the bounds of the micro:bit. If any of those conditions are met (for example if egg_x = 5) then the egg has fallen off and you show a skull for 2 seconds. **it is important that you reset the coordinates back to the middle of the micro:bit after showing the skull so that you can keep playing the game.**

If the egg hasn't fallen off, clear the old pixel and show the pixel at your coordinate variables. **You must also sleep for 300 milliseconds. This will give you some time to re-balance your micro:bit.**

Now that we know the egg has not fallen off, we need to check if the micro:bit is tilting. If it tilts too far we need to move the egg. 🥚

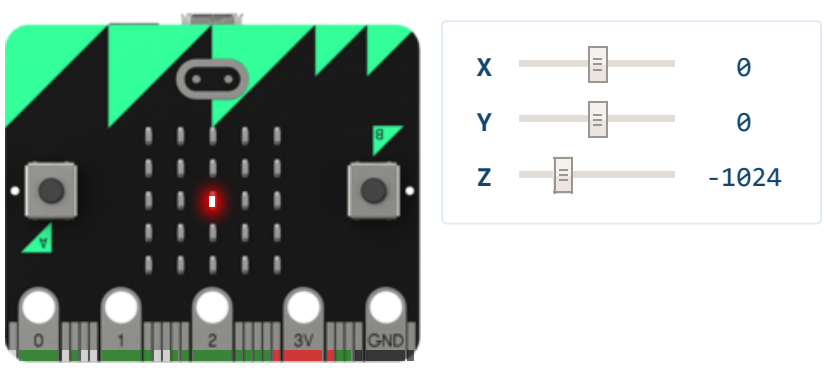
Tilt direction	Condition	Variable change
back	> 200	y + 1
forward	< -200	y -1
no tilting y	--	y = y
right	> 200	x + 1
left	< -200	x - 1
no tilting x	--	x = x

All done!

Here is an example micro:bit that you can use to try the game out. When you are finished you can download the code to a real micro:bit and race your eggs back and forth! 🥚🏇🏇

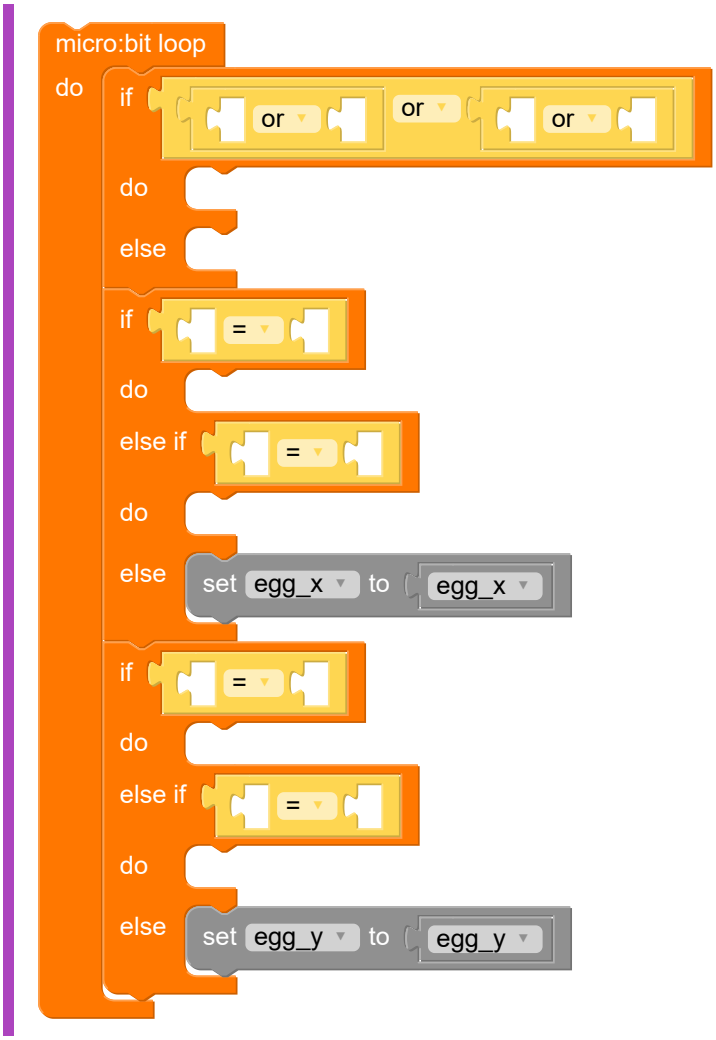
After you have marked:

If you want to make the game harder you can change the 200 in the if blocks to a smaller number. This will make your micro:bit more sensitive to tilting and require more balance from you.



You'll need

program.blockly



components.json

Testing

- [ ] Testing that your code contains an infinite loop.
- [ ] Testing that your egg starts in the centre.
- [ ] Testing that your egg doesn't move without tilts.
- [ ] Testing that the egg stays when the tilts are small.
- [ ] Testing that the x variable increases when accelerometer x > 200.
- [ ] Testing that you clear the previous pixel.
- [ ] Testing that the x variable decreases when accelerometer x < -200.
- [ ] Testing that you clear the previous pixel.
- [ ] Testing that tilting too far right for too long shows a skull.
- [ ] Testing that the y variable increases when accelerometer y > 200.
- [ ] Testing that you clear the previous pixel.
- [ ] Testing that the y variable decreases when accelerometer y < -200.

- [ ] Testing that you clear the previous pixel.
- [ ] Testing that tilting too far back for too long shows a skull.
- [ ] Testing that tilting left and right moves the egg both ways.
- [ ] Testing that your code resets after skull for 2 seconds.
- [ ] **You did it!!!! You completely finished the challenge!! Hooray!** 🎉🎉🎉🥚🥄😀

5.5.3. Congratulations!

Congratulations! You have finished the whole challenge!! 🏆

You have learned so much about using the micro:bit and some ways of keeping fit and practising balance! That's loads.

Here is a list of some of the things you can do:

- Show images 🐓
- Scroll words and numbers 🆒
- Play music 🎼🎶
- Play musical notes 🎶
- Do maths ➕➖➗✖️
- Check running times 🏃
- Balance micro:bits 🤖🤖
- play an egg and spoon race using a micro:bit! 🥚🥄

Amazing work! Well done, and we hope to see you in another challenge soon! 🥕🍇

5.6. Blockly Playground

5.6.1. Problem: Blockly micro:bit Playground

This is a blockly playground! You can use the blocks to build whatever you like!

Here are a couple of project ideas to get you started:

3, 2, 1, egg run!

Add a " "3, 2, 1, Go!" " count down before you run you run and show you your time when you finish.

- You will need to press a button to start the countdown before you start your egg and spoon code.
- After you scroll " "3, 2, 1, Go!" " save the running time in a variable.
- when you finish the race, press a button to scroll your time using the calculations from before

Count the push-ups!

- Can you use the accelerometer to detect when you do a push-up?
- Can you count to 10 push-ups then sing a song to congratulate yourself?

Save all the times

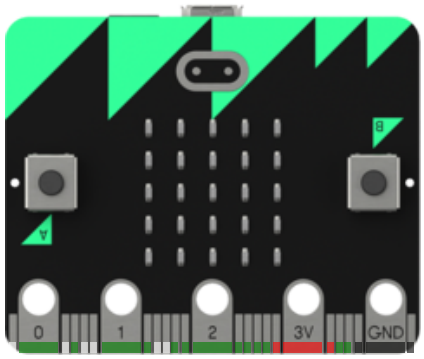
Use your micro:bit like a smart stopwatch! You can save the finish time of runners as they finish, then scroll all the times at the end. (We have already added the variables you will need for this problem)

- This one is a bit tricky and requires you to use the list blocks
- start by setting a variable called time to an **empty list** (this block [ ])
- when you press button A you start the stopwatch by saving the running time
- you should also show a happy face so you know the timer has started.
- every time you press button B it should save the finish time (running time - start, that you have done before) to the list
  - then you need to append the finish time variable to your time **list**
  - then you should append " ms " to the sime so that yo can tell when one time ends and the other time finishes.
- when you press button A and button B at the same time, you scroll what times you have saves in the list.
- To do that, use for each item i in times
- In that do space you should scroll the number i

These are just suggestions! You can build your own micro:bit game and play with your friends!

Play stepping games, balancing game, speedy games and guessing games! Have fun!

Here is an example micro:bit for the **save all the time** project.



You'll need

components.json

program.blockly

Testing

- [ ] **This is a playground question! There is no right or wrong!**