Australian **Computing** Academy

DT Mini Challenge

Intro to micro:bit (Blockly)

1. Displaying images and text
2. Buttons and gestures

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

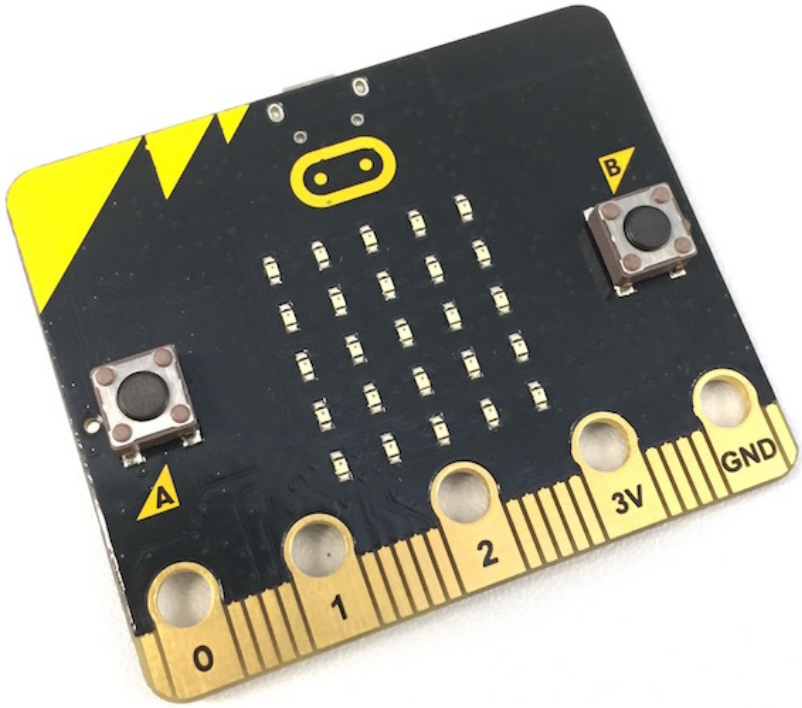
1

DISPLAYING IMAGES AND TEXT

1.1. Getting started

1.1.1. 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!

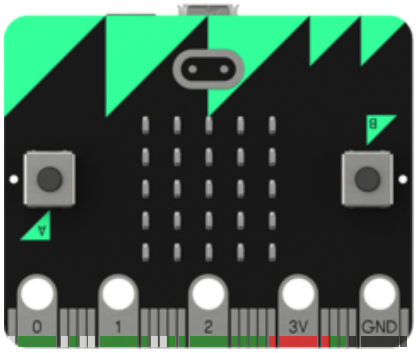
> **If you don't have a real micro:bit...**
>
> You can still do this course. We have a *simulator* which works like a real micro:bit.

1.1.2. Hello, micro:bit!

Let's run a program on the micro:bit!

ag the image block into the hole in the show block.

ck ▶ to run the program. It shows a happy face!



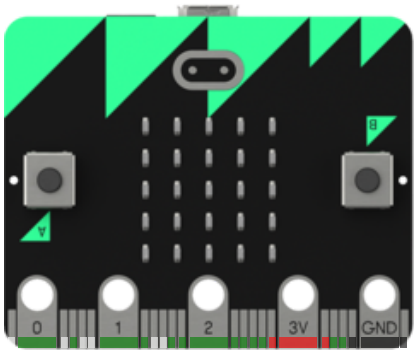
1.1.3. Change the face!

Let's make a different image display!

ag the image block into the hole in the show block.

ange the image to anything other than HAPPY

Run it to show *your* image!


B
A
0
1
2
3V
GND

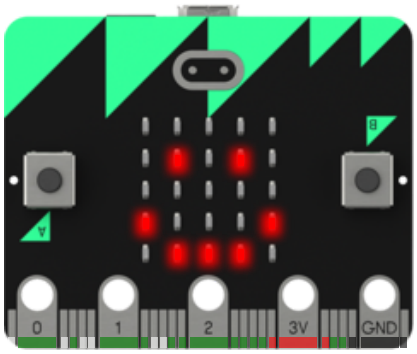
1.1.4. Problem: Happy micro:bit!

Make your own micro:bit program to show a happy face.

n the blocks in the problem editor together.

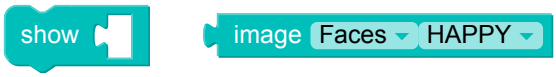
ck the Run button.

ck the Mark button and then **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!**

1.1.5. Problem: Your own Virtual Pet!

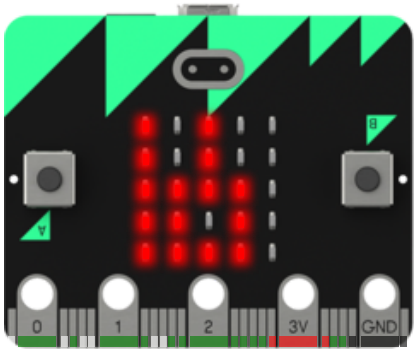
Let's create a virtual pet rabbit!

ck on micro:bit and drag the show block into the workspace.

nnect the blocks to complete the program.

and Mark your program.

Your virtual pet will look like this (click ▶ to play it):



You'll need

program.blockly

Testing

- [ ] Testing that the display is showing a rabbit face.
- [ ] **Awesome! You've made your virtual pet!**

1.1.6. Downloading

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

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

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

1.2. Writing micro:bit programs

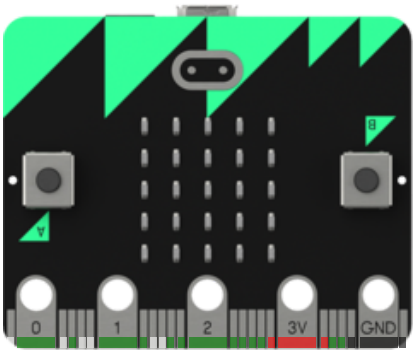
1.2.1. HAPPY to SAD

Let's change the micro:bit from HAPPY to SAD.

ag the image into the hole in the show block.

ange the HAPPY face to be a SAD face.

ck ▶ run to show a SAD face.



Click ↰ to go back to the old program.

Play with the examples!

You can change **every** example and see new programs!

1.2.2. Problem: From micro:bit with love

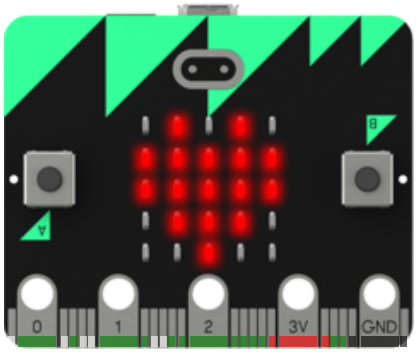
Let's make a micro:bit Mother's Day card!

ag the show block into the workspace.

n the blocks together.

oose the Symbols HEART

in and Mark your program.



You'll need

program.blockly

image Animals BUTTERFLY

Testing

- [ ] Testing that the display is showing a heart.
- [ ] **Congratulations, what a great gift!!**

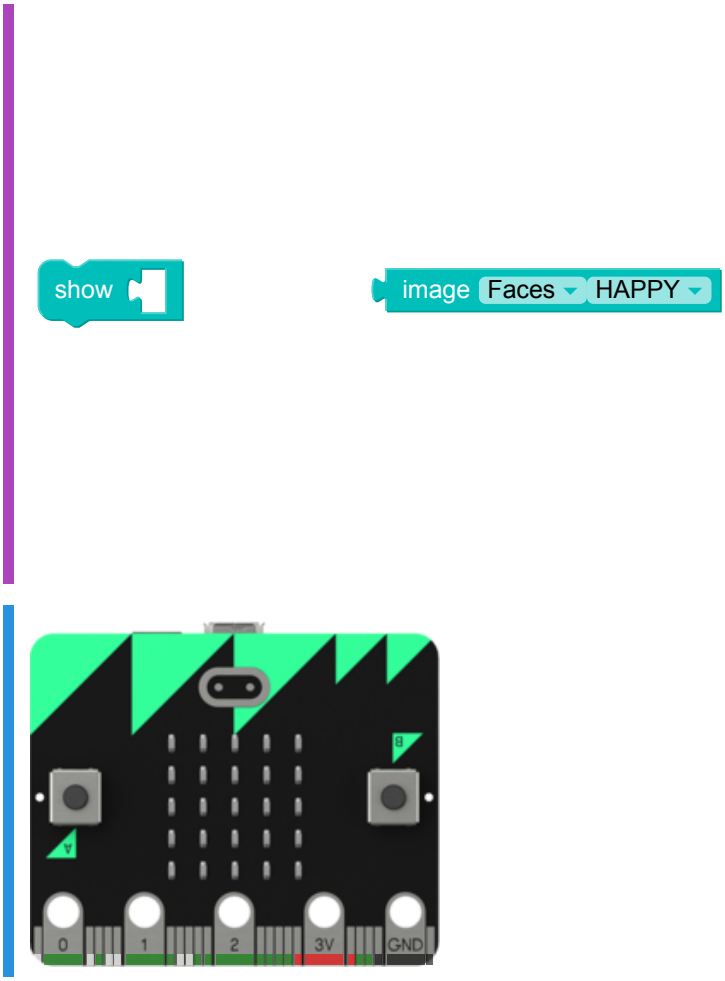
1.2.3. All the animals!

Let's pick a pet!

ag the image block into the show block.

oose Animals from the Faces dropdown.

k your favourite animal and then run your code!



1.2.4. Problem: Pick a pet!

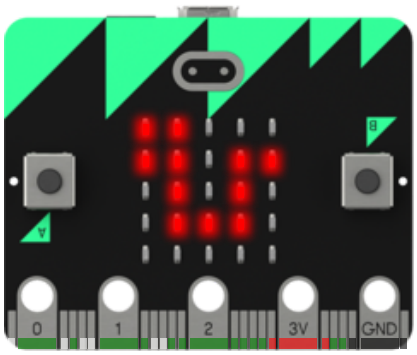
You're ready to choose your own pet now!

t the show and image blocks and connect them.

oose your favourite pet. It can be any animal.

n your program and mark it!

Here's a pet snake example:



You'll need

program.blockly

Testing

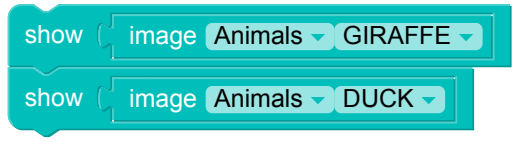
- [ ] Testing that the display is showing one of the seven pets.
- [ ] **Well done, you've made your own virtual pet!**

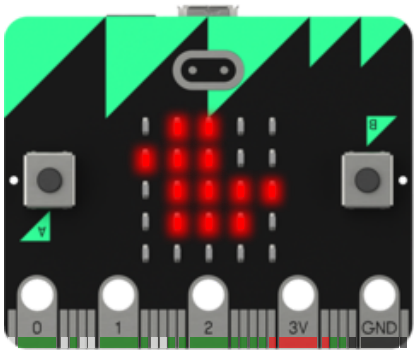
1.3. Animation

1.3.1. Ducks in a row

How do we show two images?

ck ▶ run.





Uh oh! **We only see a duck!**

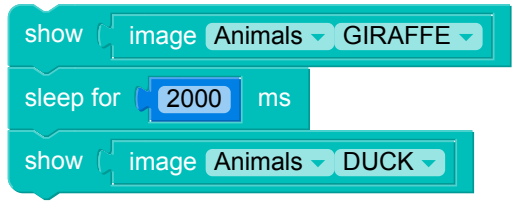
The micro:bit is *really* fast. It shows the giraffe too fast for us to see.

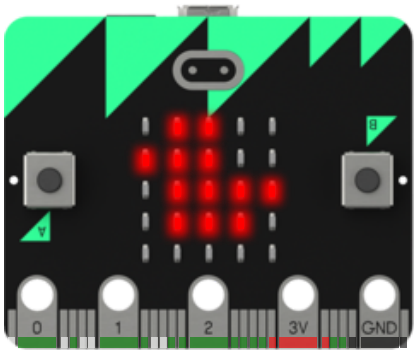
1.3.2. The sleep block

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

ck ▶ run.

e giraffe appears for 2 seconds. Then the duck appears.





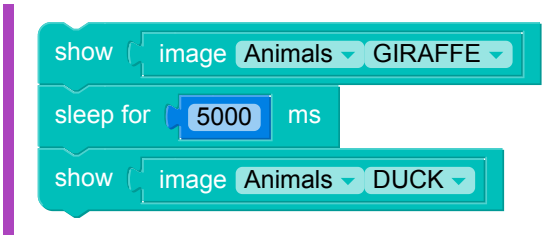
1.3.3. sleep 5000 ms

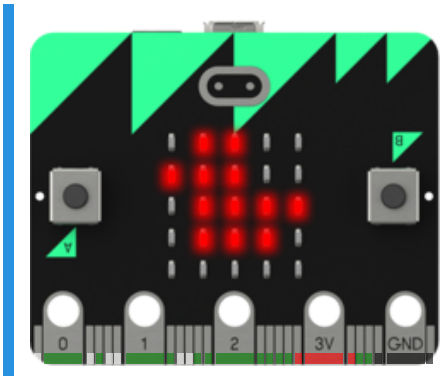
We can sleep for different lengths of time.

The example below shows the giraffe for **5 seconds**.

ange sleep 5000 ms to sleep 1000 ms .

ck ▶ run the example below. The giraffe appears for only **one second** now!





1.3.4. Problem: Pulling faces

Your micro:bit is a bit meh. Make it a bit silly! Write a program to show a MEH face for **one second**, then show a SILLY face.

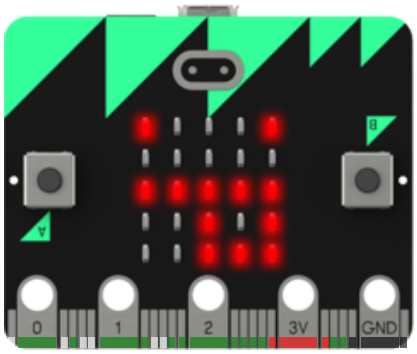
ag the blocks you need from the workspace.

ect the correct faces.

oose the correct sleep amount.

n the blocks together.

Here's an example:



Sleep for milliseconds

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

Testing

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

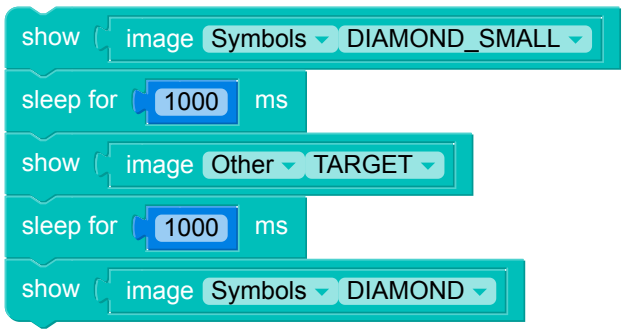
1.3.5. More images!

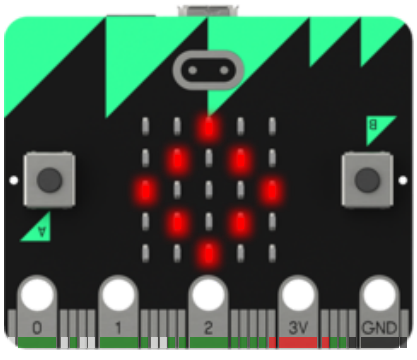
We can show as many images as we like!

ck ▶ run the example below.

You can see three different diamonds!
oose three new image blocks that tell a story.

play your story!





1.3.6. Problem: Virtual cocoon

Anything is possible with a virtual pet. Transform your virtual snake into a virtual butterfly!

- First show a `SNAKE` for **2 seconds**.
- Then show a `DIAMOND_SMALL` for **3 seconds**.
- Lastly show a `BUTTERFLY`.

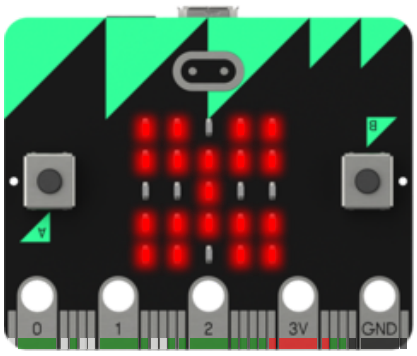
ag in the right number of show and image blocks into the workspace.

ag in the right number of sleep blocks.

ect the required images.

: the sleep time so the delay between the images is correct.

nnect all the blocks together.



Sleep for milliseconds

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

sleep 4000 ms will sleep for **4 seconds**

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

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

You'll need

program.py

Testing

- [ ] Testing that the display starts with a snake.
- [ ] Testing that the snake is still on the screen less than 2 seconds later.
- [ ] Testing that the display changes to a small diamond after 2 seconds.
- [ ] Testing that the small diamond stays on the display for 3 seconds.
- [ ] Testing that the display changes to a butterfly after 3 seconds.
- [ ] **Congratulations! You turned the snake into a butterfly!**

1.4. Letters and words

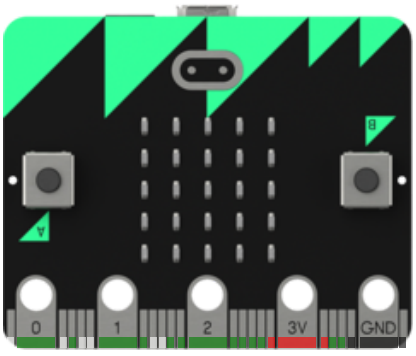
1.4.1. Scrolling letters and words

We can use scroll to show letters.

run the example.

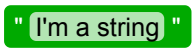
ange " Hello " to *your* name.

run the example again to scroll your name across the micro:bit!



A string of letters

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



1.4.2. Problem: I ❤ micro:bit

Show how much you like the micro:bit! Display `I ❤ micro:bit` on the LEDs in the following way:

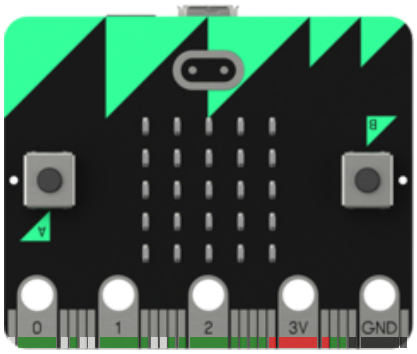
roll " I " on the micro:bit.

ow a **HEART** for **1 second**.

roll " micro:bit! ".

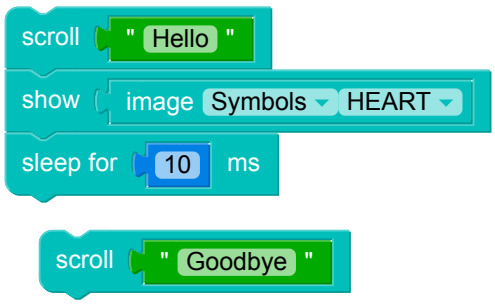
You already have the all the blocks. Remember Mark your program!

Your program should look like this:



You'll need

program.blockly



Testing

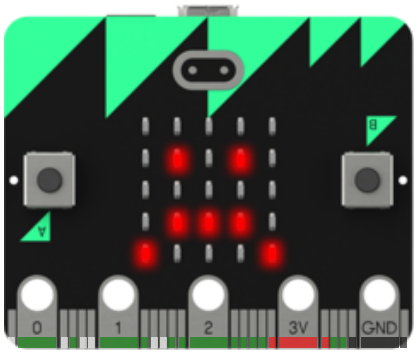
- [ ] Testing that an `I` scrolls past.
- [ ] Testing that a heart appears after the `I`.
- [ ] Testing that the heart stays on the display for 1 second.
- [ ] Testing that an `m` scrolls past.
- [ ] Testing that `micro:bit!` scrolls past.

1.4.3. Problem: My duck is sad

Let's give our pet personality!

1. Scroll " My " on the micro:bit.
2. Then show any image Animal for **1 second**.
3. Then scroll " is ".
4. Then show any image Faces.

For example, you could say "My 🦆 is ☹️":



Testing

- [ ] Testing that a `My` scrolls past.
- [ ] Testing that a pet appears after the `My`.
- [ ] Testing that the pet stays on the display for 1 second.
- [ ] Testing that `is` scrolls past.
- [ ] Testing that a face appears after the `is`.
- [ ] Testing that the face stays on the display.

1.5. Summary

1.5.1. Congratulations!

You finished Module 1!

We learned about:

- what is in the BBC micro:bit
- how to show an image on the micro:bit
- choosing your own image
- making the micro:bit wait with sleep
- scrolling " strings " on the micro:bit

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

2

BUTTONS AND GESTURES

2.1. Looping forever

2.1.1. Introducing loops

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

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

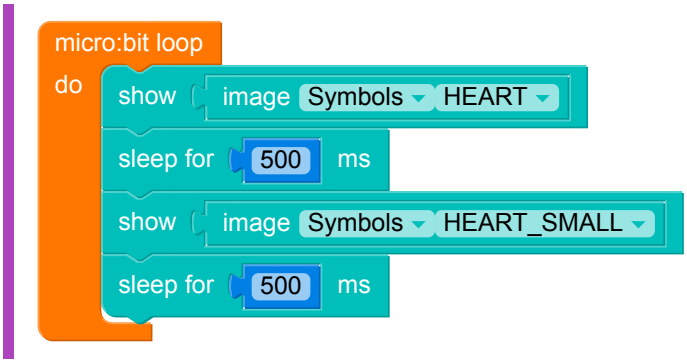
ck ▶ run below.

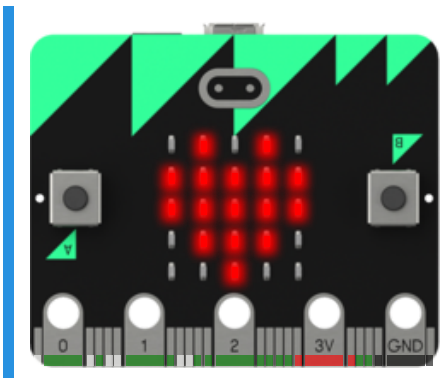
The heart can go forever!

ck the ■ button.

ange the speed of the heartbeat.

n the program again!





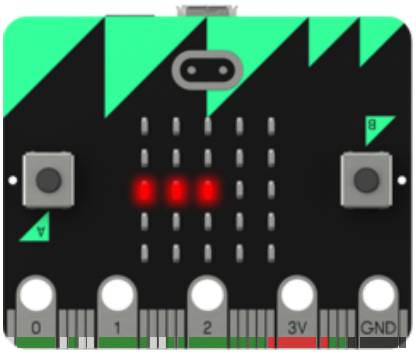
2.1.2. Problem: Tick tock

Time is ticking! Move a clock hand around the micro:bit forever.

Your program should:

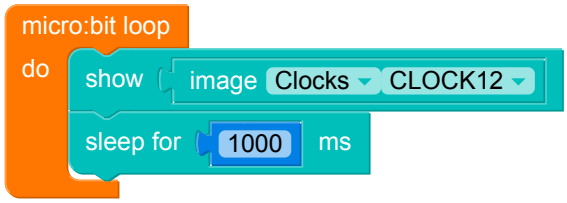
1. Show 12 O'clock for **1 second**.
2. Show 3 O'clock for **1 second**.
3. Show 6 O'clock for **1 second**.
4. Show 9 O'clock for **1 second**.
5. Loop this forever!

It will look like this, but will *loop* forever!



You'll need

program.blockly



Testing

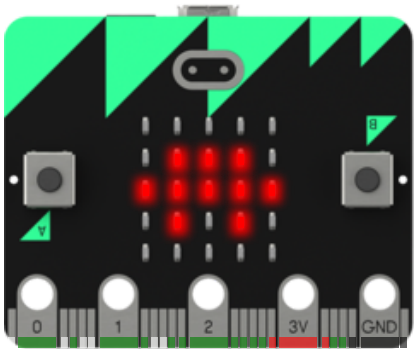
- [ ] Testing that the display starts with 12 o'clock.
- [ ] Testing that the display is still showing 12 o'clock after less than a second.
- [ ] Testing that the display shows 3 o'clock for a second.
- [ ] Testing that the display shows 6 o'clock for a second.
- [ ] Testing that the display shows 9 o'clock for a second.
- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display goes back to 12 o'clock for a second.
- [ ] Testing that the animation loops continuously.

2.1.3. Problem: Pet shop

Show me all of the virtual pets! Forever!

1. Show each `Animal` image for **1 second**. Do it in alphabetical order:
    1. `BUTTERFLY`
    2. `COW`
    3. `DUCK`
    4. `GIRAFFE`
    5. `RABBIT`
    6. `SNAKE`
    7. `TORTOISE`
2. And loop forever!

Like this example, but 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.
- [ ] Testing that the display shows a butterfly for a second.
- [ ] Testing that the display shows a snake for a second.
- [ ] Testing that the display shows a tortoise 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!**

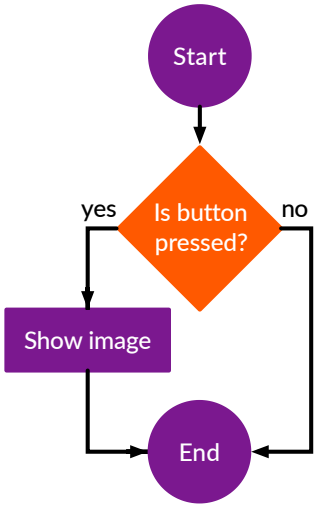
2.2. Making decisions with buttons

2.2.1. Making decisions

So far our programs only *show* things. The *same program* always shows the *same thing*.

But a program can have *user input*. This means it can do *different things* in *different situations*.

Like in this flowchart. An image is shown *only* if the button is pressed.

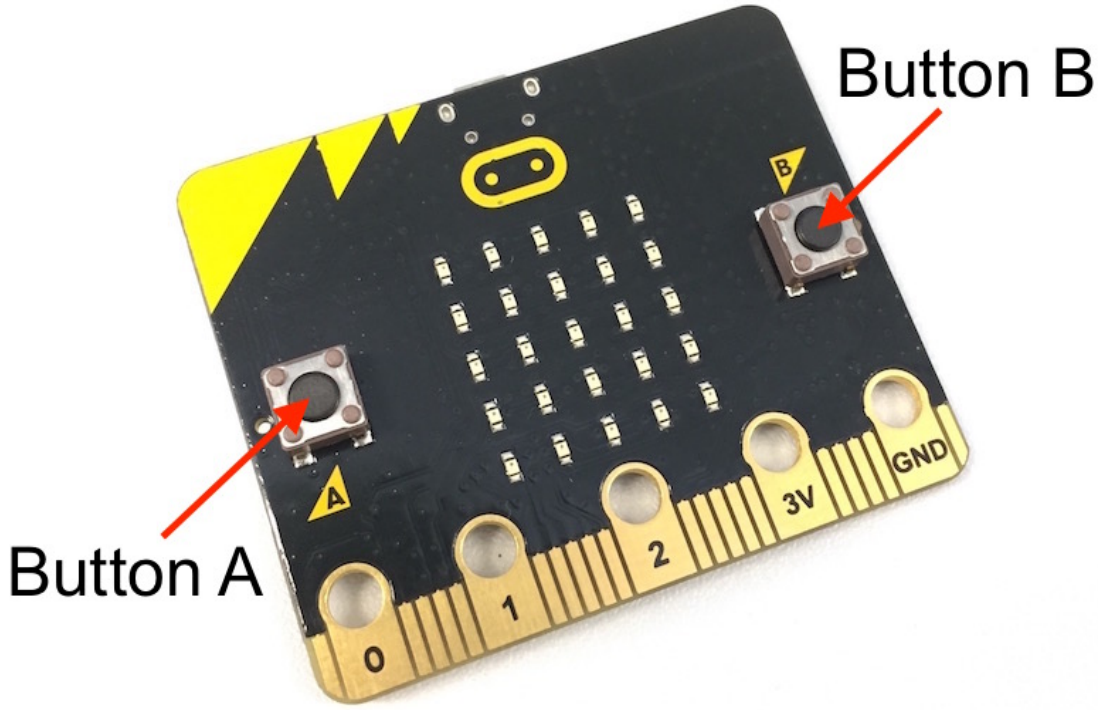


This is how we *make decisions* in a program.

2.2.2. Button A and Button B

The BBC micro:bit has two buttons.

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



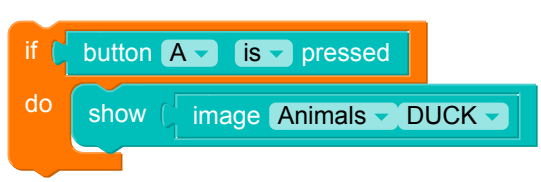
We can use button A is pressed and if to make decisions.

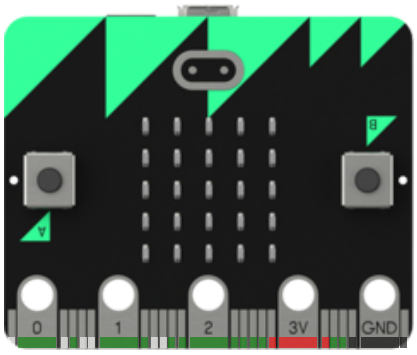
2.2.3. The if block

We can use the if block to make a decision.

run the example below.

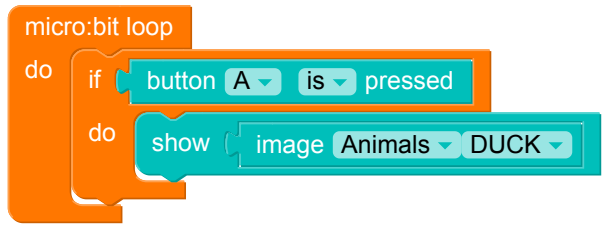
If you try and press the A button on the micro:bit *it doesn't work!*

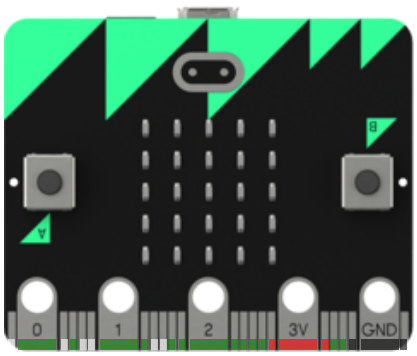




We can fix this by putting the if inside of the micro:bit loop.

run the second example below.



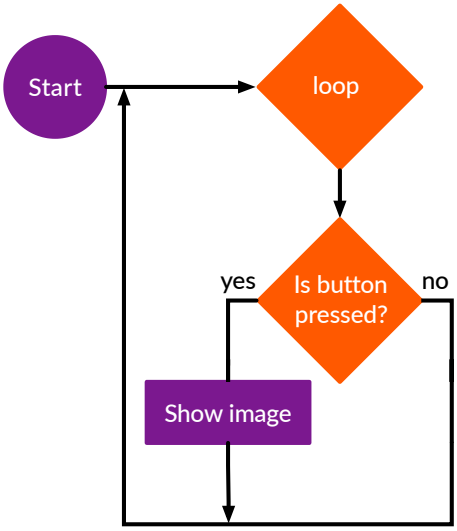


ess the ■ button.

> **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

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



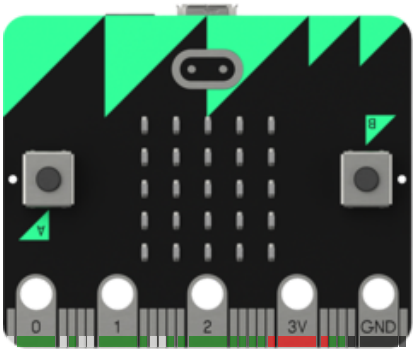
2.2.5. Problem: 3... 2... 1... GO!

On your marks, get set, GO! Make a quiet count down timer.

Write a program to count down if the **A** button is pressed.

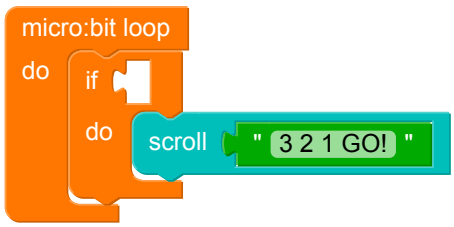
1. If the **A** button was pressed.
    - Then scroll `3 2 1 GO!`.
2. Loop forever!

Try it by running the code (▶) and pressing the **A** button in this example:



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.

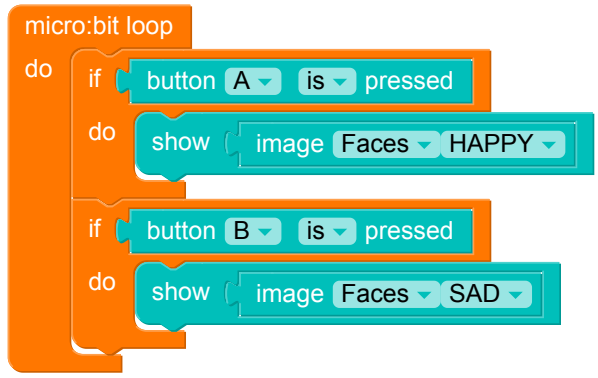
2.2.6. What if s?

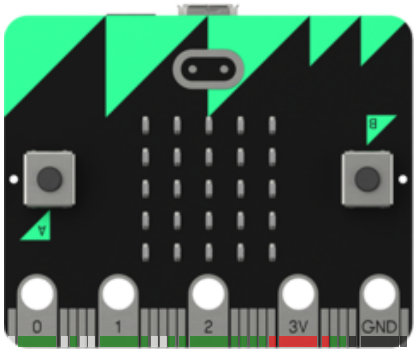
The micro:bit has more than one button.

We can use more than one if !

run the example below.

ess the ■ button.





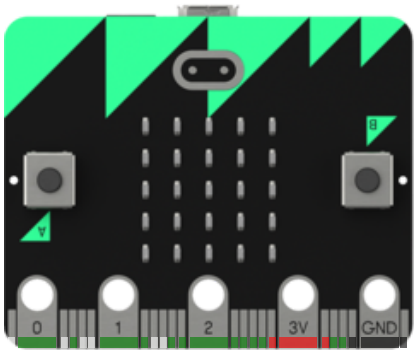
2.2.7. Problem: Pet duck vs pet rabbit

Is it a duck pet or a rabbit pet?

Write a program that will: show a duck when `A` is pressed, show a rabbit when `B` is pressed.

1. If the `A` button is pressed
    - Then show a `DUCK`
2. If the `B` button is pressed
    - Then show a `RABBIT`

Try running this example. Remember to press the `A` or `B` buttons!



Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off blank.
- [ ] Testing that it shows a duck when the A button is pressed.
- [ ] Testing that it shows a rabbit when the B button is pressed.
- [ ] Testing that it shows a duck then a rabbit when the A button then the B button is pressed.
- [ ] Testing that it continues to work multiple times.

2.3. Decisions with two options

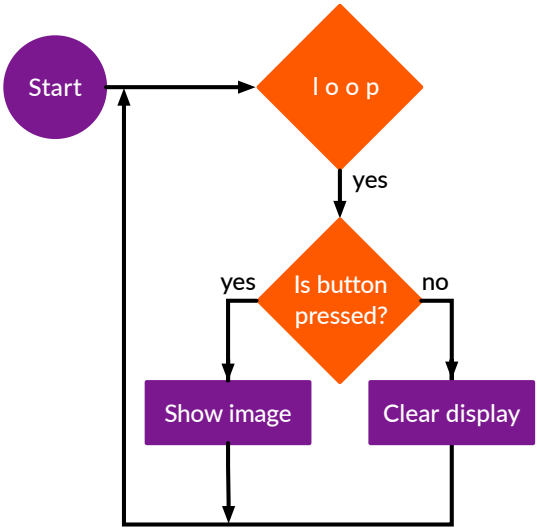
2.3.1. Decisions with two options

When we make a decision, we might care about both answers.

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

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

Like in this flowchart:

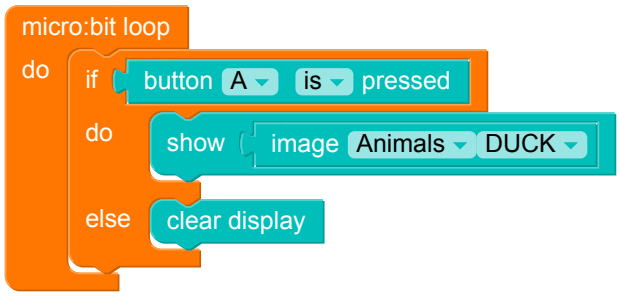


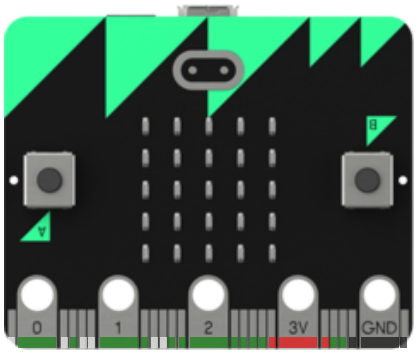
2.3.2. The if/else block

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

run the example below.

ess the ■ button.




B
A
0
1
2
3V
GND

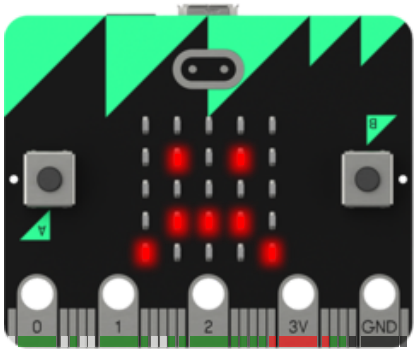
2.3.3. Problem: Smile for the camera!

Smile for the camera!

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

1. If the **A** button is pressed
    - Show a **HAPPY** face
2. Else
    - Show a **SAD** face

Run this example. You'll need to press **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.

2.3.4. Problem: Feed me!

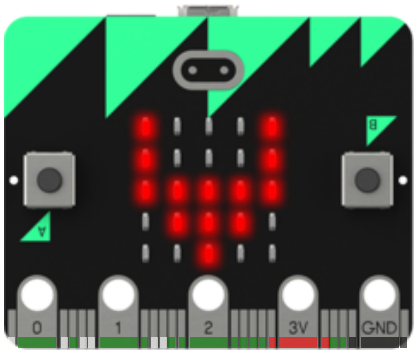
Even virtual pets get hungry!

Make a program to let your pet have some food!

1. If you "feed" the pet with the A button
    - Show an open mouth (using the **SURPRISED** face)
2. Else
    - Show an image Animal

You can use *any* animal you like!

For example the cow in this example is hungry. Feed him with **A**.



You'll need

program.py

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off showing a pet.
- [ ] Testing that it opens its mouth when the button is pressed.
- [ ] Testing that it went back to a pet after the button was released.
- [ ] Testing that holding down the button keeps the mouth open on the screen.
- [ ] Testing that it continues to work multiple times.
- [ ] **Nice work, you fed the pet!**

2.4. More complex decisions

2.4.1. Decisions with many options

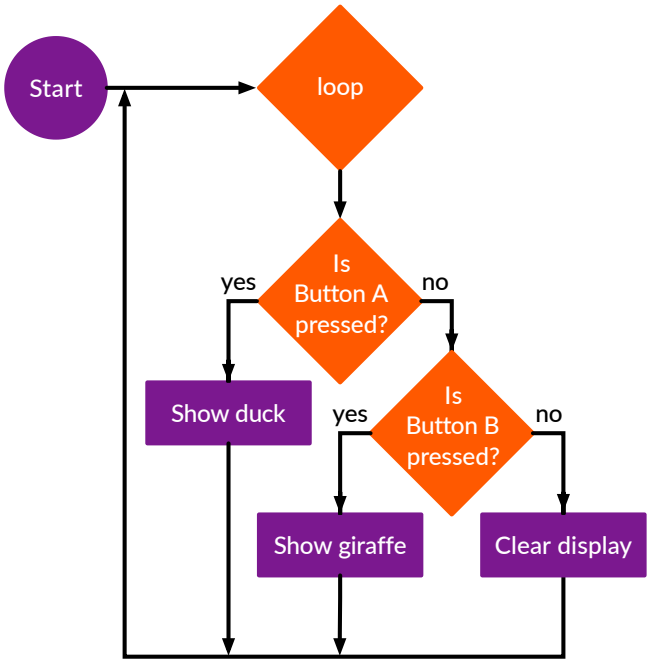
Sometimes decisions have many options.

For example:

1. Button A is pressed
2. Button B is pressed
3. Neither button is pressed

We need to ask *two* questions!

Like in this flowchart.



2.4.2. The if/else if/else block

We can add an `elif` (abbreviation of *else if*) clause to make the extra decision in the flowchart.

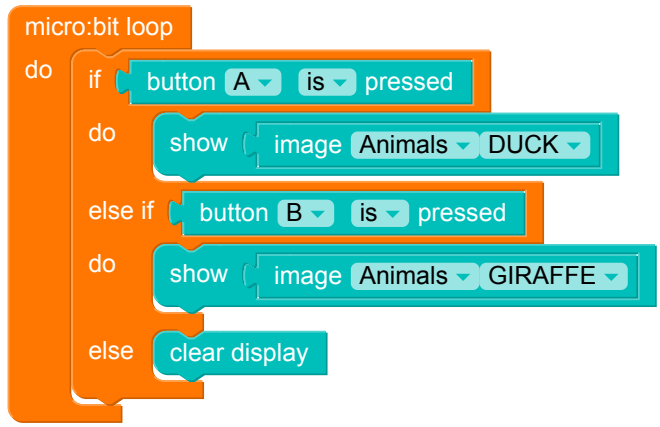
run the example below.

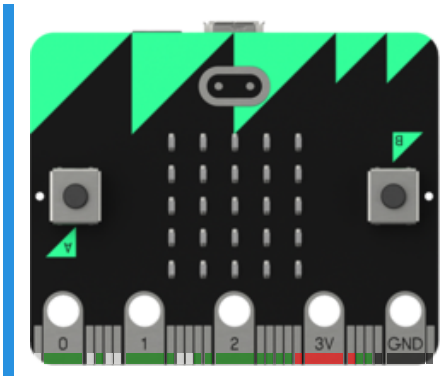
stop the example.

ap the button A condition with button B

Run the example again.

ange both images and ▶ run the program again.





2.4.3. What if both buttons are pressed at once?

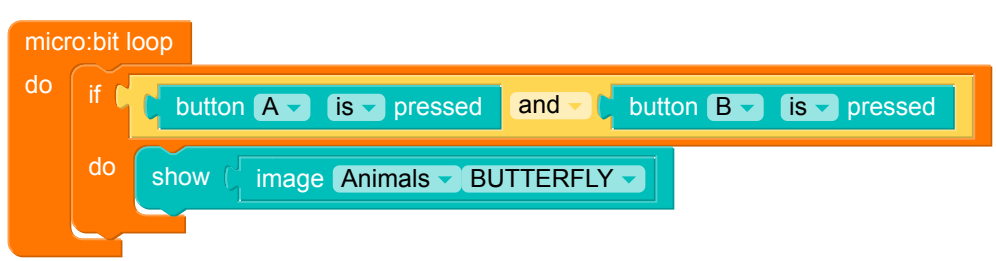
We can join decisions with the and block.

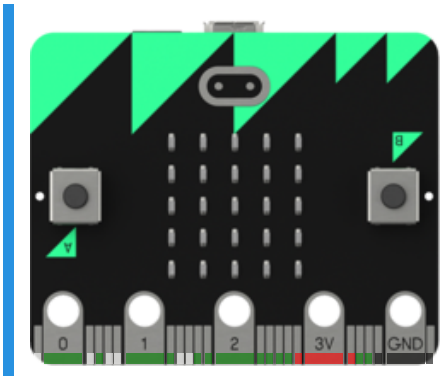
run the example below.

stop the micro:bit.

ange the image to a different image.

run the code again!





2.4.4. Even more options

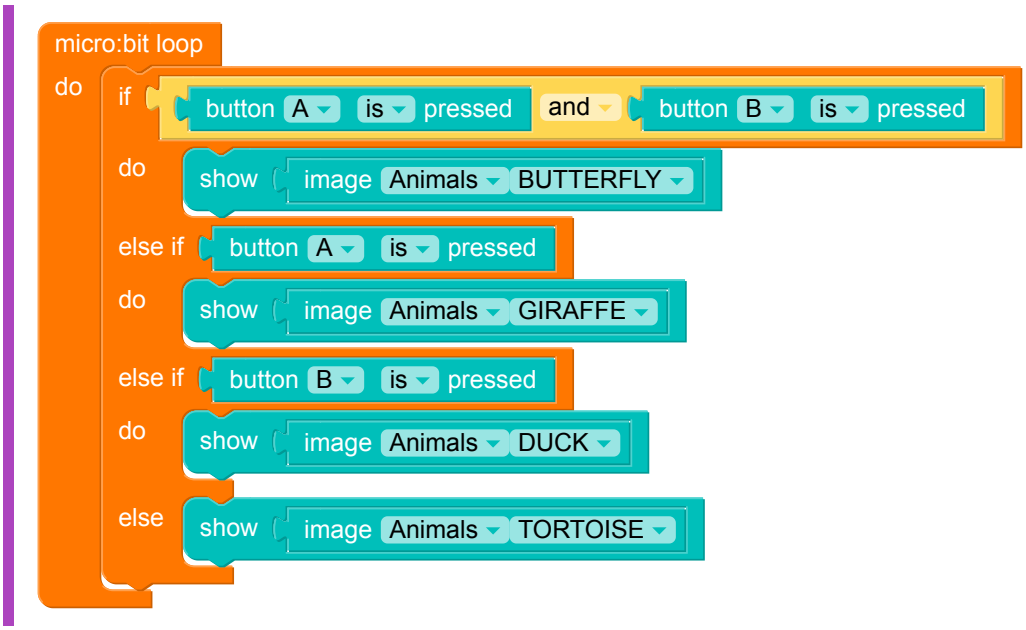
We can use blocks with many else if s to add more options!

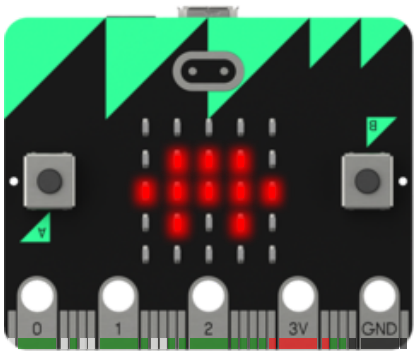
run the example below.

stop the code.

' swapping the order of the button is pressed blocks and the image blocks. What changes?

n it again!





The program above follows this table:

Buttons pressed	Output
A and B	🦋 Butterfly
only A	🦒 Giraffe
only B	🦆 Duck
none	🐢 Tortoise

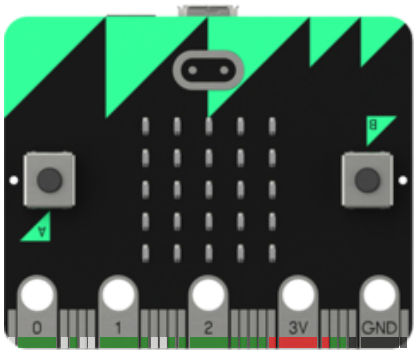
2.4.5. Problem: Petting zoo

Let's turn the micro:bit into a zoo where we can select different virtual pets!

Our program will:

1. Check if the **A and B** buttons are pressed.
    - Then show **DUCK**
2. If just **A** is pressed
    - Then show **RABBIT**
3. If just **B** is pressed
    - Then show **TORTOISE**
4. Otherwise...
    - Clear the display

Your program will act like this:

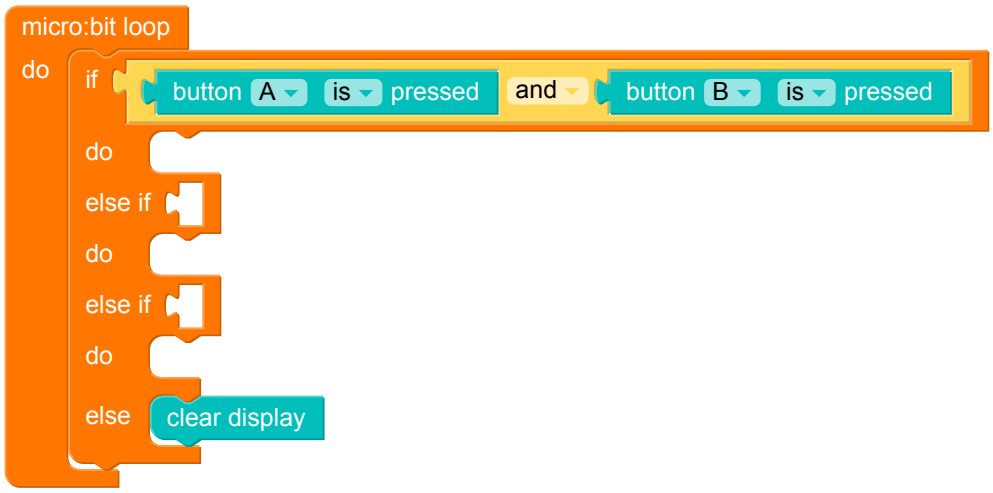


Remember!

Use the A and B keys on your keyboard to press both buttons at the same time.

You'll need

program.blockly



Testing

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

- [ ] 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 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 multiple buttons.
- [ ] **Congratulations! You made a petting zoo!** 🦕🦆🐇

2.4.6. Problem: Feed me or pet me!

Let's play with our virtual pet!

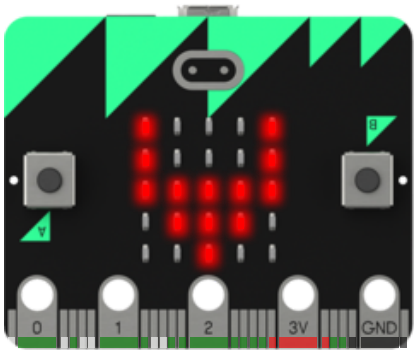
- The **A** button will feed the pet. This makes your pet open its mouth.
- The **B** button will play with the pet. This makes your pet happy.
- Feeding and playing at the same time make your pet angry.

This means your program will:

1. Check if the **A** *and* **B** buttons are pressed.
    - Then show an **ANGRY** face
2. If just **A** is pressed
    - Then show a **SURPRISED** face (for an open mouth)
3. If just **B** is pressed
    - Then show a **HAPPY** face
4. Otherwise...
    - Show the pet image

> **Animal friends**
>
> Any image Animal can be you pet!

Here's an example, if your pet was a cow:



You'll need

program.py

Testing

- [ ] Checking that your code contains an infinite loop.
- [ ] Testing that the display starts off showing a pet.
- [ ] Testing that the display shows an open mouth when the A button is pressed.
- [ ] Testing that the display goes back to the pet again when A button is released.
- [ ] Testing that the display shows a happy face when the B button is pressed.
- [ ] Testing that the display goes back to the pet again when B button is released.
- [ ] Testing that the display shows an angry face when both buttons are pressed.
- [ ] Testing that the display goes back to the pet again when the buttons are released.

- [ ] Testing multiple button presses.
- [ ] **Well done! You can pet, feed and make your pet angry! Like a real one!**

2.5. Summary

2.5.1. Congratulations!

Fantastic work! You made an interactive virtual pet!

We learned about:

- visualising programs as flowcharts
- a loop that repeats forever
- buttons on the micro:bit
- the difference between input and output
- simple decisions with if blocks
- decisions with two options with if/else blocks
- decisions with many options if/else if/else
- joining decisions with the and block

You can do all this and more with your micro:bit!

The next problem has lots of blocks you can try!

2.5.2. Problem: Blockly micro:bit Playground

This is a micro:bit playground question! Use the blocks to build anything you like!

Testing

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