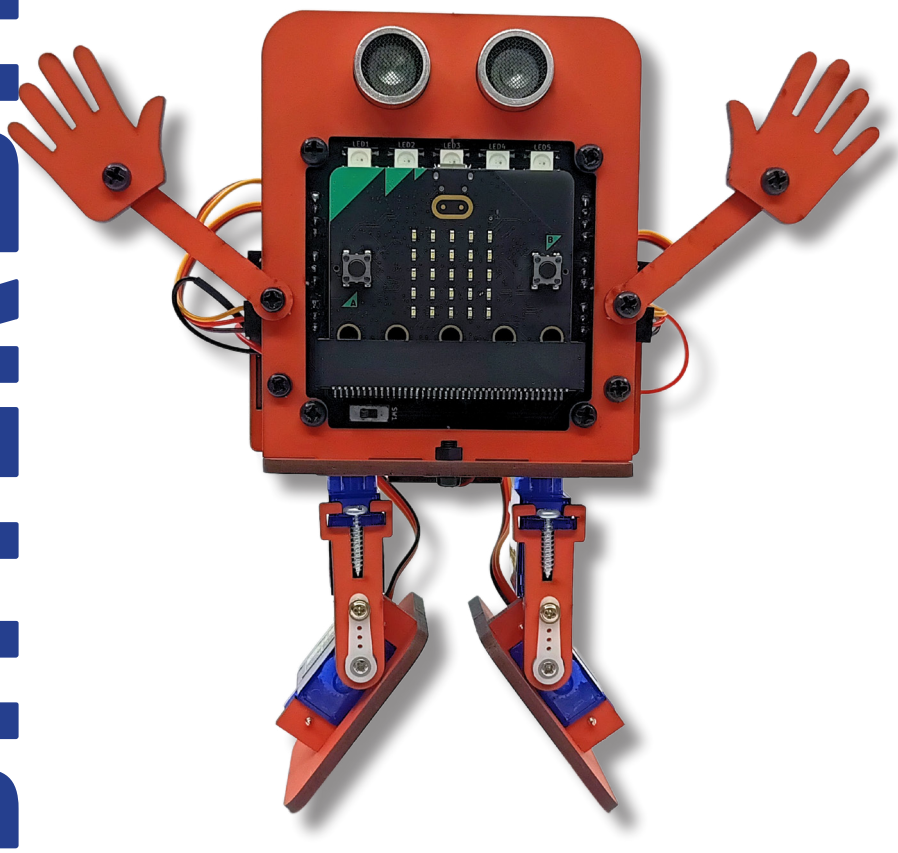
Coding and Robotics

JITTERBIT



Level 1
Student Manual

BOTSHOP
LET THE FUN BEGIN
EST 2014

Copyright:

What We Offer:
Innovative Robots: Dive into the XO-Skeleton series where modular robots come to life with the micro:bit, opening up a world of possibilities.

Comprehensive Training: From student guides to video tutorials, we provide everything you need to turn education into an epic journey.

Sustainability: Our robots are not just smart; they're made from recycled wood, teaching the value of sustainability along the way.

Join the STEM Revolution with us! Whether you're dancing with the JitterBit Dancing Robot, exploring with the Tank-o-Bot, or growing with GrowBotics, you're not just learning coding and engineering—you're becoming an innovator!

Get ready to transform your educational journey from "giving up" to "what do I build next?" with Bot Shop, where learning is not just education; it's an adventure!

Table of Contents

Introduction

This is your guide to learning how to make robots move, follow your commands, and do great things.

What Will You Learn?

In this guide, you'll learn how robots work and how to talk to them using simple codes. You'll also get to:

- Build robots and see how their parts fit together.
- Write instructions (called codes) to tell the robots what to do.
    - Solve fun problems by thinking like a detective.
- Work on exciting projects that let you use your imagination!

Why Learn Robotics and Coding?

Robots are everywhere! They help doctors, farmers, and even astronauts. Learning about robots and coding is like opening a door to endless possibilities. You'll:

- Learn great skills that you can use in the future.
- Discover how fun science, technology, and mathematics can be.
- Feel proud of the amazing things you can create!

Activities:

Almost all sections have an activity for you to participate in. This manual will guide you through each activity step by step.

Challenges:

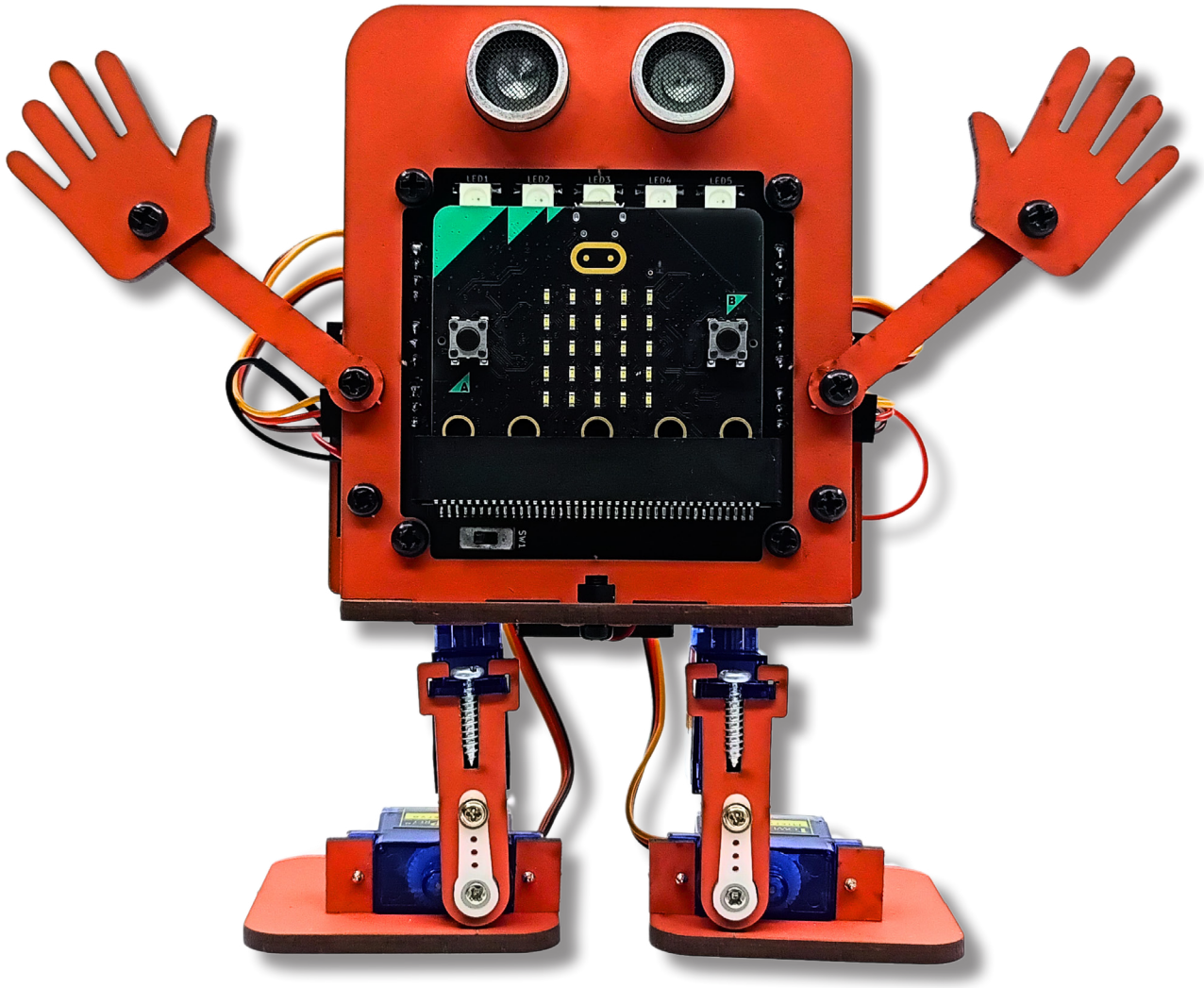
Many activities have challenges that let you develop your own ideas to make your activities do additional things.

Challenge yourself even more:

Do not be afraid that you can break something with coding.
Coding can not harm your robot or robot brain. Try different blocks with different values to learn even more.

Section 1:

Introduction to Coding and Robotics



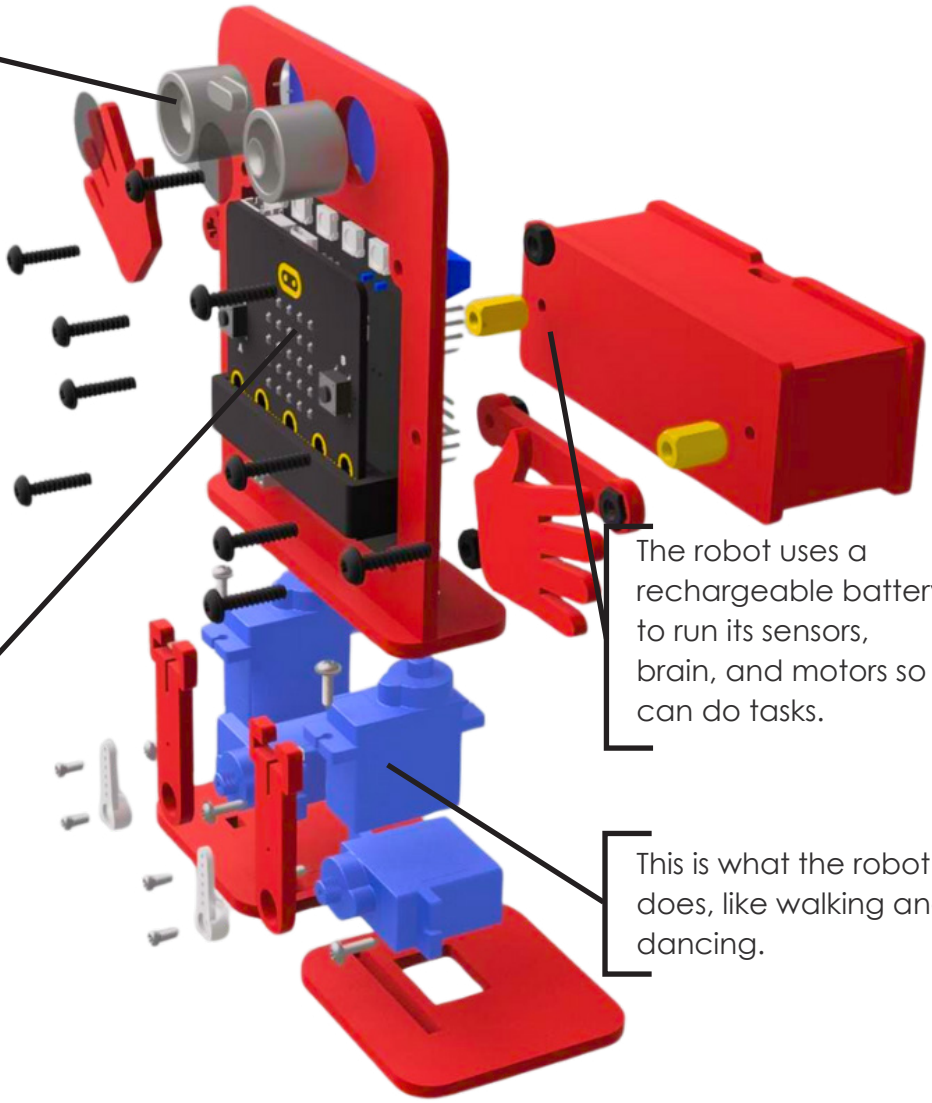
Section 1: Introduction to Coding and Robotics

Outcome

- Understand what robotics are.
- Understand the main parts of a robot.
- Understand what coding is.
- Understand what an algorithm is

What a robot consist of.

Sensors help the robot sense the world, like eyes for seeing or ears for hearing. They can also detect things humans can't, like magnetic fields or supersonic waves.



This is like the robot's brain! It has a processor and memory to help it think and learn about the world. It uses this "brain" to plan tasks, do actions, check if things are going well, and make changes to do better.

The robot uses a rechargeable battery to run its sensors, brain, and motors so it can do tasks.

This is what the robot does, like walking and dancing.

What Are Robotics and Coding?

Let's explore two super interesting things — robotics and coding!

What Is a Robot?

A robot is a special kind of machine. It can follow instructions to do different tasks, just like you follow a recipe to bake cookies or build something with Lego blocks.

Robots are made of different parts, such as:

- **A brain (computer):** This tells the robot what to do.
- **Sensors:** Sensors are like eyes and ears that help the robot see and hear. For example, a temperature sensor allows your robot to know the temperature of something.
- **Motors:** These help the robot move.
- **Wheels or legs:** These help the robot get around.

Robots can:

- Move around (like robot cars or animals).
- Pick things up (like a robot arm in a factory).
- Help people (like robots in hospitals or schools).

Switching things like lights, water sprinklers and motors on or off according to your instructions (like home automation)

Imagine building your own robot and making it move, talk, or even dance! You'll be learning how to do that.

What Is Coding?

Coding is how we talk to robots and computers. Instead of words, we use a special language called "code." Computers and robots understand this language well.

When you code, you write instructions for the robot to follow.

It's like giving directions to a friend:

- Turn left.
- Pick up the ball.
- Say hello.

Codes like pressing buttons or dragging blocks in a program can be simple. You'll learn to create longer codes to make robots do amazing things as you practice.

For example, you can code a robot to:

- Move forward, backwards, or spin around.
- Follow a line on the floor.
- Stop when it sees something in its way.
- Play music or flashing lights.

Coding is like solving a puzzle — it's fun and makes you feel like a genius when it works!

Why Are Robotics and Coding Important?

Robots and coding are part of our everyday lives.

They help us:

- Build cars and houses.
- Clean up messes and explore new places (even outer space!).
- Make life easier and more fun.

By learning robotics and coding, you'll understand how things work, think of new ideas, and even create something that could change the world.

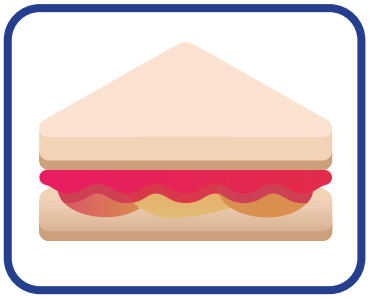
What Is an Algorithm?

An algorithm is a big word that means a list of steps to solve a problem or do something. It's like a recipe that tells you what to do, step by step, to get the job done!

Example: Making a Sandwich

Let's say you want to make a peanut butter and jam sandwich. Here's the algorithm (the steps) you would follow:

1. Take two slices of bread.
2. Spread butter on one slice
3. Spread butter on the other slice
4. Spread peanut butter on one slice.
5. Spread jam on the other slice.
6. Put the two slices together.
7. Eat your sandwich!



If you follow the steps in order, you'll end up with a yummy sandwich. That's an algorithm — it helps you finish something by doing each step one step at a time.

The order of steps is important:

When you create an algorithm, the steps should be done in a way that makes sense. Some steps need to be done before other steps.

For example, you should first put your socks on before you put your shoes on. This will not work if you have your shoes on and then want to put your socks on afterwards.

The order of steps in an algorithm is very important. When creating an algorithm, it's important to organize the steps logically. Some steps must be completed before others. For example, you should put on your socks before putting on your shoes; attempting to put on socks after wearing shoes won't work.

Algorithms for Robots:

Robots also use algorithms to determine how to perform tasks. For instance, if you want a robot to pick up a ball, its algorithm might look like this:

1. Move forward until you see the ball.
2. Stop when you reach the ball.
3. Open your claw (or hand).
4. Grab the ball.
5. Lift the ball.

If the robot follows these steps, it will successfully grab the ball! You can also see that the order of these steps is important; you can not lift the ball before you grab it.

Activity - Create Your Own Algorithm

Activity time - 20min

Let's Pretend!
Imagine your robot is helping you put away toys in a box.
What steps would you tell it to follow?

Write your algorithm like this:

1. Move forward to the toy box.
2. Stop when you see a toy.
3. Pick up the toy.
4. Put the toy in the box.
5. Repeat until all the toys are gone.

See how simple it can be? Algorithms are all about thinking step by step to get things done.

You can not brake anything with coding, so you can try any of the blocks you like!

Challenge

Create an algorithm on a piece of paper for a robot that will move forward until it sees something in its way. Then, it must stop, wait for 5 seconds, turn around, and move forward for 3 seconds.

Draw a robot that you would like to build.

Write down 3 things a robot can do to make your life easier or make things happen faster.

Section 1: Questions

1: What is a robot?

2: What is coding?

3: Why are robotics and coding important?

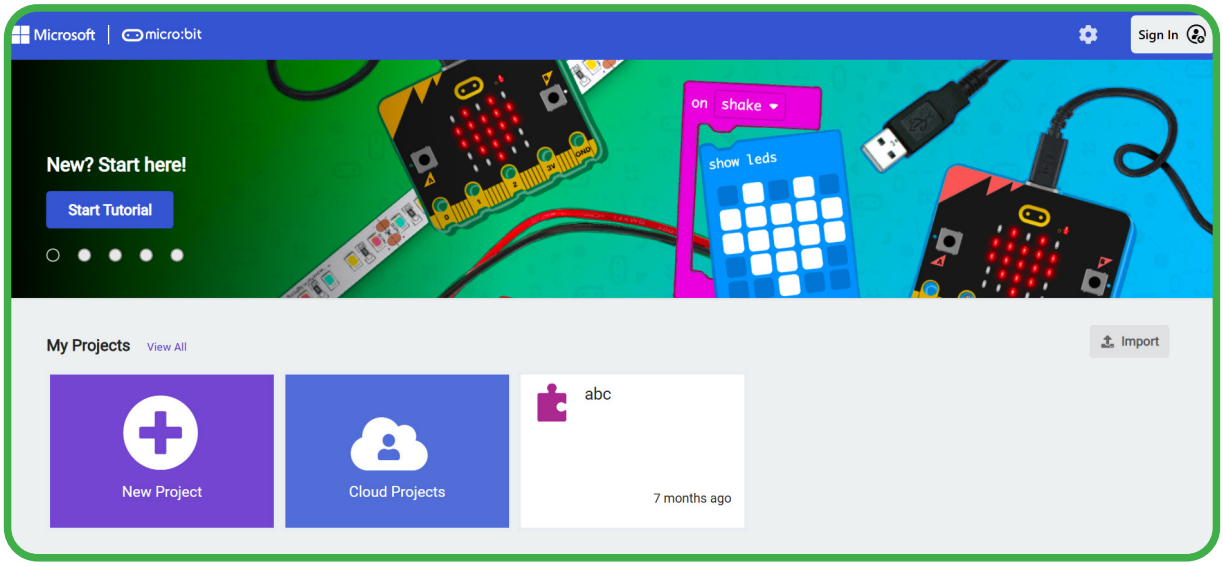
4: What is an algorithm?

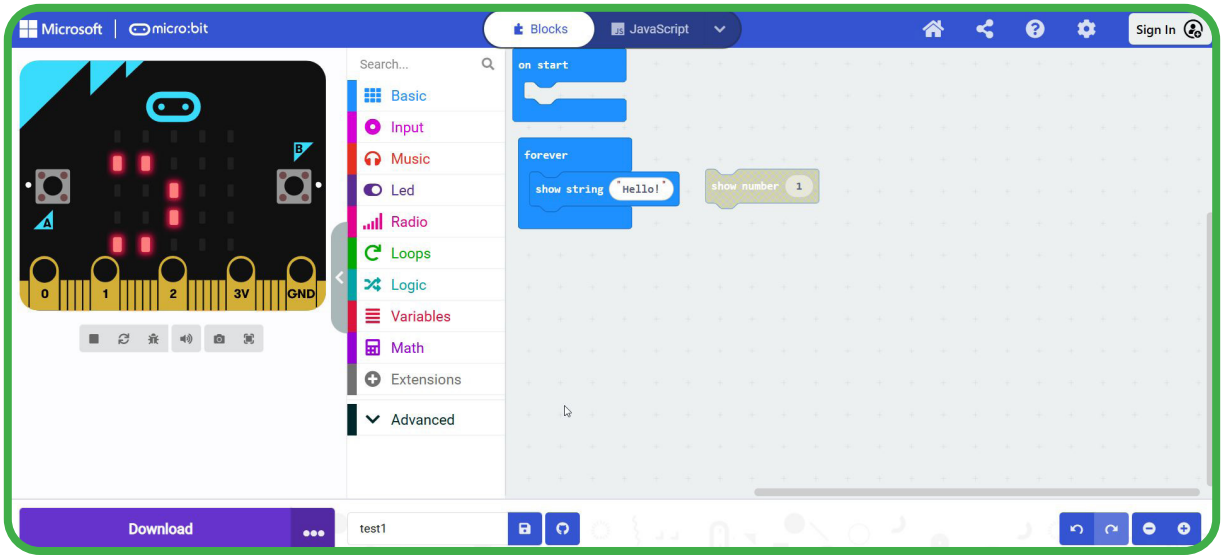
5: Why is the order of steps important in an algorithm?

6: Give an example of an algorithm for a robot:

Section 2:

Block Based Programming using the micro:bit (Robot Brain)





Section 2: Block Based Programming using the micro:bit (Robot Brain)

Outcomes

- Understand what block based programming is.
- Understanding block based software.
- Understanding the 3 main parts of a block based program.
- Downloading code to the robot brain.

You now know that algorithms are the steps a robot must follow to do things. These steps can be created in many ways, and we call the creation of algorithms, coding.

Coding is done in many ways. We use software to write our code and send it to a robot brain. The software takes the code we write and changes it into a language that robots can understand.

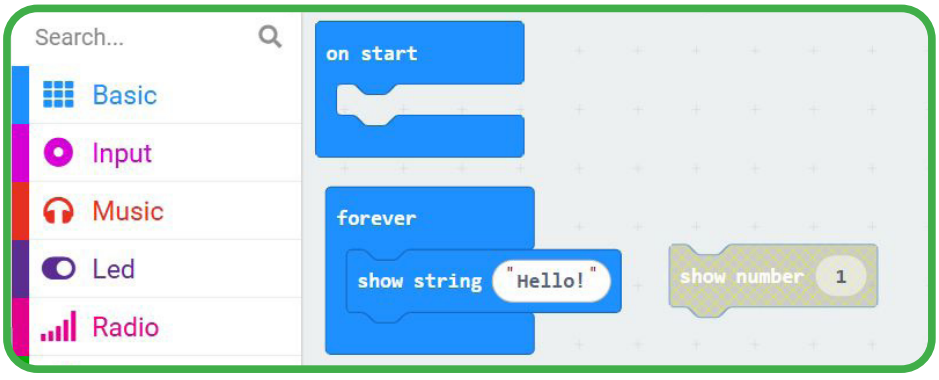
Block-based programming:

One way of writing these steps, so that the robot will understand, is using block-based programming.

Each block is a step the robot can take to do something like displaying a name on a screen or to move forward, if your robot has a motor and wheels.

These blocks work like puzzle pieces. Blocks that do not belong together or work together can not be put together, just like puzzle pieces.

In the software, these blocks will snap together if they can work together.

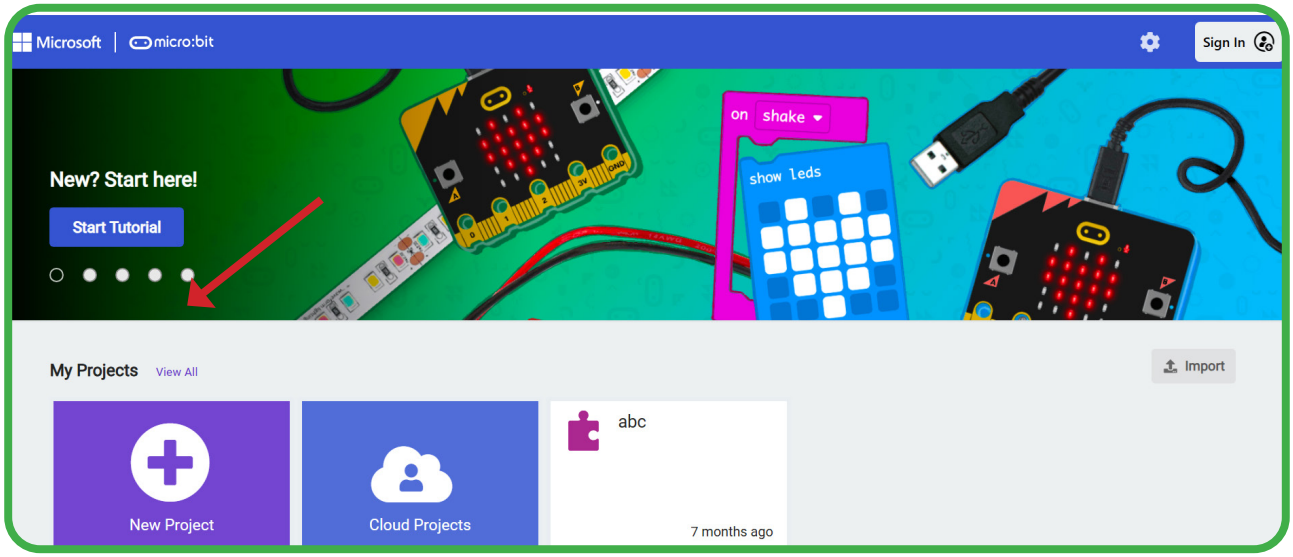


Block based environment (Software):

There is a lot of software available for block-based programming, like Scratch and *MakeCode*. With the micro:bit robot brain, we will be using *MakeCode*.

You can find the software here:
https://makecode.microbit.org/

You will then be taken to the coding software. See the screen below:



Create a new project

Projects: Each time you code something new, it is called a project. So, when you start using *MakeCode*, the first thing to do is create a new project.

Name your project so that you know what it is about. As you learn, you will create many projects, and to remember which project is doing what, it will be very helpful if you name it something that makes sense to you.

1. Once in the software, click the "New project" button.
2. Give your project a name. An example name can be "The first project I ever coded."

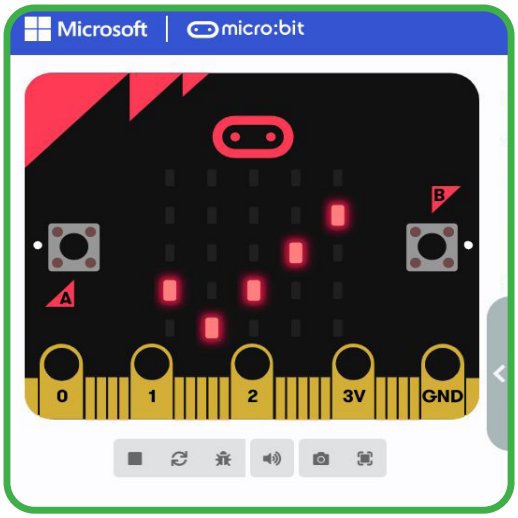
Understanding the MakeCode coding platform:

There are different areas on the platform, and we will look at each separately.

The simulator area:

In coding, a simulator is like making a pretend version of something real on a computer. For example, if you want to learn how to drive, you can get a simulator that is like a game that will teach you how to drive.

In coding, a simulator shows you what your code will do on a computer without having to upload the code to a real microcontroller.



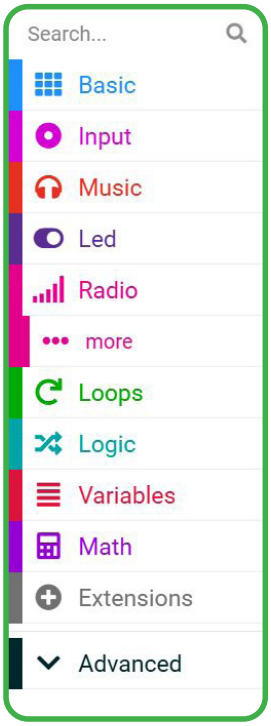
The first area we will look at is the simulator area to the left of your screen.

You will see an image of your micro:bit.

As you add blocks in the work area, you can see what happens on the simulated micro:bit.

At the bottom of the micro:bit simulator are six buttons to:

- Stop the simulation
- Restart the simulation
- Debug mode that shows you if there are errors in your code
- Turn off sound in the simulation
- Take a screenshot
- Make the simulation full screen (make it bigger so it is the size of your whole screen).



The toolbox area:

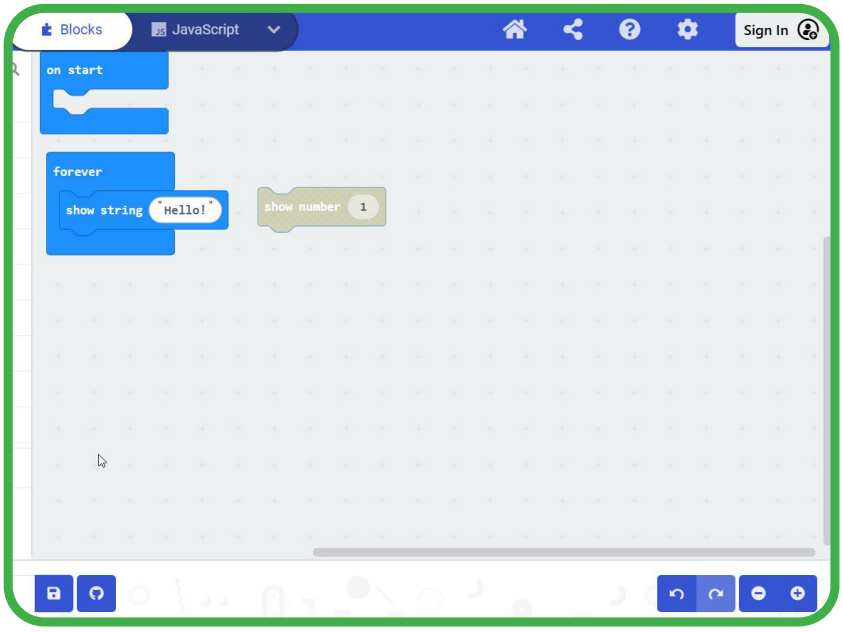
In the toolbox area, you will find all the different coding block folders. When you click on a folder like Basic, you will be shown a window with all the blocks available in that folder.

The toolbox is in the middle of the software screen, and at the top is a search area for quickly finding blocks.

The workspace:

This is the biggest space in the block-coding software. You will drag all your blocks from the toolbox here. This is the space where you will build your algorithms.

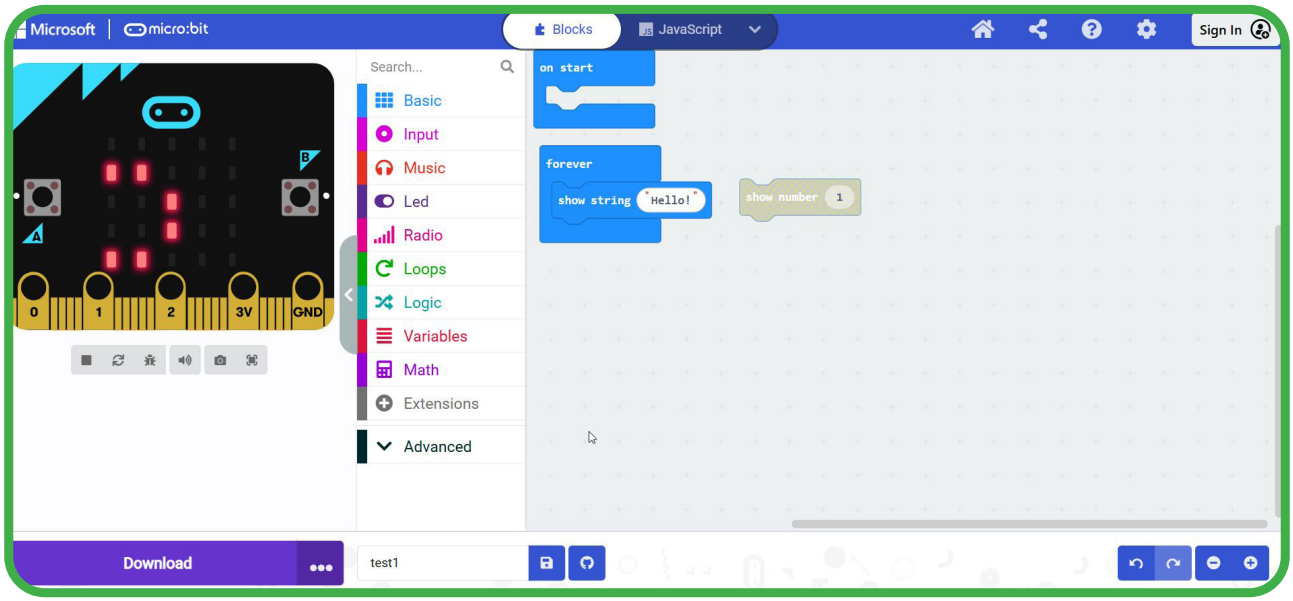
Below is an image you have seen before showing all three areas.



Activity: Downloading your code to the robot brain

Activity time - 20min

When you code your algorithm, you can upload it to your micro:bit robot brain.



1. Always first see how your code looks on the simulator, and once you are happy with the result, upload it to the micro:bit.
2. Use the images above as an example to show "hello" on the micro:bit LED display.
3. The "show string" block is in the basic toolbox folder. Drag the block inside the forever block, then change the wording to "hello".
4. Connect the micro:bit to your computer or tablet with the provided USB cable.
5. Lastly click on the big blue download button at the bottom left-hand side of the software, in the simulation area.
6. Select the micro:bit in the popup window, pair it with your software. Once it is paired and connected - press download.

Challenge

Find all blocks that have the word "show" in them in a fast way.

Section 2 - Questions

1: What is block-based programming?

2: What software will we use for block-based programming with the micro:bit?

3: What is a project in the MakeCode software?

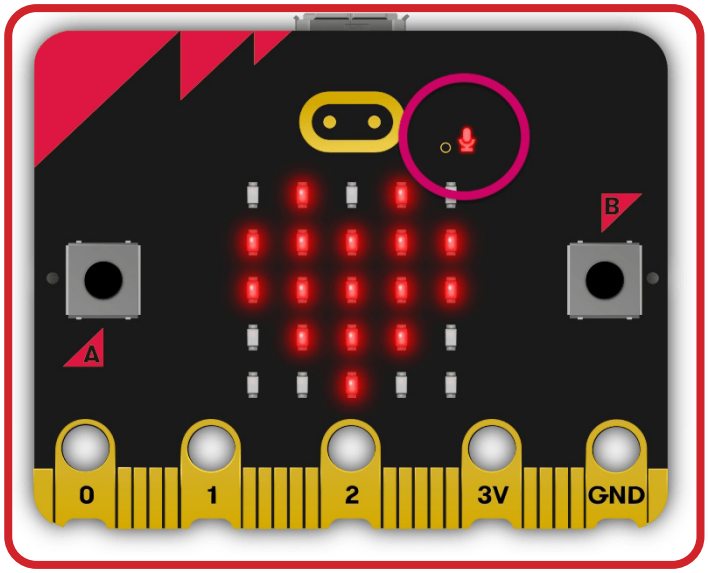
4: What is the purpose of the simulator area in the MakeCode platform?

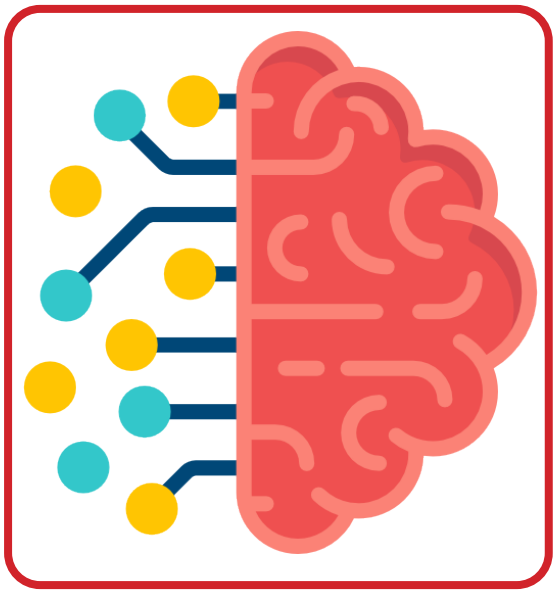
5: Where do you find the different coding blocks in MakeCode?

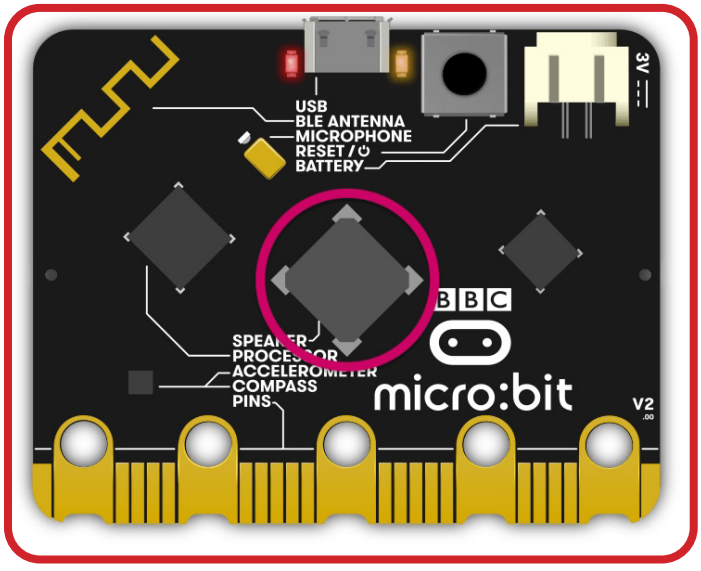
6: How do you download your code to the micro:bit?

Section 3.

The micro:bit microcontroller (The robot brain)



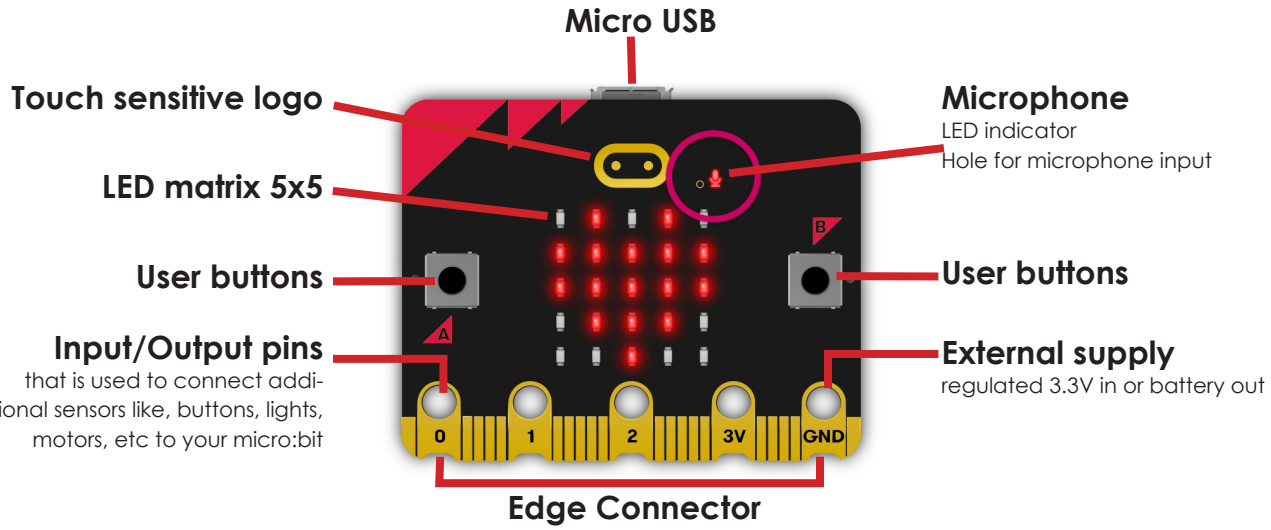


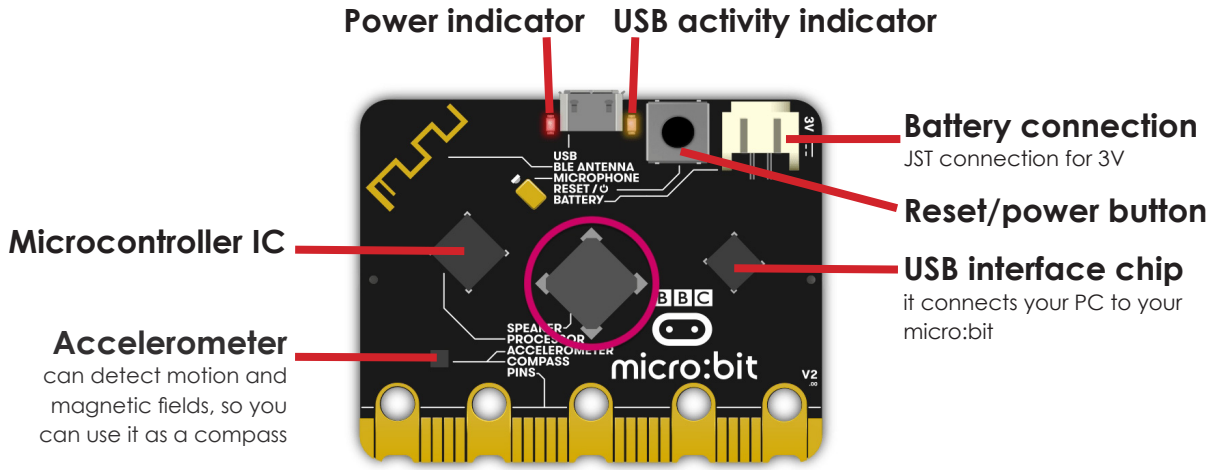


Section 3: The Micro:bit microcontroller (The robot brain)

Outcomes

- What is a microcontroller
- What are sensors
- What are inputs and outputs





What can a microcontroller do?

- Store the code you wrote. It will store it even if there is no power.
- Can do calculations and computations (do the steps you said it must do).
- It can send power to things connected to it, like, for example, making a motor turn
- Can read data from things like temperature from a temperature sensor
- Can send data to things like music to a speaker.
- It can store data from sensors and other things in memory and be used repeatedly.

A microcontroller consists of many different parts. The micro:bit comes with many other things, like sensors, that most other microcontrollers don't have.

Sensors:

A sensor is an electronic component that can measure weight, temperature, distance, and movement. It can "sense" things outside the microcontroller.

- **Accelerometer:** detects motion, tilt, and orientation.
- **Magnetometer:** measures magnetic fields and can act as a compass.
- **Temperature Sensor:** tells how warm or cold the micro:bit's brain (CPU - Central Processing Unit) is, which is close to the temperature of the room
- **Light Sensor:** detects light levels using the LED matrix.

Input/Output:

All microcontrollers come with pins to connect to things like lights, motors, etc. A microcontroller listens to the world and talks back!

Here's how:

- **Input:** This is how the microcontroller "listens." It can get information from buttons, sensors, like a thermometer, or light and sound.

  For example, when you press a button, the microcontroller knows you did it.

- **Output:** This is how the microcontroller "talks." It can turn on lights, play sounds, or move things like a motor. **For example**, it can light up an LED to say "Hello!"

So, input is what it hears, and output is how it answers!

The micro:bit microcontroller comes with the following Inputs and Outputs:

- **2 Buttons (A and B):** For user interaction and input.
- **LED Matrix:** 5x5 grid of LEDs for displaying text, images, and animations.
- **Edge Connector:** 25 pins for connecting to external devices and accessories.
- **Microphone:** Detects sound levels.
- **Speaker:** Plays sounds and music.

Input and output pins:

All microcontroller boards also have pins for connecting sensors, keypads, screens, speakers, and hundreds of other things to them.

Wireless Communication:

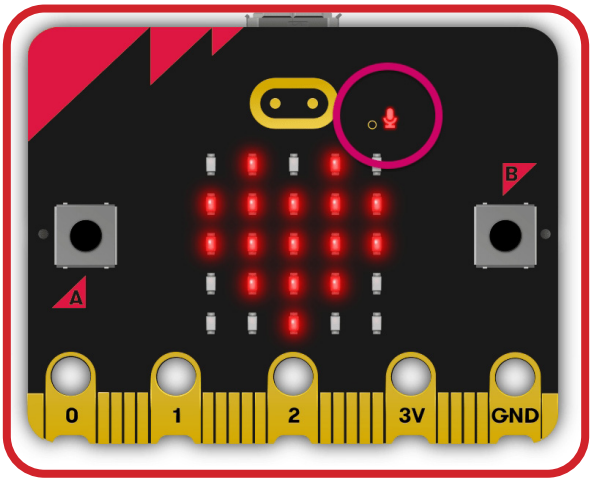
- **Bluetooth Low Energy (BLE):** Enables wireless communication with devices like phones, tablets, and other micro:bits.
- **Radio Module:** Allows micro:bits to communicate over short distances using a simple radio protocol.

Power and Connectivity:

- **USB Port:** This is for programming and power supply.
- **Battery Connector:** For powering the micro:bit with AAA batteries.
- **Reset Button:** Restarts the micro:bit.

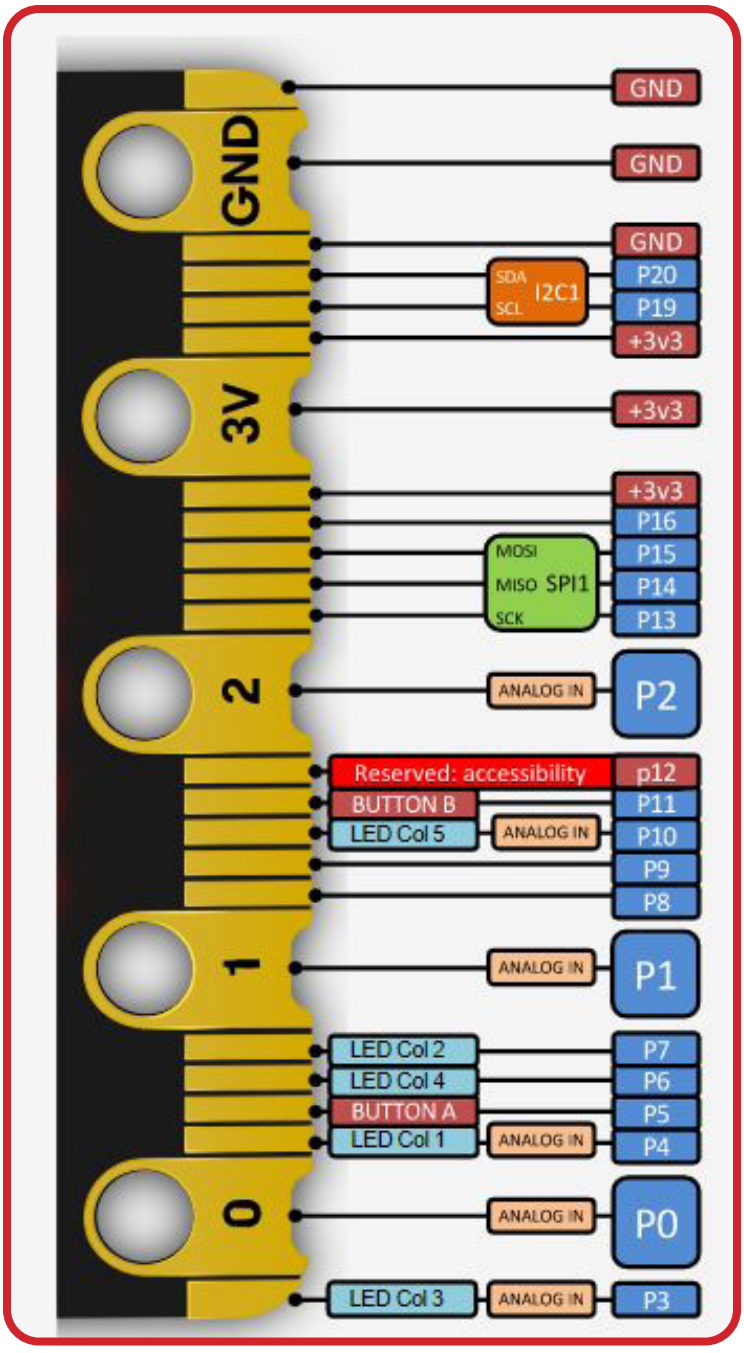
Other Feature:

- **Logo Touch Sensor:** The logo is touch-sensitive on the micro:bit v2.



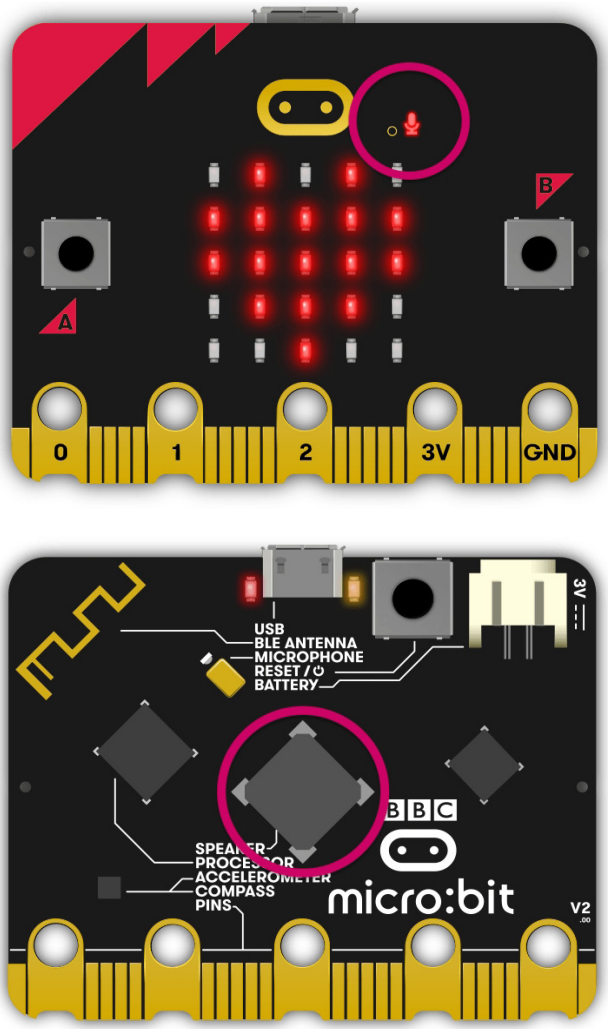
The micro:bit has many input and output pins, and although many are used on the board for LEDs, speakers, etc., there are still plenty available for additional input and output electronics like moisture sensors, LCD screens, and much more.

Below is an image from the micro:bit website of the pins you can connect sensors, screens, etc, to the micro:bit.



Activity: Find all the different components on the microcontroller board.

Use the image in the beginning of this section and find the following components on your physical micro:bit:



1. Button A and button B
2. USB port for coding
3. The microphone
4. The LED matrix
5. Touch sensitive logo
6. Inputs and outputs
7. Reset/Power button

Section 3 - Questions

1: What is a microcontroller?

2: What is a sensor?

3: What is the difference between input and output for a microcontroller?

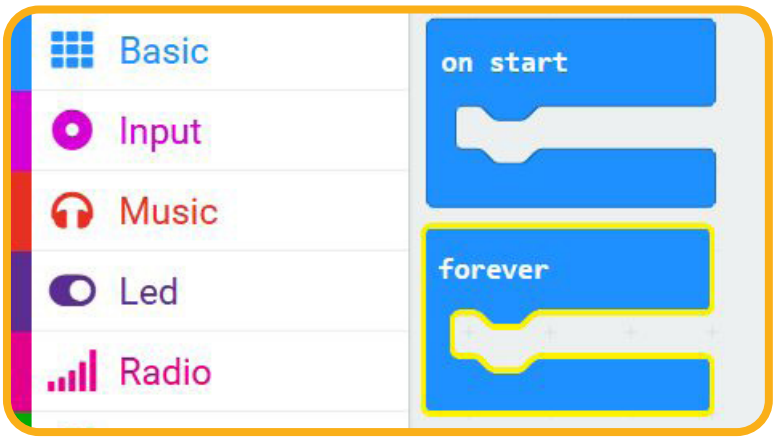
4: Name three inputs that the micro:bit has.

5: Name three outputs that the micro:bit has.

6: What are two ways to power the micro:bit?

Section 4:

Use Variables and the Forever Block



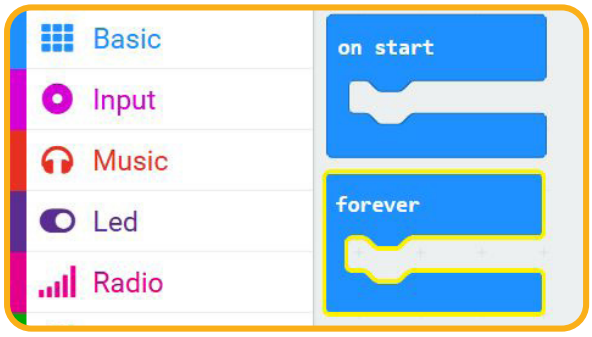
Section 4. Use variables and the forever block.

Outcomes

- Understand the on start and forever block
- What are variables
- The two most used variable types
- Naming, changing and deleting a variable
- Creating your first variable

The most important blocks: “on start” and “forever” blocks.

The On Start and Forever Blocks:



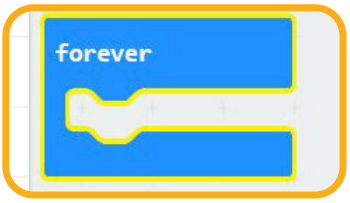
The “on start” and “forever blocks” are very important, and almost all other blocks are placed inside these two blocks.

The blocks you place inside the “on start” block will only be done once when you power up your microcontroller.

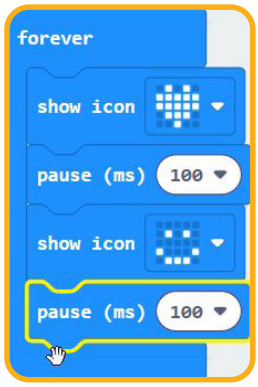
On start :

This block instructs your microcontroller on what to do when the microcontroller starts up. An example can be to make sure all the motors on your robot are switched off so that your robot will not start moving when you switch your robot on. Another example is letting your robot make 3 beeps when switched on.

Forever:



The blocks you place inside the "forever" block tell the microcontroller to repeat the instructions forever until the microcontroller is switched off. Let's say you have 4 blocks like below. The microcontroller will start doing block 1, then 2, then 3 and last block 4. Once it completes the last block, the microcontroller will start all over again and do block 1, and so on. It will only stop once you switch off the microcontroller.



Block 1 will show a heart on the microcontroller LED screen, block 2 waits for 100 milliseconds, block 3 then shows a smiley face, waits for 100 milliseconds. Then it starts over again showing the heart etc.

Variables:

What Are Variables?

Think of variables as data storage spaces in your robot's mind or your computer where you can keep different information or special items. You can then, at any time, see that data or even change that data.

You will give the variable a name like room temperature. You will then read the temperature from a temperature sensor. The last thing to do is to tell the microcontroller to save that value in the variable named "roomTemperature"

From now on, you can instruct the microcontroller to read the value in "roomTemperature" and, if the temperature value is high, start a fan, for example.

Why Are Variables So Useful:

They help keep track of things (like game scores or the number of times your robot has done something).
They simplify your code because you don't have to write long instructions every time. You can play and experiment by changing what's in the variables to see how it affects your robot or game.

1. Storing Information:

Imagine you have a secret note that says, "The magic word is 'Open Sesame'." In coding or robotics, a variable is like that note. You can write something in it, and you look at that note whenever you need to remember what it says!

Example: If you're making a game where your robot has to collect coins, you might have a variable called *coinCount*. Every time the robot picks up a coin, you add 1 to the variable *coinCount*. In the beginning *coinCount* will be 0. When your robot collects the first coin, the microcontroller will add 1 to *coinCount* so that *coinCoint* is now 1.

A variable in this example will keep count of how many coins it collected.

2. Naming Variables:

You can name these variables whatever you want. But they need to be names that make sense to you so you remember what they're for. For example, you could call a box with your favorite snack "snackBox."

Example: If your robot needs to know how many steps it should take before turning, you might name your variable *stepsBeforeTurn*.

Let's start learning more about good programming practice.

In programming, the rule for variable names is that they must not contain spaces or special characters like "-+*()." A variable is all in lowercase, like "room." If it makes more sense to use two words, the second word will start with a capital letter (still no spaces), like this: *roomTemperature*.

In many coding programs, a variable name is case-sensitive. In coding, the computer treats uppercase letters (A, B, C) and lowercase letters (a, b, c) as entirely different characters. So, if you're telling the computer to use a word, it has to be exactly right, with the same big and small letters, or it won't recognize it. So, a cat is different from a Cat for microcontrollers.

3 Changing Variables:

The great thing about variables is that they can change! If you start with 0 coins in *coinCount*, and then find 3 coins, you can change *coinCount* to 3.

Example: If your robot has a variable for its speed, let's call it *robotSpeed*; you might start with *robotSpeed* at 50. But if you want your robot to go faster, you change robot speed to a higher number.

4. Using Variables:

Once you've put information in a variable, you can use it repeatedly. It's like telling your robot, "Remember this number," and then later saying, "Move this many steps," and it knows exactly how many steps to take because it remembered that number!

Example: If you have a variable *colorOfLight* set to "Red", your robot can use this to light up red lights or change its behavior when it sees red.

5. Types of Variable:

There are many types of variables.
Text variable contains text like "Hello", there is also number variables that will store numbers especially for use in mathematics.

Number variables will not work if you try to put text in.

Text cannot be used in mathematical operations, and numbers cannot be stored in text variables.

How Variables Work in Block-Based Coding:

Blocks: In programs like Scratch or *MakeCode*, you use blocks to create your code. There's usually a special block for making, changing, and using variables.

Set Blocks: You use a "set" block to put information into a variable. Like "Set *coinCount* to 0".

Change Blocks: To automatically add or subtract numbers from your variable, there's a "change" block. Like "Change *coinCount* by 1" when picking up a coin.

Use Blocks: When using the information, drag the variable block into your code. So, if you want to show how many coins you've collected, you'd use the *coinCount* block.

Activity: Create a variable

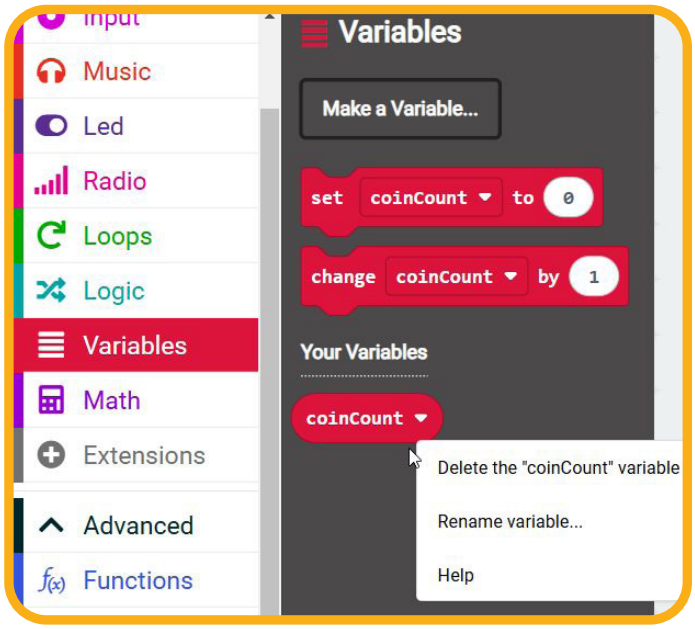
In this activity we will create a variable that contains numbers.

1. Click on the "*variables*" toolbox folder.
2. Next click on the "*Make a Variable*" button as shown in the image below.

3. Give your variable a name like "*coinCount*"

Once you click on the OK button, your new variable is saved and you will see it in the variables toolbox folder with a few new options.

Deleting a text variable and changing your variable name:



First, look at your newly created variable at the bottom of the image above. You can create as many variables as you like, which will all be listed under the "Your variables" list. You can drag them to many other blocks where you want your code to use the variable's value.

If you right-click on your variable name, you can delete the variable or change its name. You can see this in the image above.

Section 4 - Questions

1: What is a variable in coding?

2: What are the “on start” and “forever” blocks?

3: Why are variables useful??

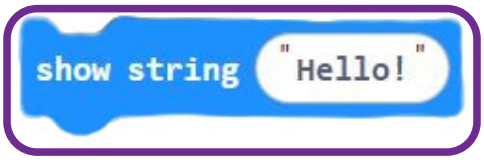
4: What are some rules for naming variables?

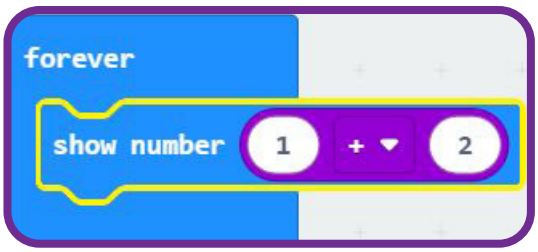
5: How do you change the value of a variable in block-based coding?

6: What is the difference between number and text variable types?

Section 5:

Creating and Working with Text Variables





Section 5: Creating and working with text variables.

Outcomes

- Understanding strings.
- Creating a text variables.
- Using a text variable in code.
- Joining text variables.

In coding text is also called strings because text is a string of different characters.

A newly created variable is always a number variable and not a *text* variable. The reason for this is that you, most of the time, create *number* variables.

If we want to use text, we need to tell the microcontroller that this is not a number but a *text* variable. It is easy to change a number variable to a *text* variable.

Activity: Create a text variable

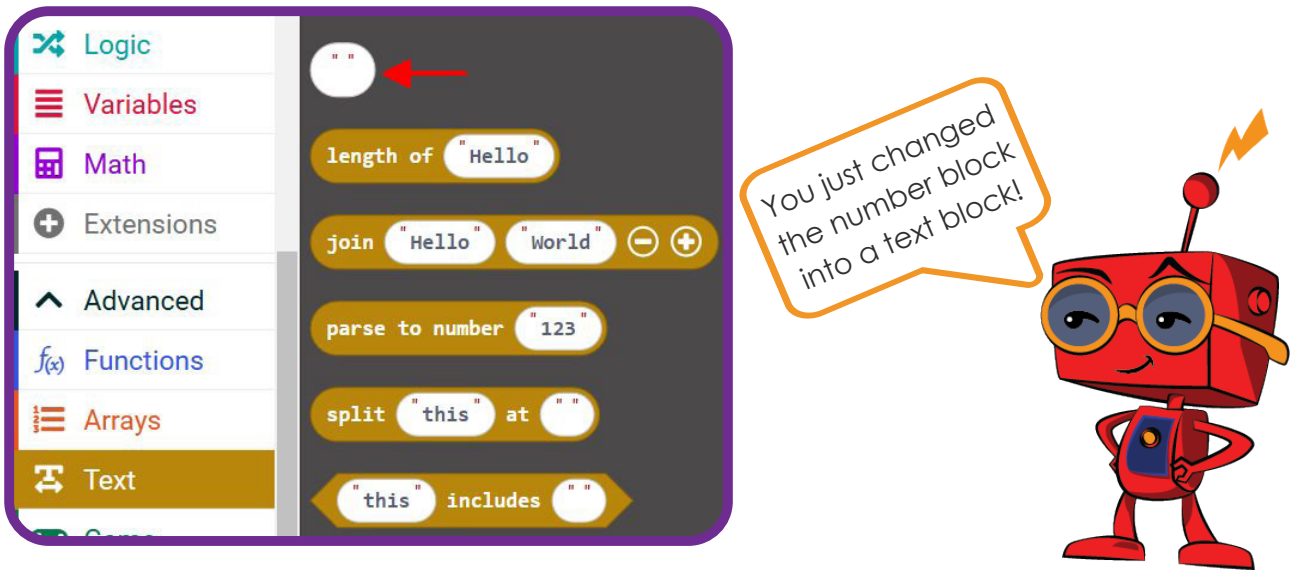
1. Create a new variable and call it “dogName”. Use the same steps as before when you create a new variable.

2. Once you created a variable, it is a *numbers* variable;
   continue with the steps below to make it a text variable.

3. Drag the new set block into your code work area.

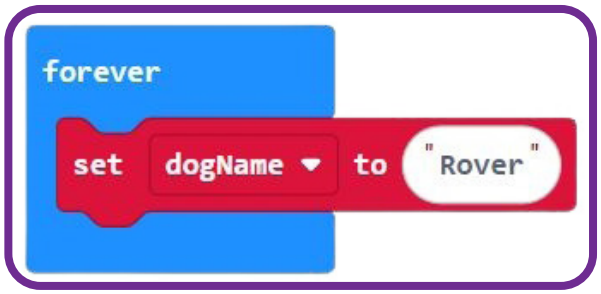
4. Click on the Text toolbox folder and find the “ “ block.

5. Drag the “ “ block into the value slot of your variable set block.



Store data in the variable:

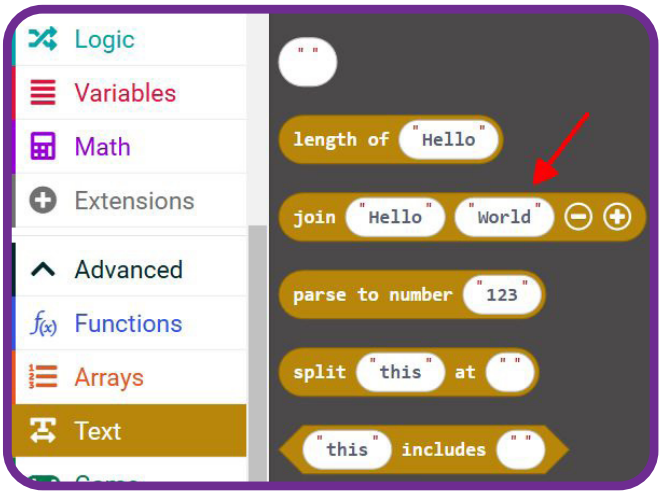
Now you can type a dogs name into the " " block as in the image above.



Joining text variables:

You can join *text* variables together. You can do that by using the *join* block from the *text* toolbox folder.

We are going to take a basic join block a step further and instead of typing two words we are going to enter two variables.

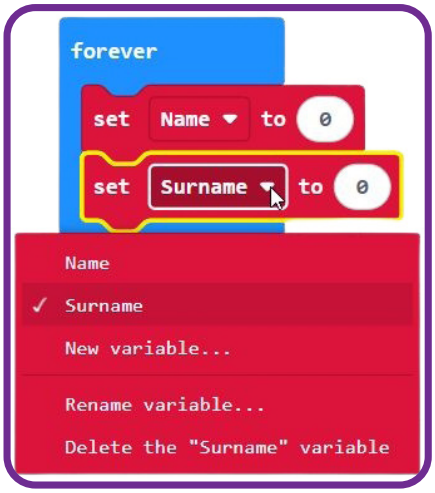


Activity: text variable, Joining text variables.

In this activity we are going to create two variables called *Name and Surname*. We will then set the value for the *name* variable "John" and *surname* "Smith"

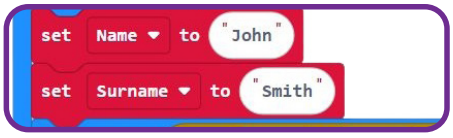
We will join them together to display on the micro:bit LED screen.

1. Use the steps earlier that explain how to create a variable and create a variable called "*Name*" and another called "*Surname*".

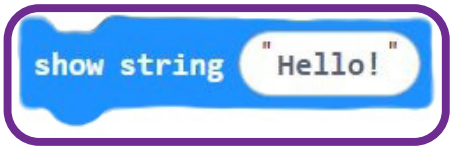


2. Drag the *variable set* blocks in the *variables* " toolbox folder to the code work area. Use the dropdown box inside the set block to choose the correct variables, "*name and surname*" as shown in the image below:

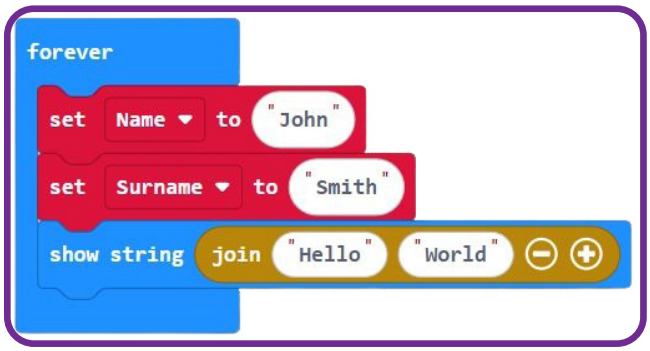
3. As shown earlier in this section, go to the *text* toolbox folder and move the " " block over to the code work area in to the *name* value area and enter the *name* value "John" in it. Do the same for the *surname* block and enter the *surname* value "Smith" in the value field.



4. Next drag a *show string* block, that you will find in the *basic* toolbox folder, in the work area. Drag a *join string*, from the *text* toolbox folder, into the value of the "*show string*" block.



5. The last step is to get both of your variables, from the *variables* toolbox folder, over into the "*show string*" values. Drag the "*name*" variable into the field that shows "*Hello*" and the "*surname*" variable into the field that shows "*World*".



Challenge.

1. Create the code that will display your own name and surname on the micro:bit LED screen.

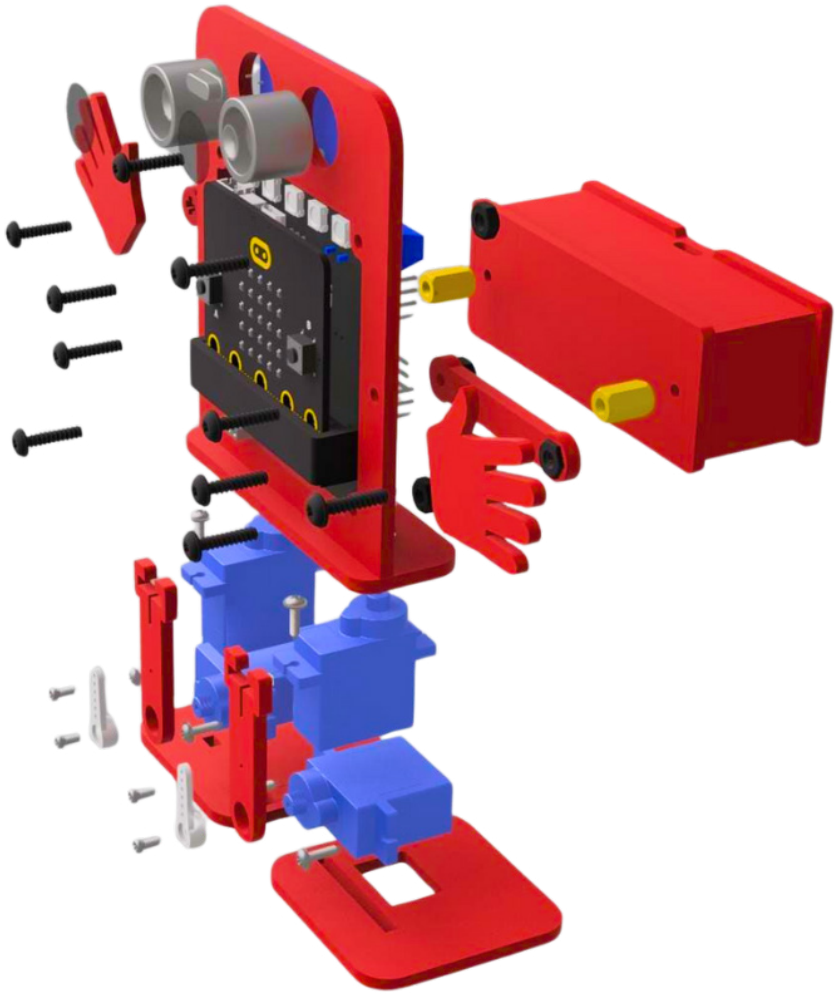
2. Change the code so that it only display your name.

Section 5 - Questions

1: What is another name for text in coding?
2: How do you create a text variable?
3: How do you join text variables together?
4: What is an example of a text variable?
5: If you want to show a text variable on the micro:bit LED screen, which block is useful?
6: What is the purpose of the set block when working with text?

Section 6:

Building a robot



Section 6: Building a robot

Outcomes

- Understand and using fasteners.
- Understanding and working with a servo motor.
- Understanding and working with an ultrasonic distance sensor.
- Build a small robot.

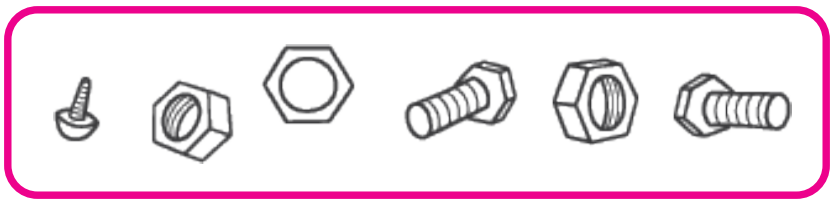
Now that you learned about a robots brain, the microcontroller, it is time to use it in a real little robot.

In this section we will build the robot step by step. Before we do that there is a couple of things you need to understand first.

Fasteners:

Fasteners in robotics are screws, nuts, bolts and anything else that can be used to hold everything together.

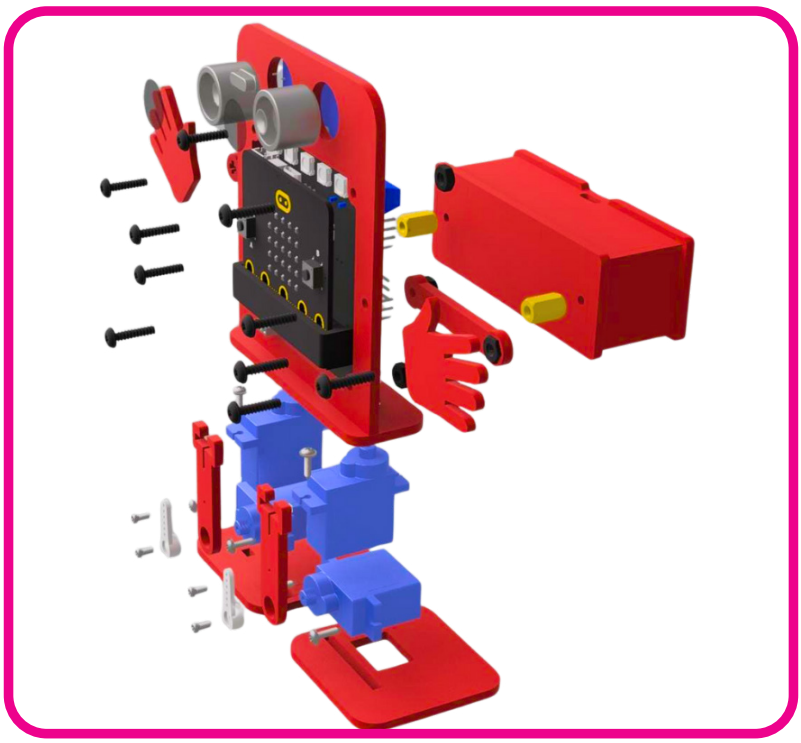
These fasteners are important because they prevent the robot's pieces, such as motors, gears, and panels, from falling apart when it moves around or does its job.



Structures:

Structures in robotics are like the body of a robot, which can also be thought of as its skeleton. One type of structure in robot cars is called a chassis. The chassis is where the wheels and other components are connected.

Fasteners are used to connect wheels, robot arms, microcontrollers, etc., to the structure, as seen in the image below:



Servos:

A servo in robotics is a type of motor that can be very precise about how it moves. A servo can move to a specific position. In robots, servos are used to make parts move just right — like the arms, legs, or even the head of a robot.

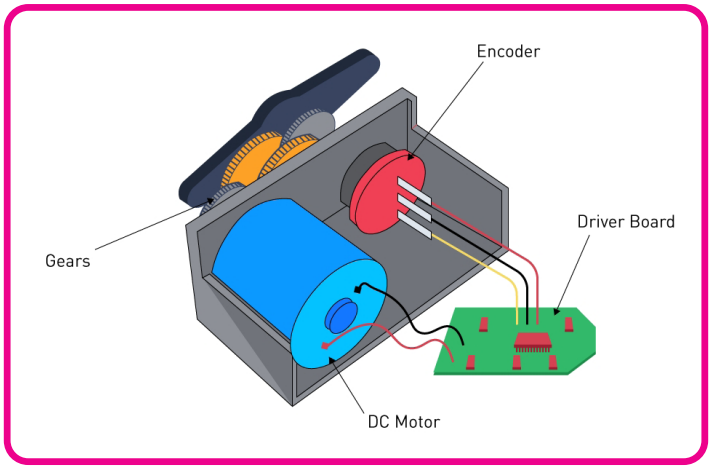
It can make a robot do a dance move and hit every step perfectly, because the servo knows precisely where to go. So, it helps robots to be more accurate when they do things like picking up objects or waving hello.

Servo motors are the backbone of your JitterBit robot, making its movements precise, controlled, and lifelike. In this section, we'll explore everything about servo motors, from how they work to how you can use them in programming.

What Are Servo Motors?

A servo motor is a type of motor that moves to a specific position or angle. Unlike regular motors that spin continuously, servo motors are designed for precise movements, making them ideal for robotics applications like controlling a robot's arms, legs, or even fingers.

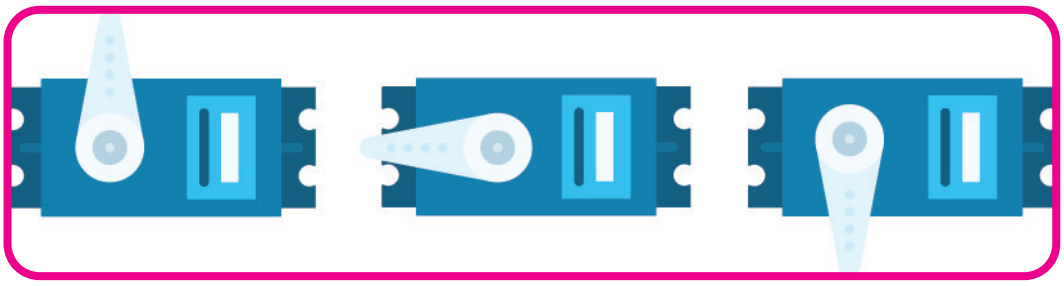
How Servo Motors Work:



Servo motors operate using three main components:

1. **DC Motor:** The core part of a servo, responsible for driving the movement
2. **Gearbox:** Reduces the speed of the motor, but increases its rotating force, allowing precise control over movements.
3. **Control Circuit:** Interprets signals from the micro:bit to determine the angle to which the motor should move.

Angles and Positions:



0 degrees 90 degrees 180 degrees

Servo motors can rotate to a specific angle, typically between 0° and 180°, based on the instructions sent by the micro:bit.

- **0°:** Fully to one side.
- **90°:** Neutral or center position.
- **180°:** Fully to the opposite side.

Real-Life Example: Imagine a servo motor as a steering wheel. Turning it slightly (0°) moves the wheels one way, turning it fully (180°) moves them the other way, and leaving it in the middle (90°) keeps the wheels straight.

Why Are Servo Motors Important for Robots?

Servo motors are the muscles of robots. They enable the JitterBit to:

- Lift its legs to scare.
- Move its feet to dance.
- Shake and wiggle during fun routines.

Without servo motors, your JitterBit would be static and unable to perform its unique movements.

Advantages of Servo Motors:

1. **Precision:** Servo motors can move to exact angles, making them perfect for detailed actions like kicking, waving, or grabbing objects
2. **Stability:** Once a servo reaches its target angle, it holds that position, ensuring the robot remains stable.
3. **Easy Control:** Servo motors are simple to program using block-based commands, making them beginner-friendly.
4. **Compact Design:** Small but powerful, servos fit easily into robots like the JitterBit.

Real-World Applications:

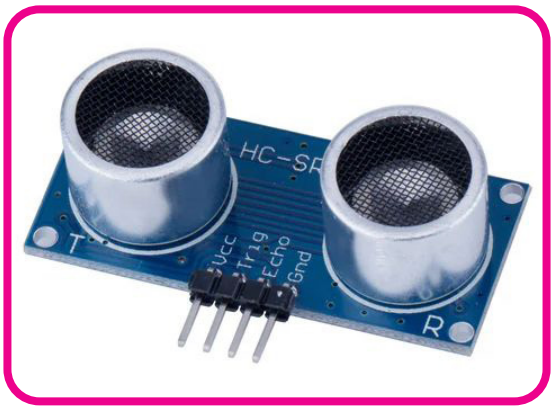
Servo motors are everywhere! Here are a few examples of where they're used:

- **Robotics:** Controlling arms, legs, or grippers.
- **RC Toys:** Steering wheels or controlling propellers in airplanes.
- **Industrial Machines:** Precision tasks like assembling small components.
- **Medical Devices:** Robotic surgery tools.

Distance sensor:

An ultrasonic distance sensor in robotics is like a robot's very own bat sense. It works by sending out little beeps of sound too high for us to hear. When these sound waves hit an object, they bounce back like an echo.

The robot then listens for this echo and can tell how long it took for the sound to come back. From that, it knows how far away the object is, kind of like playing a game where you shout into a canyon and listen for the echo to come back to guess how big the canyon is. This helps the robot know where to go without bumping into things, making it smarter and safer as it moves around.



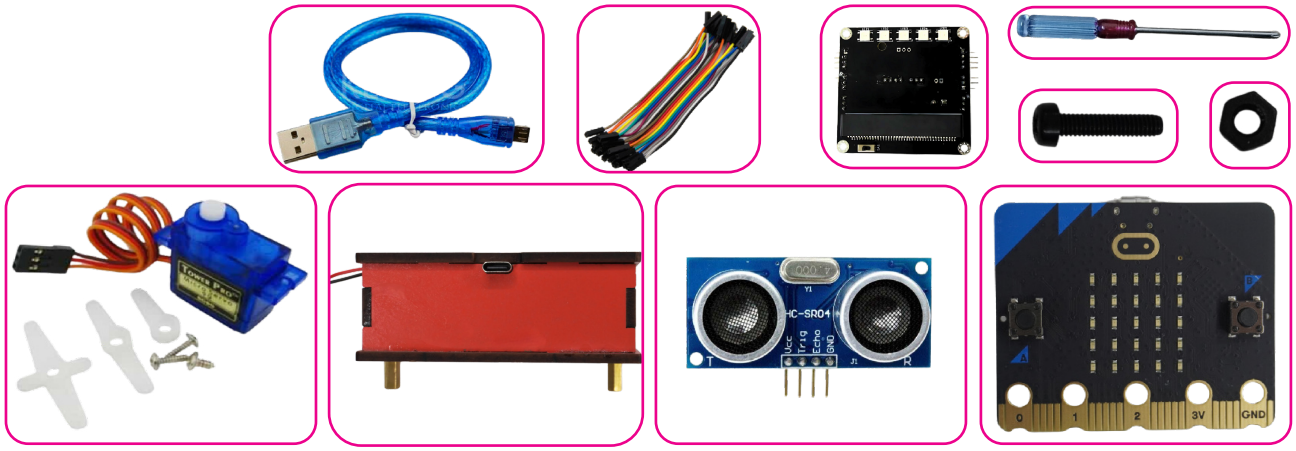
Activity: Building the JitterBit Robot

What is in the BOX

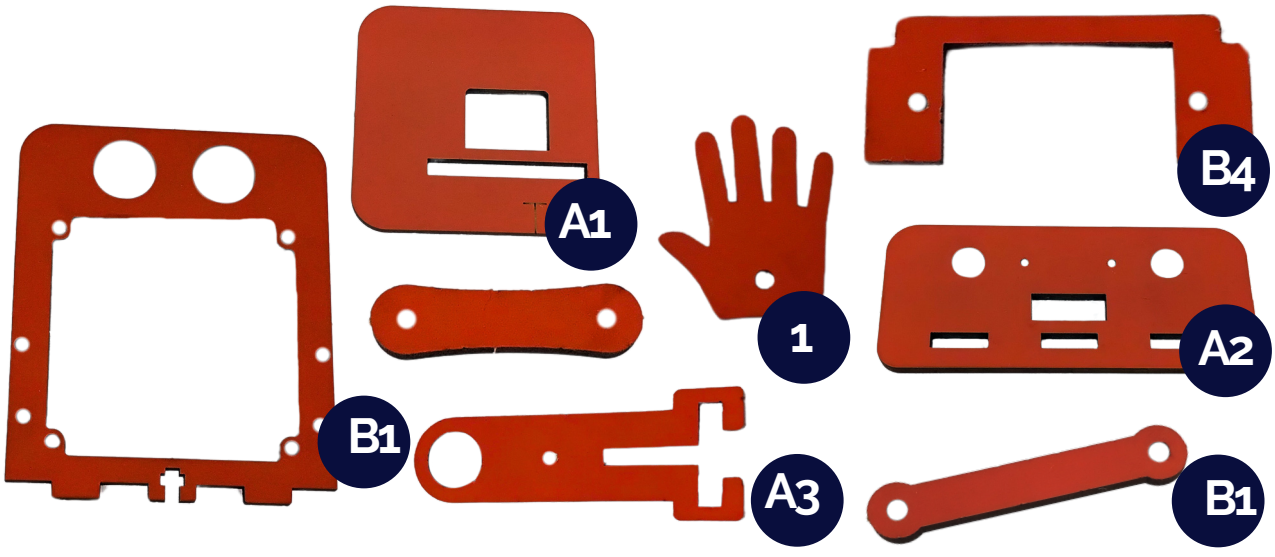
12 wooden parts
4 x 180 servo's
1 x ultrasonic
10 x nylon screws
8 x nylon nuts
1 x micro:bit

1 x micro:bit board & battery
10 x 10cm female to female jumper wires
1 x screwdriver
1 x micro cable

Components:



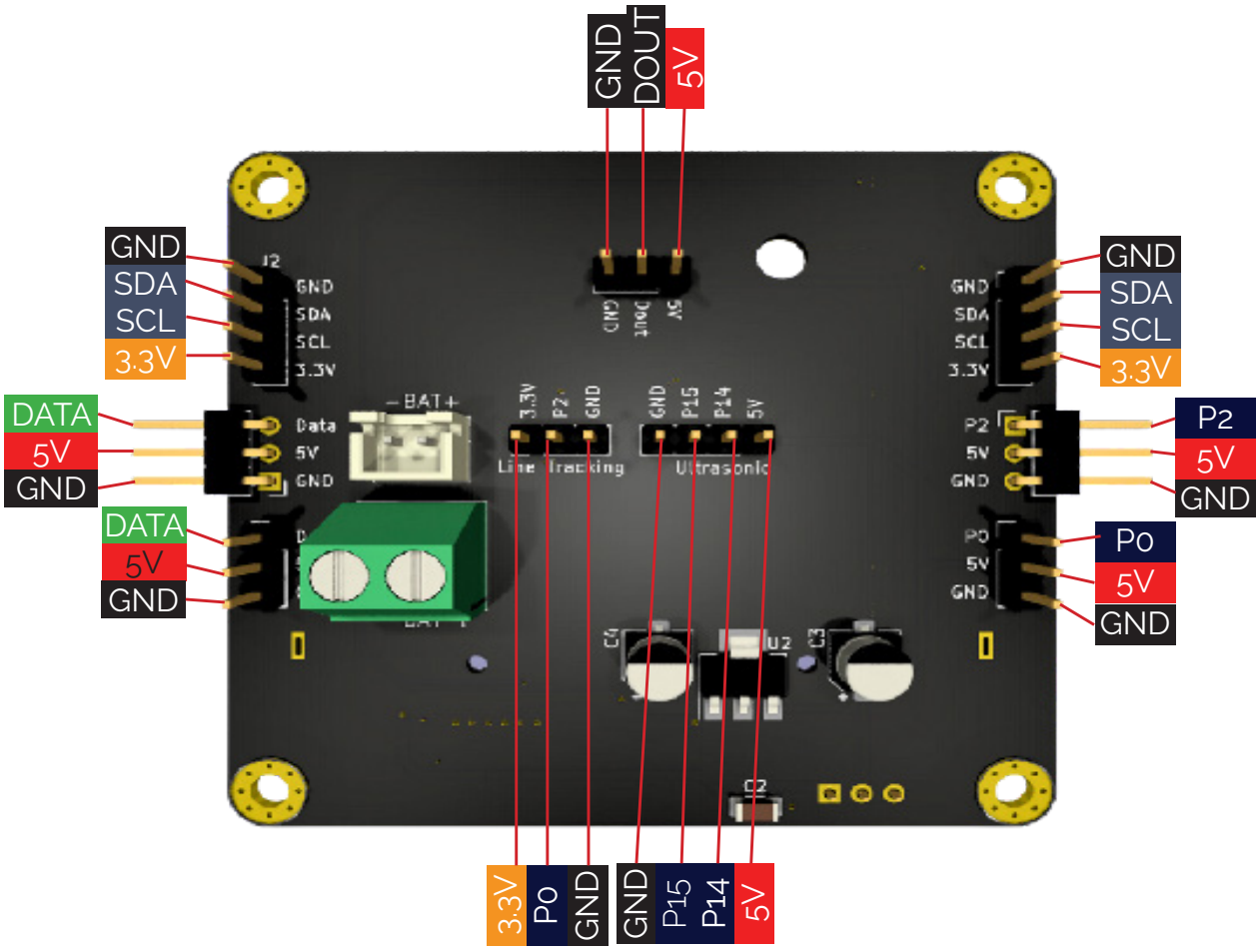
x-o skeleton:



Features:

- **Scare Function:** Clap loudly, and Jitter will look shocked!
- **Tickle Mode:** Touch the icon above the LED matrix (its belly) and watch it react.
- **Obstacle Avoidance:** Jitter reverses when something gets closer than 5cm to its eyes.
- **Light Sensor:** The LED matrix shows a sun in bright light and a red circle in the dark.
- **Dance Mode:** Press button B to make the robot dance.
- **Jitter Mode:** Press button A to make the robot jitter.
- **Servo Centering:** Press buttons A and B together to center the servos.
- **Read Sensors:** Check temperature, magnetic wave intensity, and orientation with the G-sensor.
- **Control the LED Matrix:** Display different arrows and symbols.
- **Send Commands:** Make Jitter *jitter*, *shuffle*, and perform a *little dance* with just a tap.

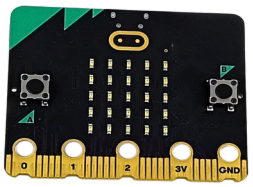
micro:bit Board Pinouts:



Step 1

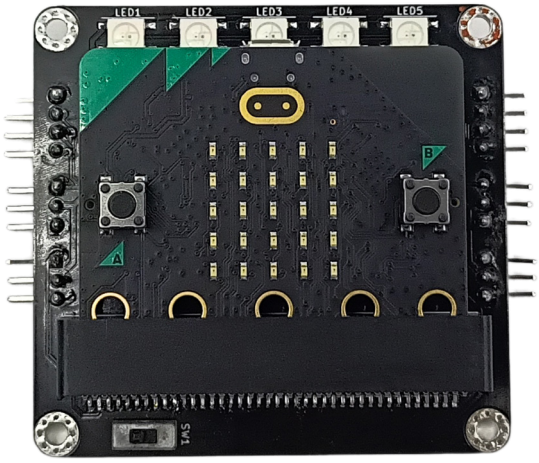
What's needed:

xo-core



micro:bit

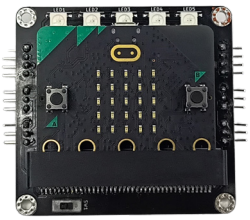
Insert the micro:bit into the slot of the black connector on the X-O-Core. Press down firmly to ensure the micro:bit is securely seated in the connector. This provides a stable connection.



When inserting the micro:bit into the slot, ensure the white LED faces forward. This ensures the correct orientation for proper functionality.

Step 2

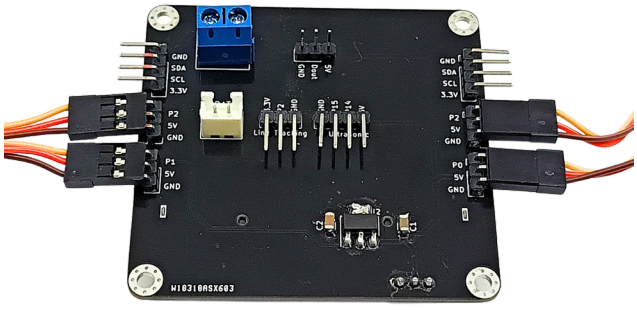
What's needed:



step 1

4 x servos

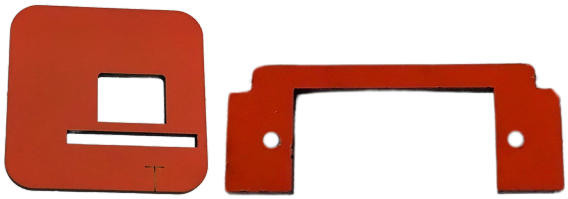
To set up the servos, connect the first servo's brown wire to Ground (GND), red wire to 5V, and yellow wire to P2. For the second servo, connect brown to Ground (GND), red to 5V, and yellow to P1. For the third servo, use the same connections as the first one.



For the fourth servo, connect brown to Ground (GND), red to 5V, and yellow to P0. Then, turn your board around and press both black buttons simultaneously to calibrate the servos. Finally, remove all the servos.

Step 3

What's needed:



2x board A1 2x board B4

Take the two pieces labeled Board A1 and Board B4. Align each Board B4 with Board A1 and press them together until they are securely connected.

Press firmly when inserting the pieces into each other, as they may fit tight. Applying steady pressure will help ensure they connect securely without damaging the parts.

Step 4

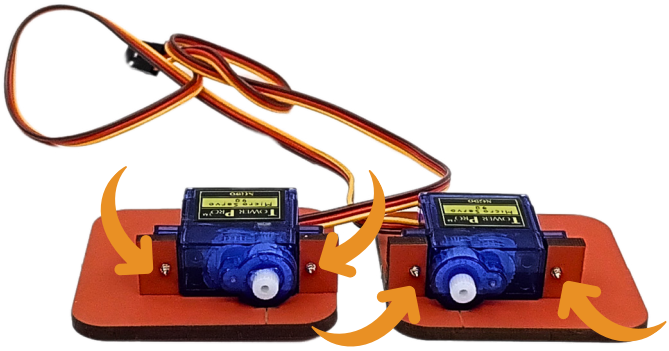
What's needed:

2x servos

step3

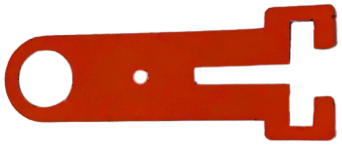
Place the servos on Board A1, ensuring that the shafts align with the "T" symbols on the board.

Then, insert two long screws through the holes on Board B4 and into the servos to secure them in place.



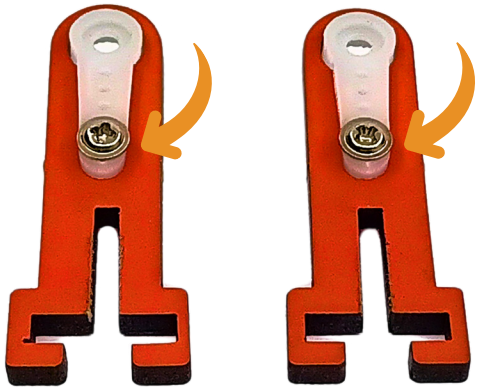
Step 5

What's needed:



2x board A3

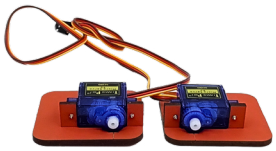
Take the teardrop bracket that comes with the servos. Insert the rounded part of the bracket into the hole on Board A3, and align the second small hole with the small hole on Board A3.
Then, insert a long metal screw through both aligned holes and tighten it to secure the bracket in place.



Ensure the screw head is facing forward, as shown in the picture. This positioning will ensure proper assembly and secure fastening.

Step 6

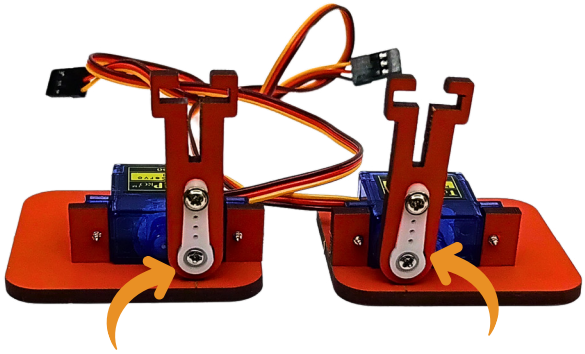
What's needed:



step 4

step 5

Insert the shafts of the servo motors from step 4 into the holes on the boards from step 5. Then, use two short metal screws to secure the parts together by tightening the screws into the holes on the servo shafts.

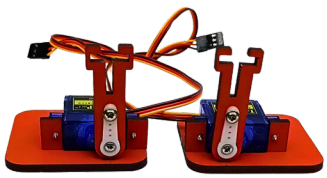


Ensure the white bracket is facing forward, as shown in the instructions. This will ensure it is correctly aligned for the correct fit and function.

Step 7

What's needed:

2x Servo



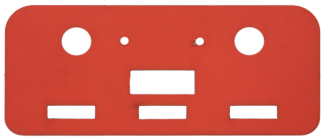
Step 6

Slide the flat part of the servo into the groove on Board A3. Press the board firmly against the servo, then insert one of the stainless steel screws through the hole in the flat part of the servo and into the small groove on Board A3 to secure them together.

When mounting the servo, ensure the wire is positioned at the back. This will help keep the wire neatly arranged and prevent it from interfering with other components.

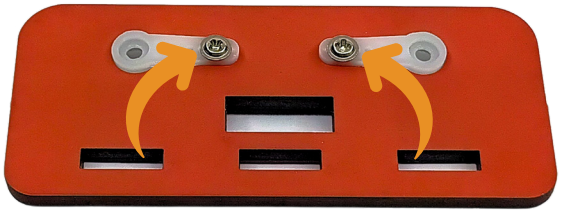
Step 8

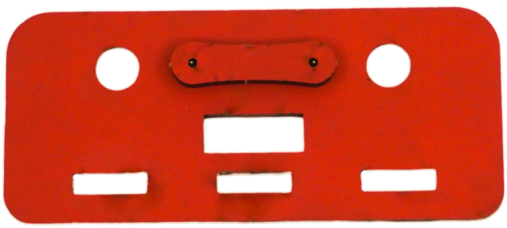
What's needed:



board A2

Insert the teardrop servo brackets into the large holes on Board A2 and align the small holes. Fasten with longer metal screws. When the screws push through the back, place the small board behind to secure them, then tighten fully.

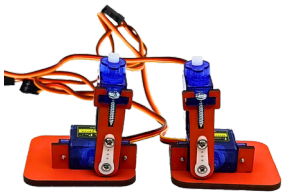




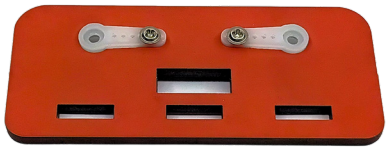
The teardrop brackets must be positioned on top of Board A2. Ensure they are securely placed in the correct alignment with the other components.

Step 9

What's needed:

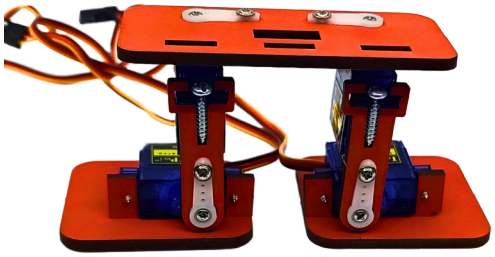


step 7



step 8

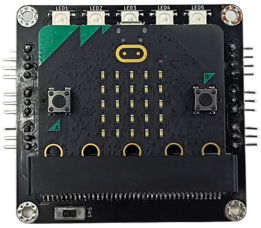
Insert the shafts of the servos, with the shafts pointing upwards on the legs, into the holes of the teardrop servo brackets from step 8. Use two short metal screws to secure the servos to the brackets by tightening the screws into the holes on the servo shafts.



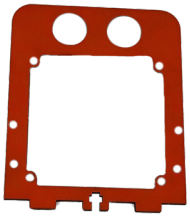
The front of the legs must be positioned on the side of the three rectangular holes on Board A2. Make sure they are correctly aligned for secure attachment and proper function.

Step 10

What's needed:



step 1

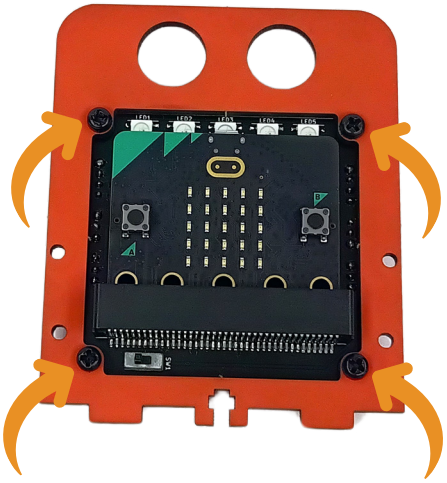


board B1

2x screws

2x nuts

Place the X-O-Core on the back of Board B3, ensuring that the micro:bit is visible through the rectangular hole on the front. Align the four corner holes of the X-O-Core with the four holes around the rectangular hole on Board B3. Use four plastic screws and nuts to secure the X-O-Core to the board by inserting the screws through the aligned holes and fastening them with the nuts.



Ensure the five white LEDs on the X-O-Core are positioned at the top. This ensures proper visibility and alignment when assembling the components.

Step 11

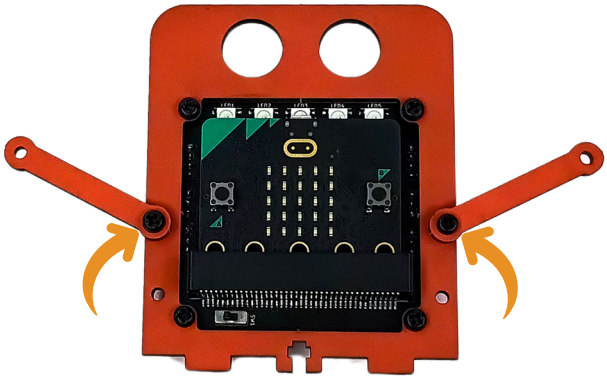
What's needed:

2x board B2

2x screws

2x nuts

Align one hole of Board B1 with the top outer holes of Board B3. Use two plastic screws and nuts to secure the boards together by inserting the screws through the aligned holes and fastening them with the nuts.



Board B1 must be positioned in front of Board B3, with the screw heads facing the front. This ensures proper alignment and secure attachment between the boards.

Step 12

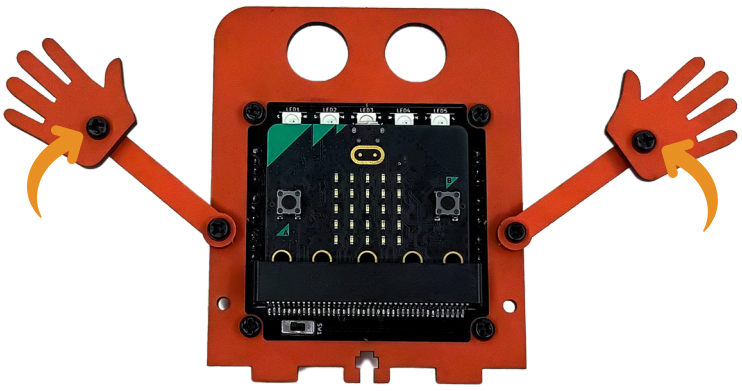
What's needed:

Hands

2x Screws

2x Nuts

Align the holes of the hands with the open holes on Board B1. Use two screws and nuts to secure the hands to the board by inserting the screws through the aligned holes and fastening them with the nuts.



Make sure the thumbs are facing upwards. This will ensure the correct orientation for proper assembly and functionality.

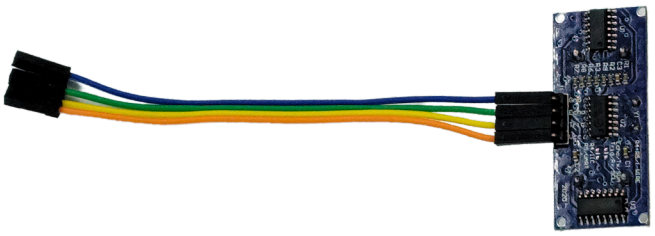
Step 13

What's needed:

Ultrasonic


4x rainbow cables

Take four rainbow cables and press the ends of each cable onto the connector pins of the ultrasonic sensor. Make sure each cable is securely attached to the corresponding pin.

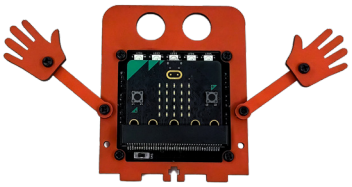


The colour of the components does not matter. What's important is making sure they are connected and positioned correctly for proper functionality.

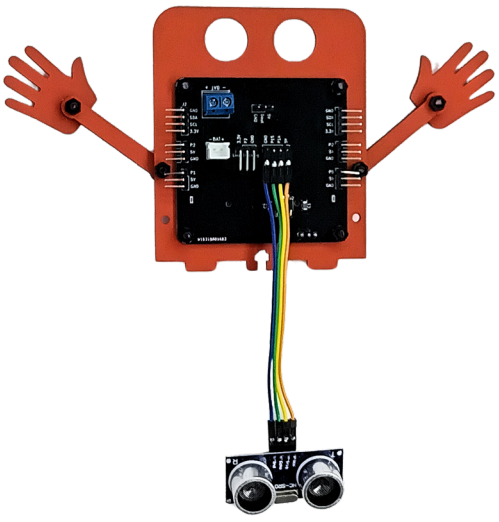
Step 14

What's needed:


Step 13


Step 12

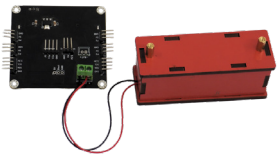
Press the four open pins of the wires connected to the ultrasonic sensor onto the four pins of the X-O-Core that are labelled "ultrasonic sensor." Make sure each wire is securely attached to the corresponding pin. The wires must be connected as follows:
GND = > GND
ECHO => P15
TRIG = > P14
VCC => 5V

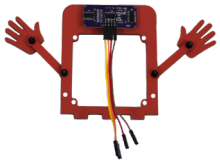


Ensure the wire colours are in the same order on both connector pin sets. This ensures consistent and correct wiring for proper functionality.

Step 15

What's needed:

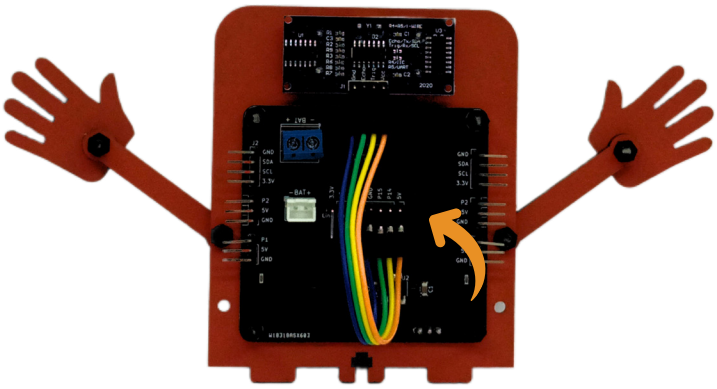

Step 1


Step 14


4x Screws


4x Nuts

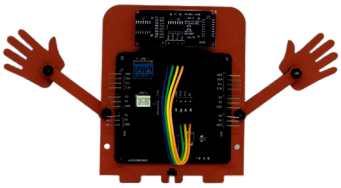
Insert the round silver parts of the ultrasonic sensor into the large holes on Board B3, ensuring the connector pins are pointing downwards. This will ensure proper alignment for the sensor.



Press firmly on the ultrasonic sensor, as it is a tight fit. Be sure to apply pressure at the back of the silver parts of the ultrasonic sensor to ensure it fits securely into place.

Step 16

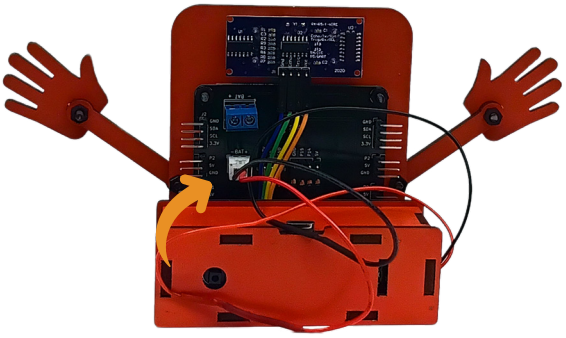
What's needed:


step 15


battery pack


2x Screws

Connect the battery box to the X-O-core by connecting the white connector of the wire to the white connector on the X-O-core. Cut two plastic screws to make them 5mm shorter. Neatly fold the wires at the back, keeping them in place. While holding the wires down, position the battery over the wires, aligning the golden spacers with the two open holes on Board B3. Use the shorter screws to fasten the battery onto Board B3 by inserting them into the golden metal spacers and tightening them securely.



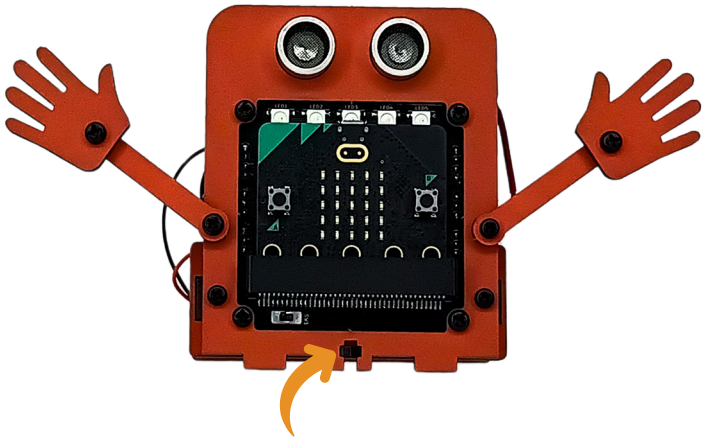
Make sure that the USB is showing upwards. That way will make it easy to charge the battery.

Step 17

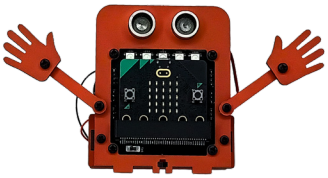
1x nut

What's needed:

Insert the nylon nut in the slot on the bottom.



Step 18



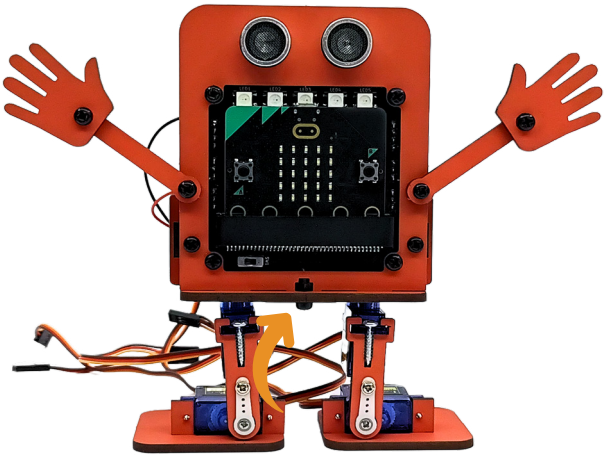
Step 17

Step 9

1x Screw

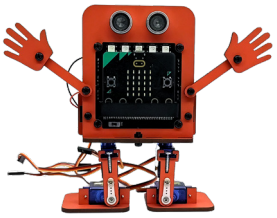
What's needed:

Press the three points on the bottom of Board B3 into the three rectangular holes on Board A2, ensuring they fit securely into place. Use a nylon screw to secure it in place.



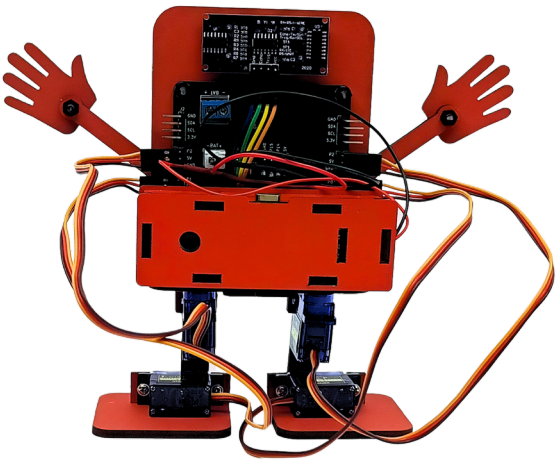
Step 19

What's needed:



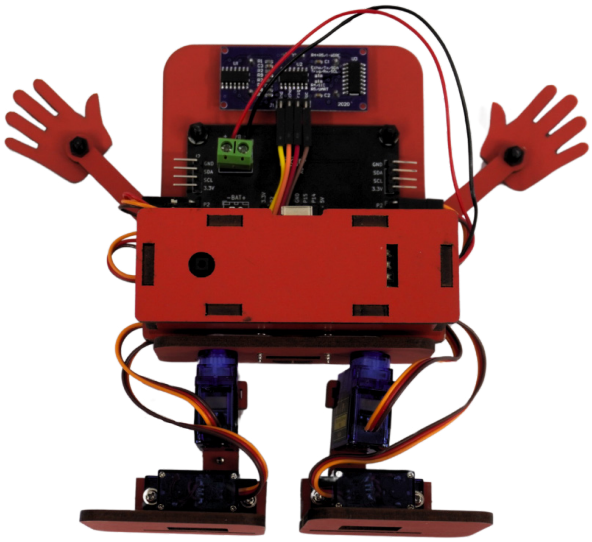
Step 18

Connect the wires of the four servos to the corresponding connector pin sets on the X-O-Core, labelled P0, P1, and P2. The two bottom servos should be connected to P0 and P1, while the others should be connected to P2. Make sure each wire is securely attached to the correct pin.



Since this is a tight fit, press firmly to ensure the parts connect securely. Applying steady, even pressure will help avoid damaging the components.

Step 20



For the last step, organize the wires neatly, and then you are ready to play with the JitterBit.

Great job! Your JitterBit is now complete. Simply switch it on and enjoy playing with your creation! **Have fun!**

Download the JitterBit default program:

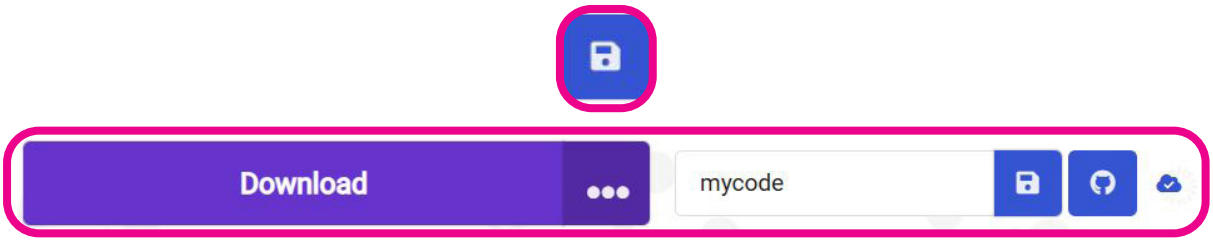
Before we build the robot, we need to add the JitterBit code to the microcontroller. This code was written by Bot Shop and explores all the JitterBit abilities.

The code can be downloaded from here: botshop.co.za/jitterbit

Saving your MakeCode projects on your computer:

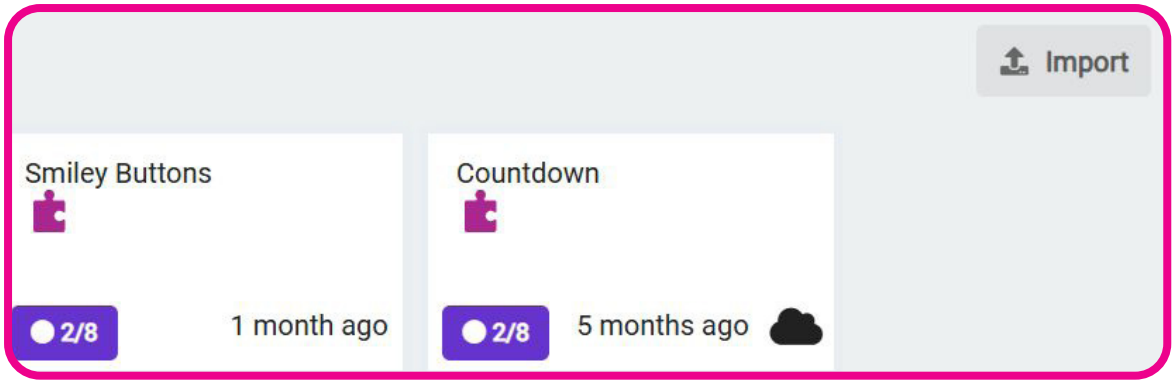
All your projects are stored on the *MakeCode* website, but you can also save them on your computer. There are many reasons why you want to do that, including being able to send the code to other people.

Next to the Download button in *MakeCode* is a text area where you can give a name to the code you want to download to your computer. I called mine "*my code*." Once you give the code file a name, you can click on the download button.

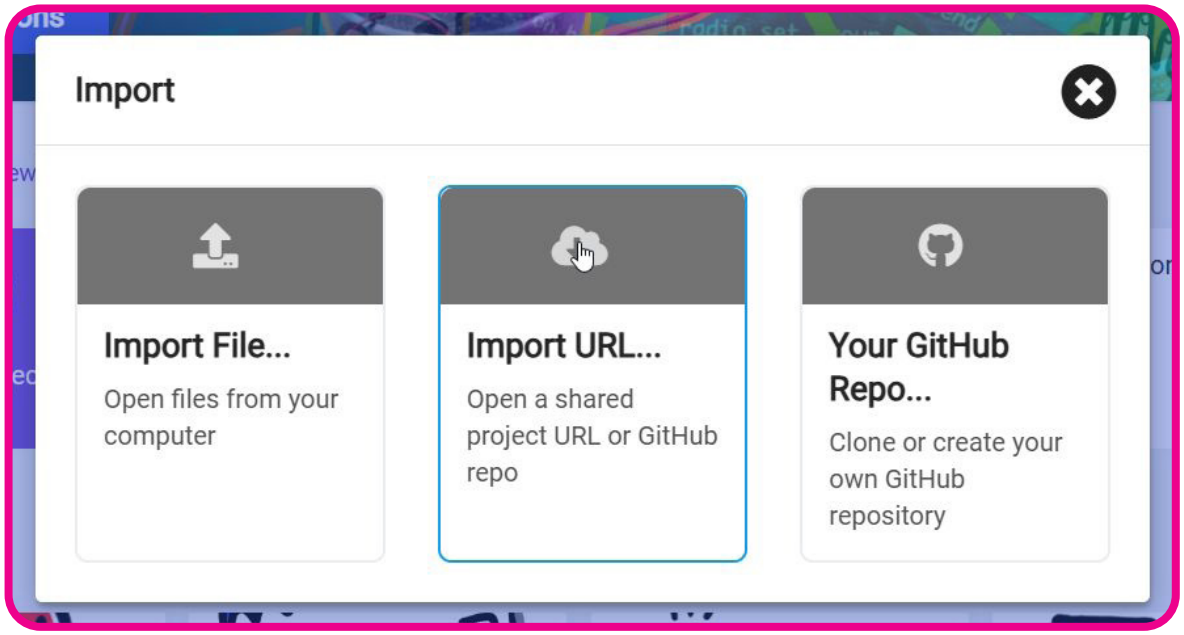


Uploading saved code to MakeCode:

- Importing (uploading a saved project) back into your *Makecode* is also easy.
- The first thing to do is go to the *MakeCode* home page, where you can create new projects.
- Click on the "*Import*" button as shown in the image below.



From the screen that pops up select "*Import file*", select the file to import on your computer and click the "Go ahead button.



A new project will be created with the same name as your file (you can change the project name at any time), and the code will be placed in it.

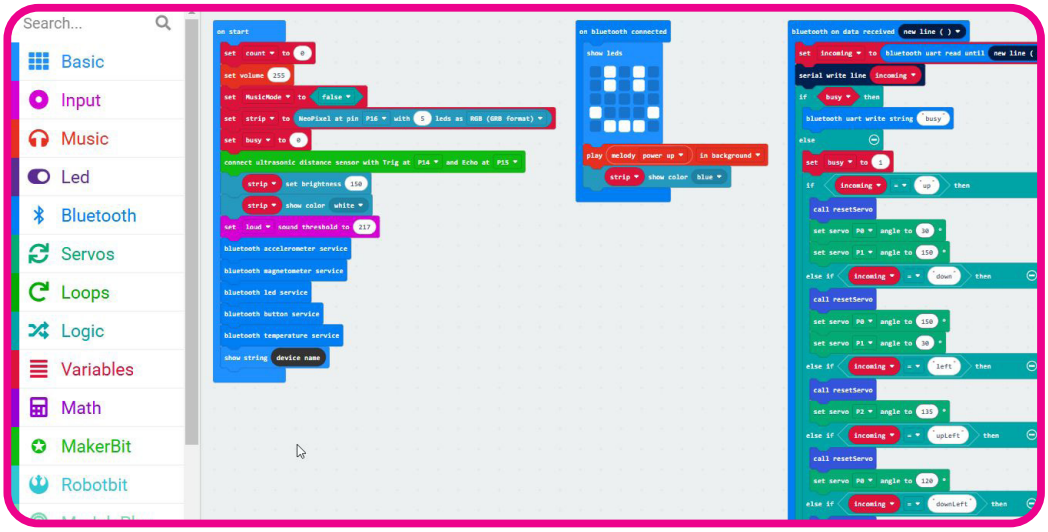
Activity: Upload the demo code to your JitterBit

Demo code demonstrates the functions of the JitterBit. This code is very useful, because it shows all the blocks we used and how we used them. There is a lot of code, which is great, because you can create your own code from these examples or modify the code to your liking.

If you mess up the code, you can upload the original code again and try once more. As mentioned earlier, the demo source code is an excellent example of uploading *Makecode* files to your *Makecode* software. We wrote demo code for your JitterBit to help you quickly start experimenting with the robot.

1. **Download the JitterBit code from *www.botshop.co.za/jitterbit* to your computer.**

2. Upload the code into *Makecode* by clicking the "*Import*" button on the *Makecode* home page. As shown in the image below, it contains many blocks and numerous examples you can work with.



3. Download the code to your JitterBit by plugging the micro:bit into your computer and clicking the "Download" button.

Section 6: Questions

1: What are fasteners in robotics?

2: What is a servo motor?

3: What does an ultrasonic distance sensor do?

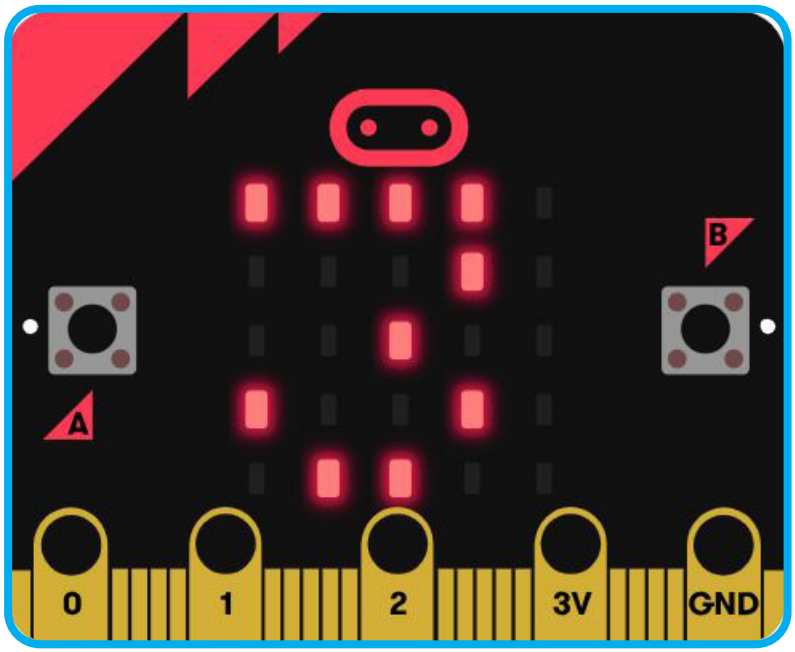
4: What is the role of fasteners when building a robot?

5: How does a servo motor help a robot?

6: Why is a distance sensor useful for a robot?

Section 7:

Inputs and Outputs



Section 7: Inputs and outputs

Outcomes

- Understand what an input is.
- Understand what an output is
- Code a button
- Code a speaker

Understanding inputs and outputs:

Inputs:

- **What Are Inputs?**
  - Inputs are like the robot's senses. They help the microcontroller understand what's going on outside.
  - Think of inputs as the robot's eyes, ears, or touch sensors.

- **Examples:**
  - **Buttons:** When you press a button, you give the microcontroller input. It's like telling the robot, "Hey, I want you to do something now!"
  - **Sensors:** Sensors can feel heat and light or even detect when something is close. If a sensor feels heat, it tells the microcontroller, "It's hot around here!". You already used a distance sensor, you now also know that a distance sensor is an input device.

- **How They Work:**
  A signal goes into the microcontroller when you touch a button or something that triggers a sensor. It's like whispering a secret to your friend, saying, "This just happened."

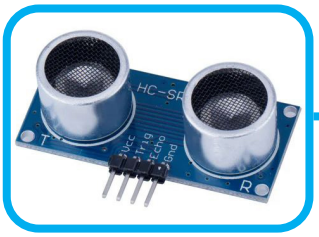
Outputs:

What Are Outputs?

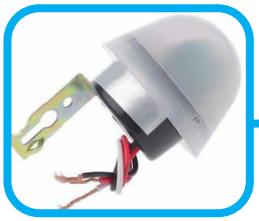
- Outputs are actions the microcontroller can take. They are like the robot's ability to move its arms, make sounds, or light up.

- **Examples:**
  - **LED Lights:** When the microcontroller decides to show something, it might light up an LED. That's like the robot blinking its eyes to tell you it's happy or excited.
  - **Motors:** If your robot has a motor connected, the microcontroller can make it move. It's like the robot is waving or spinning around.
  - **Servos** as used in building the robot.
  - **Speakers or Buzzers:** The microcontroller can make noise through a speaker or buzzer, like when your robot wants to sing or beep an alarm.

Inputs



Ultrasonic



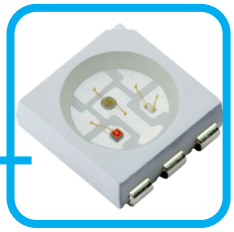
Day / night sensor

Outputs

Buzzer

Servo

LED

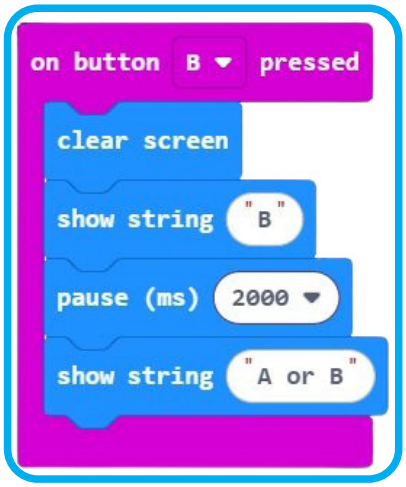


Activity: Ask a question and wait for input

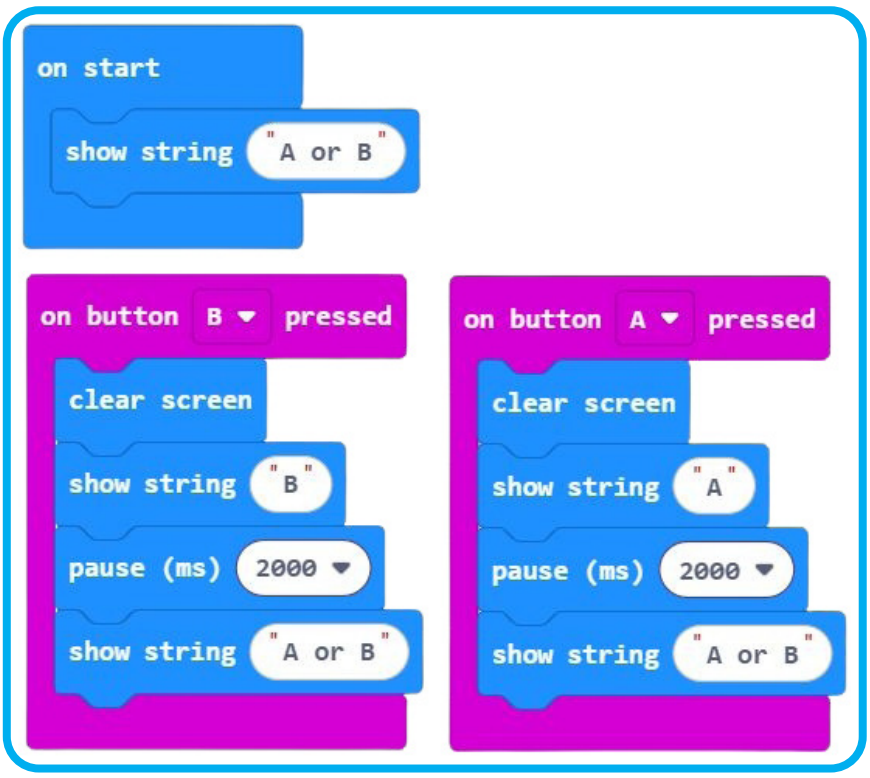
In this activity, we will write "a" or "b" on the LED screen. We'll use the buttons as inputs and the LED screen as the output. The micro:bit has two buttons labeled "a" and "b." When you press one of the buttons, the microcontroller will display which button you pressed.

First, drag two "*on button press*" blocks to the work area. You will find them in the *Input* toolbox folder.

These blocks don't fit in the "on start" or "forever" main blocks. They are among the few blocks that work independently, constantly monitoring the two buttons for a press in the background.



1. First, drag two "*On button Press*" blocks to the work area. You will find them in the *Input* toolbox folder.
2. Use the dropdown in the "*on button press*" block to choose "A" in the first block and "B" in the second one.
3. Next, drag the "*clear screen*" block (this block will remove all text currently displayed on your LED screen), the "*show string*" block, and the "*pause*" block into the workspace. They are in the *Input* toolbox folder.
4. Also, add a "*show string*" block inside the "*on start*" block. The "*on start*" block runs the block inside it only once, and it does so only when you power up your micro:bit. We added the "*show string*" in the "on start" block so that the text will appear once on startup.
5. The last step is to change all the text in the "*show string*" blocks so that it matches the text in the image below:


on start
show string "A or B"
on button B pressed
clear screen
show string "B"
pause (ms) 2000
show string "A or B"
on button A pressed
clear screen
show string "A"
pause (ms) 2000
show string "A or B"

Section 7: Questions

1: What are inputs in robotics?

2: Give three examples of inputs:

3: What are outputs in robotics?

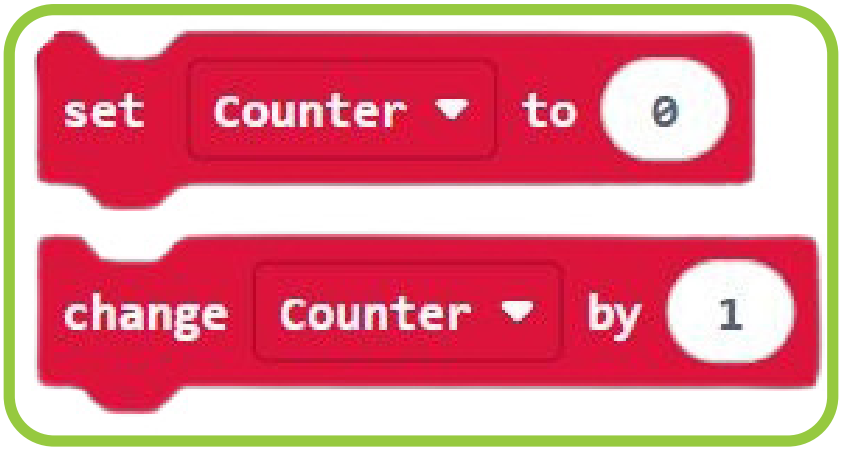
4: Give three examples of outputs:

5: When do the “on button press” blocks work?

6: How can we use the LED screen as output?

Section 8:

Number Variables



Section 8: Number variables.

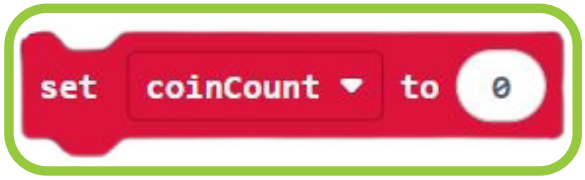
Outcomes

- Change number variable values.
- Create a number variable.
- Understand counters

We looked at text variables in the previous section and they are not difficult to do. Number variables is even easier because number variables is the default. This means that you do not have to change the variable type, it is already a number variable.

Important: It is important to know that you can not do mathematics with the text type variable, even if you put only numbers in the value field of a text variable. You must use a number type variables to do mathematics.

Setting the value for your number variable:



Once you created your variable you can add a default value to it, this means that every time you power up your microcontroller that value will be automatically set for that variable.

Important: The value can only be changed once the set block is dragged into the code area and not while it is inside the variable toolbox folder.

In the case of a coin collecting robot it will be "0", because your robot did not collect any coins yet.

You can add any number in the block shown above, but for now "0" will work the best. Changing the variable value automatically.

This block allows you to instruct the microcontroller to add a number to the number already in a variable.

If you start your microcontroller, the *coinCount* variable will automatically be "0."

Then, when you use the "change" block above, it will add one to the *coinCount* value (0+1)=1.

When that block is used again, the value will now be the value of *coinCount* (that is now 1), and 1 will be added again. The value will now be 1+1+2.

Activity: Number variables: Create a variable and use it to make a robot work like a counter.

In this activity we will create a counter that will count from "0", it will not stop counting until the microcontroller is turned off.

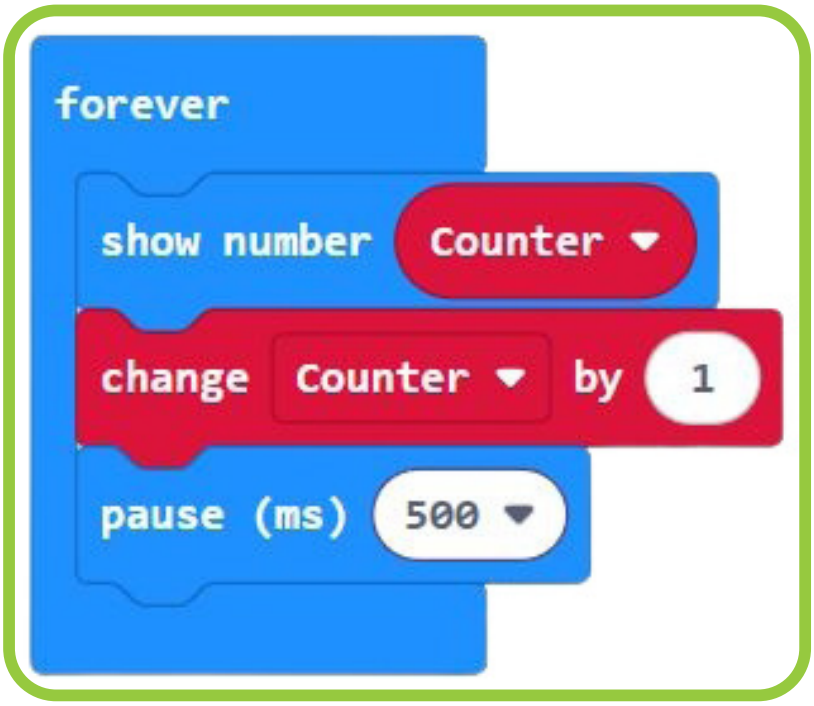
1. Create a new variable called *Counter*.
   Your blocks should look like below:

2. Move the change variable block to your work area in the software and place them inside the "forever" block

3. Inside the Basic toolbox folder find the "*pause*" block and the "*show number*" block and place them also in the "*forever*" block.

4. Move the variable in the variable list inside the variables toolbox folder over to the workspace and drag it to the "*show numbers*" block.

It should now look like the image below:



Upload your code to the microcontroller or view the simulation on the right side of the *MakeCode* software.

Notice how the micro:bit will display the counter on the LED screen .

Challenge

- Change the value in the "Pause" block to make the counter count faster or slower.
- Change the value in the "show number" block to change the counting 0,1,2,3,4, etc. to 0,2,4,6,etc.

Section 8: Questions

1: What is the default type of a newly created variable?

2: Can you do mathematics with a text variable?

3: How do you set a default value for a number variable?

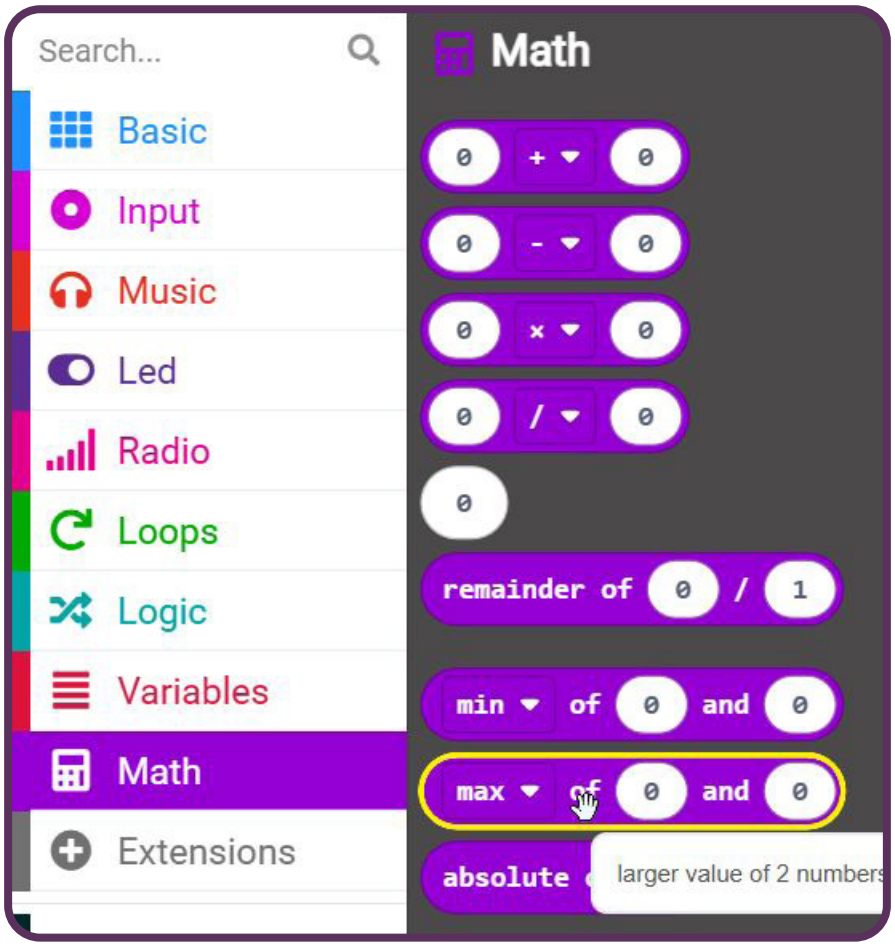
4: How can a variable value change automatically?

5: What is a counter in coding?

6: What is the purpose of the "pause" block when you are working with counters?

Section 9:

Use Operators like Add, Subtract, Multiply



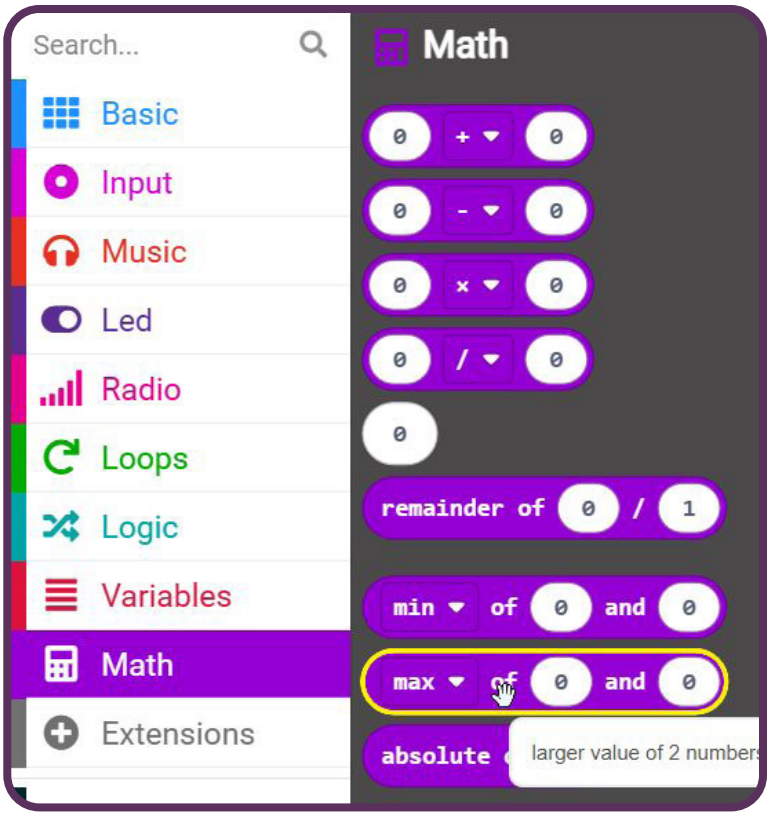
Section 9: Use operators like add, subtract, multiply.

Outcomes

- Adding
- Subtracting
- Multiplying
- Dividing
- Display mathematics results on screen

All microcontrollers can do mathematics. We can use the micro:bit brain to display mathematics on its screen.

Below is the blocks we will use: The first top 4 blocks will be used in this section.



1. Adding (+)

- **What it does:**
  Combines two numbers to make a bigger number.
- **How it looks:**
  - A purple block with a + in the middle.
  - ***Example***: Drag a + block, then put 5 and 3 in it. It makes 8.
- **What you can do:**
  Add scores in a game.

2. Subtracting (-)

- **What it does:**
  Takes one number away from another.
- **How it looks:**
  - A purple block with a - in the middle.
  - ***Example***: Drag a - block, then put 10 and 4. It makes 6.
- **What you can do:**
  Find out how many candies are left if you ate some.

3. Multiplying (×)

- **What it does:**
  Adds the same number many times.
- **How it looks:**
  - A purple block with a × in the middle.
  - ***Example***: Drag a × block, then put 3 and 4. It makes 12.
- **What you can do:**
  Multiply to find out the total if you get 4 apples for each of 3 baskets.

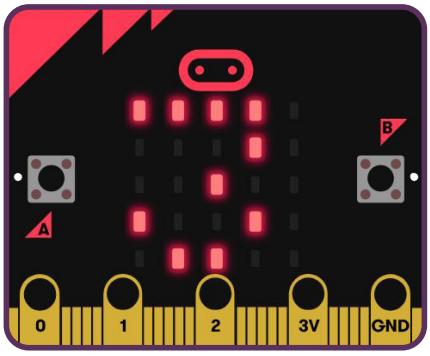
4. Dividing (÷)

- **What it does:**
  Splits a number into equal groups.
- **How it looks:**
  - A purple block with a ÷ in the middle.
  - ***Example***: Drag a ÷ block, then put 12 and 3. It makes 4.
- **What you can do:**
  Share candies equally among friends.

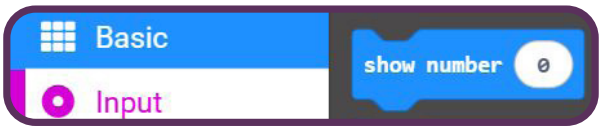
Displaying mathematical results

The mathematics has been done inside the robot brain, but how can we see the results for ourselves?

The micro:bit uses a lot of little lights (called LEDs) to display numbers like 3, as shown below:



From now on, we will call the robot brain a microcontroller because we are talking about microcontrollers in robotics.



We must instruct the microcontroller to display the numbers on the LED screen. We do that as follows:

Display on screen:

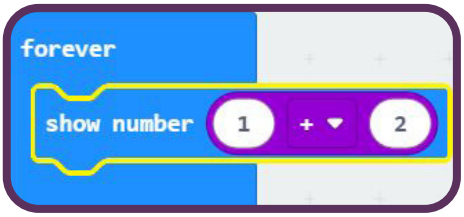
- **What it does:**
  Display numbers, text, etc. on the micro:bit screen
- **How it looks**
  - Available inside the Basic block toolbox folder.
  - A blue block with this text "show number" with a space to enter a number in the middle.
- **What you can do:**
  Display numbers, text ,and basic images like a heart or X etc.

Joining things together

You have two blocks on your computer or tablet screen. One is a calculation block, and the other is a display box.

We can join them together by dragging the "*calculation*" block to the "*display*" block.

It will look like the image.



Section 9: Questions

1: What are math operators used for?

2: Name the four math operators.

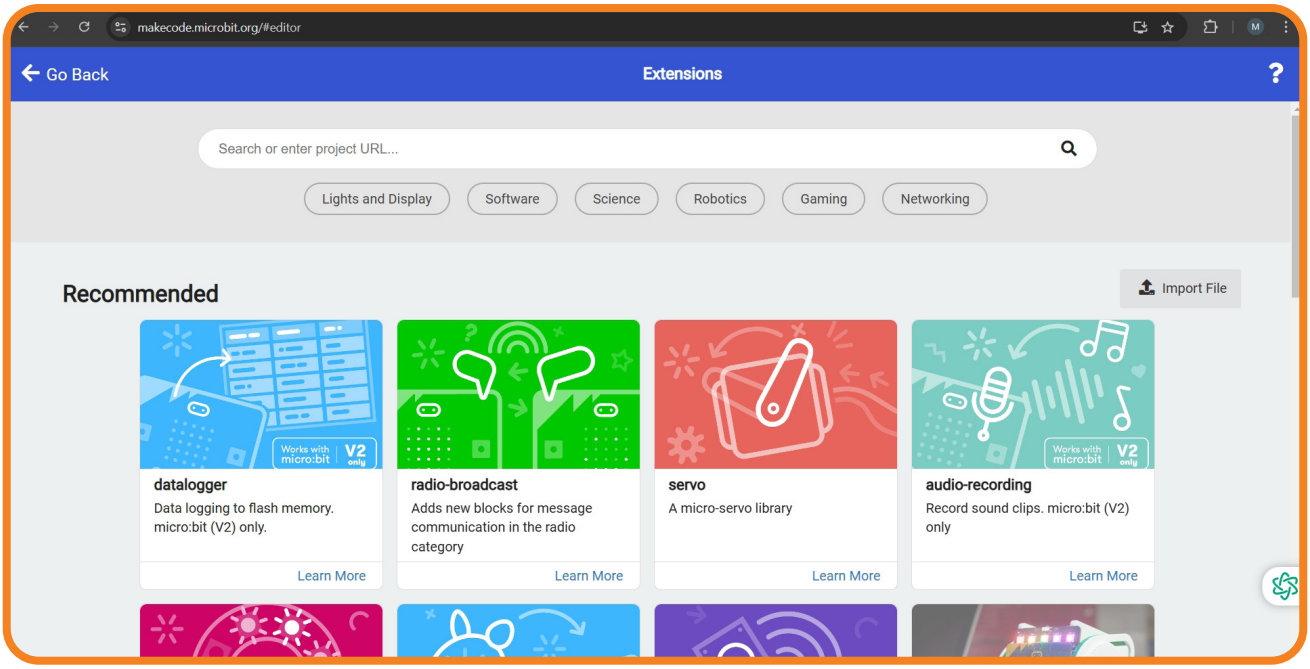
3: How do you display mathematical results on the micro:bit screen?

4: How do you connect a calculation block and a display block?

5: Why do you put the calculation blocks in a "forever" block?

Section 10:

Programming a Robot

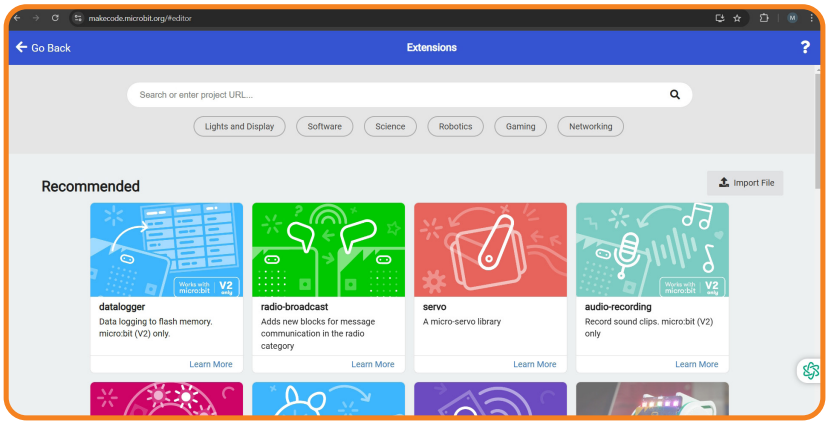


Section 10: Programming a robot

Outcomes

- Understanding Extensions
- Using Specific Extensions
- Adding and Testing Extensions
- Programming Servo Motors
- Understanding Demo Code Functions

Extensions in MakeCode:



Introduction to Extensions:

Extensions in *MakeCode* are powerful tools that add extra functionality to your programming environment. They allow you to use special features, like controlling sensors, motors, or LEDs, which aren't part of the basic *MakeCode* library. Extensions are essentially blocks of code created by developers to work with specific hardware or functions.

For the JitterBit, we'll use three key extensions:

MakerBit Ultrasonic Sensor Extension (to control the ultrasonic sensor).

1. **Servos Extension** (to program the servo motors).

2. **NeoPixel Extension** (to control the onboard addressable LEDs).

What Are Extensions?

In simple terms, extensions are like add-ons for *MakeCode*. They expand the capabilities of the editor, giving you access to blocks tailored for specific components or actions.

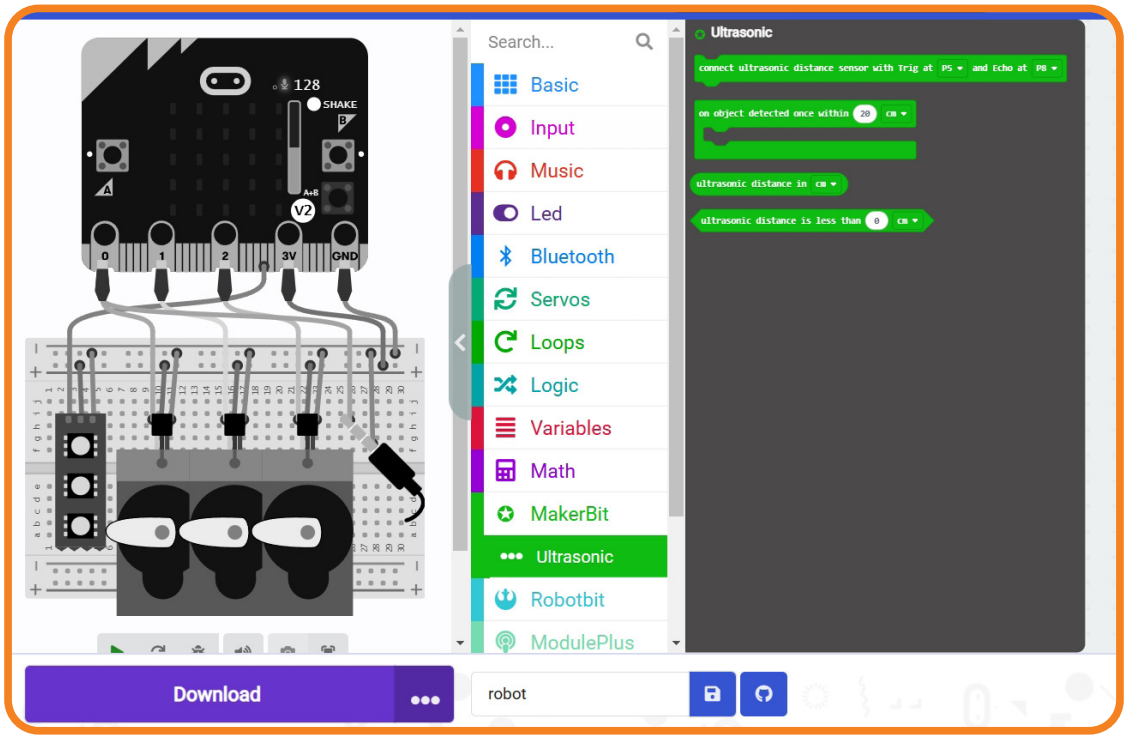
For example:

- The MakerBit Ultrasonic Sensor Extension provides blocks to measure distance with an ultrasonic sensor.
- The NeoPixel Extension allows you to control LEDs with colorful light effects.

Why Do We Use Extensions?

- **Component-Specific Functionality:** Extensions let you control hardware that isn't natively supported by *MakeCode*.
- **Ease of Use:** Instead of writing complex code, extensions provide ready-to-use blocks.
- **Customization:** You can mix and match extensions to suit your project's needs.

Extensions We'll Use for the JitterBit:



1. MakerBit Ultrasonic Sensor Extension:

What It Does:

This extension helps the JitterBit use its special sensor to measure how far away things are.

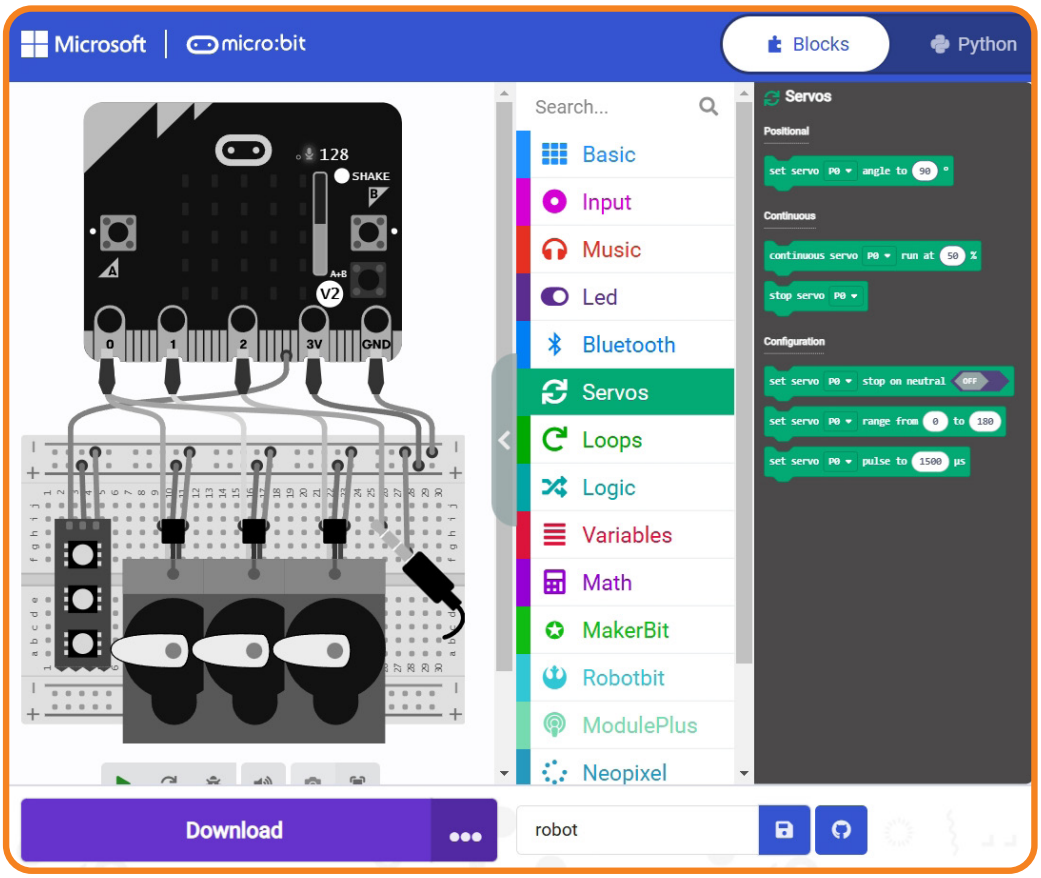
It gives you blocks for:

- Finding out the distance in centimeters or inches.
- Seeing if something is within a certain distance.

Key Blocks from the MakerBit Extension:

- **Measure Distance in cm**: This block tells you how far away something is in centimeters.
- **When Object Detected**: This block lets you run your code when something comes close to the sensor.

2. Servos Extension:



What It Does:

This extension helps you control the servo motors, which make the JitterBit move.

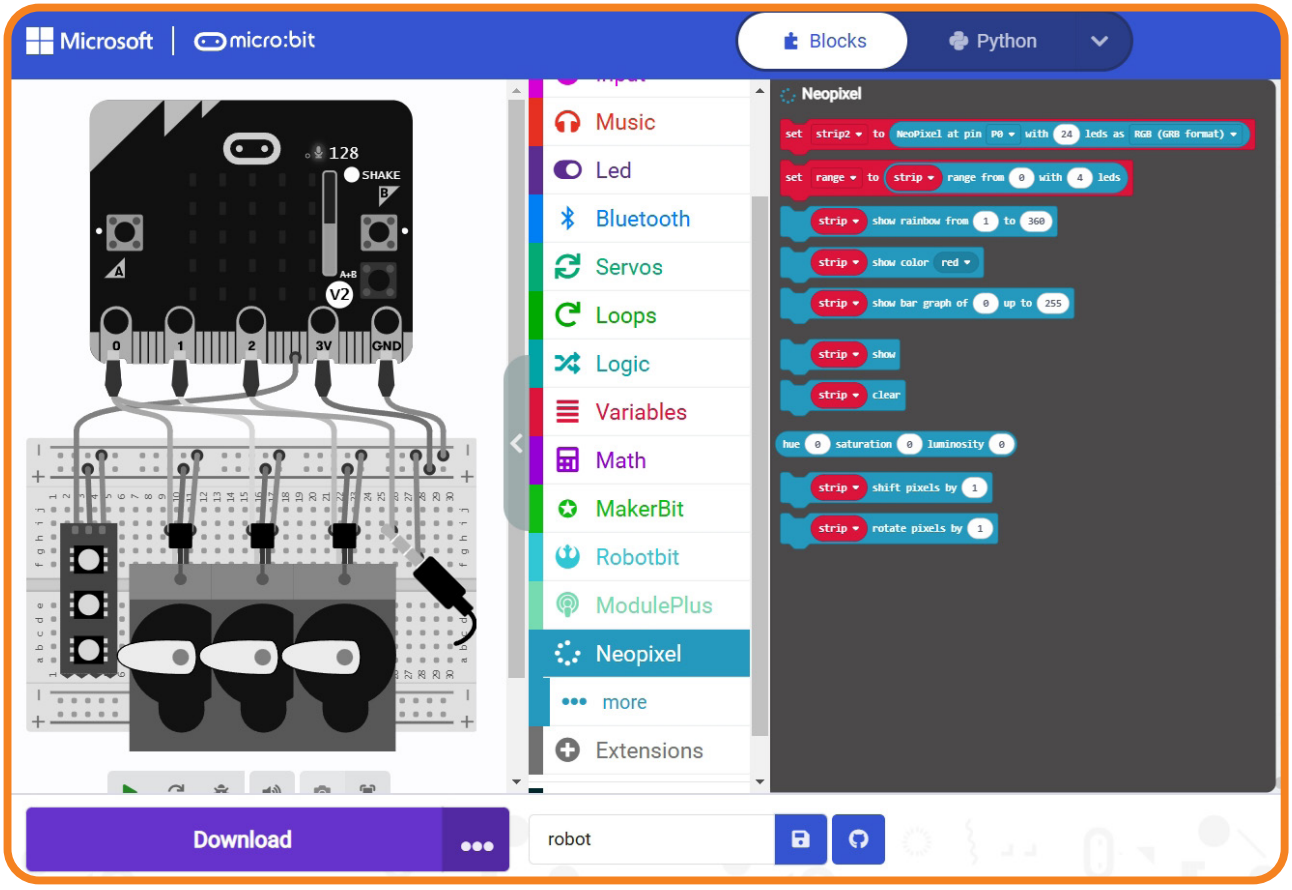
It gives you blocks for:

- Setting the servo motor to different angles (like 0 degrees, 90 degrees, or 180 degrees)
- Controlling more than one servo at the same time.

Key Blocks from the Servos Extension:

- **Write Servo on Pin P0 to (Angle)**: Sets the servo motor connected to pin P0 to a specific angle.
- **Stop Servo on Pin P0**: Stops the servo motor connected to pin P0.

3. NeoPixel Extension:



What It Does:

This extension controls the colorful lights (RGB LEDs) on the JitterBit.

It lets you:

- Show bright colors.
- Create fun animations like rainbows or color fades.
- Control single lights or groups of lights.

Key Blocks from the NeoPixel Extension:

- **Show Color (Red)**: Lights up all LEDs with one color.
- **Rotate Pixels**: Moves the LED colors around.
- **Set Pixel Color**: Changes the color of one specific LED.

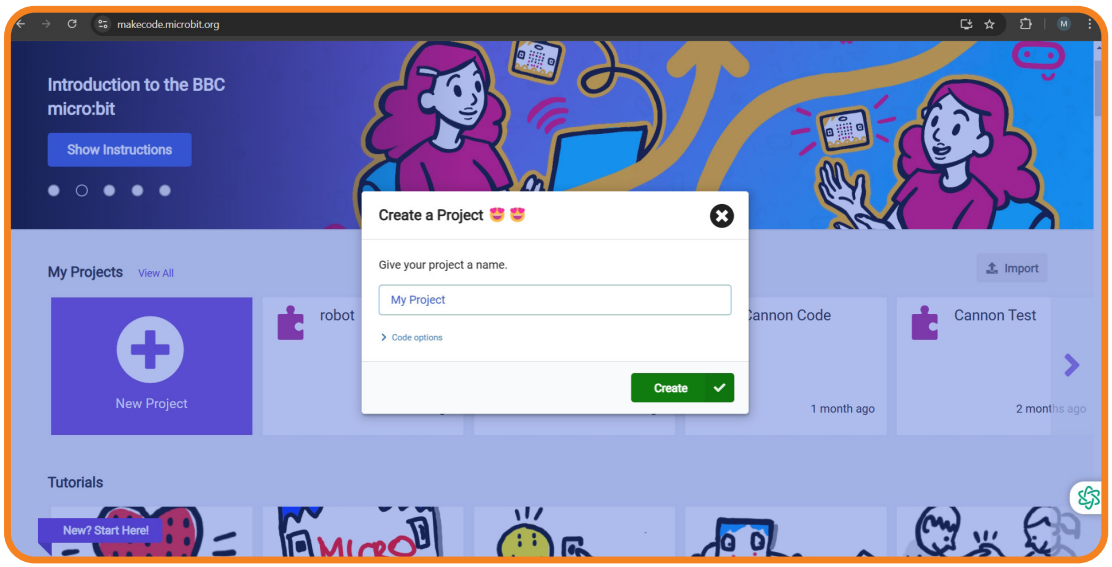
4. Adding Extensions in MakeCode:

It's quick and easy to add extensions in *MakeCode*. Just follow these steps to add the extensions you need for the JitterBit:

Step-by-Step Guide to Adding Extensions:

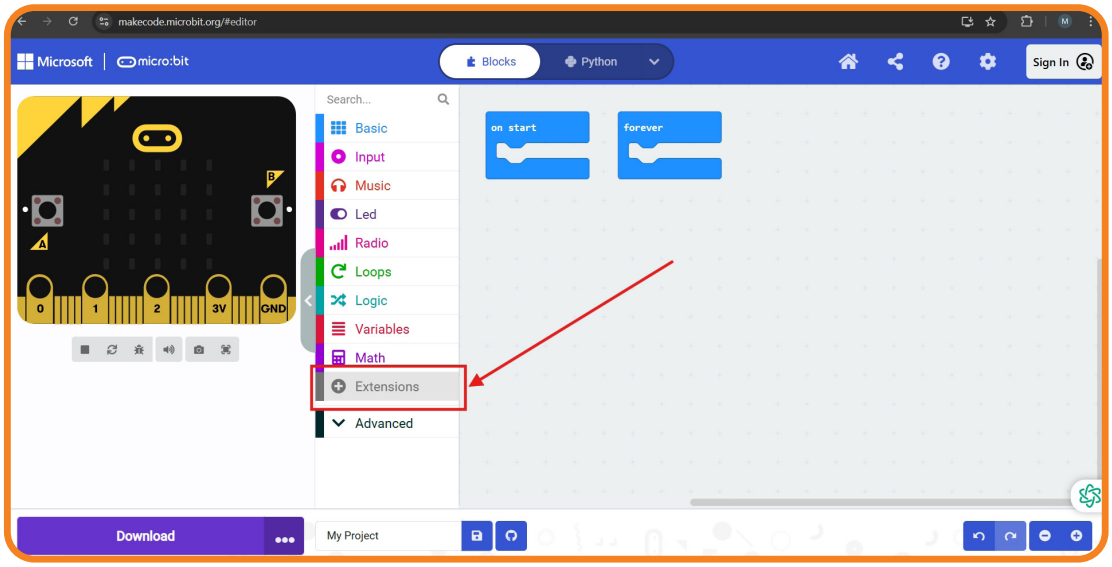
Step 1: Open the MakeCode Editor:

1. Go to ***https://makecode.microbit.org*** in your web browser.
2. Create a new project or open an existing one.



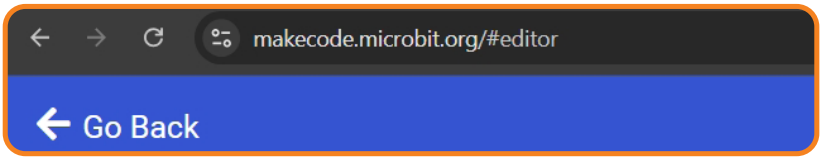
Step 2: Open the Extensions Menu:

Click on the *Extensions* Tab.



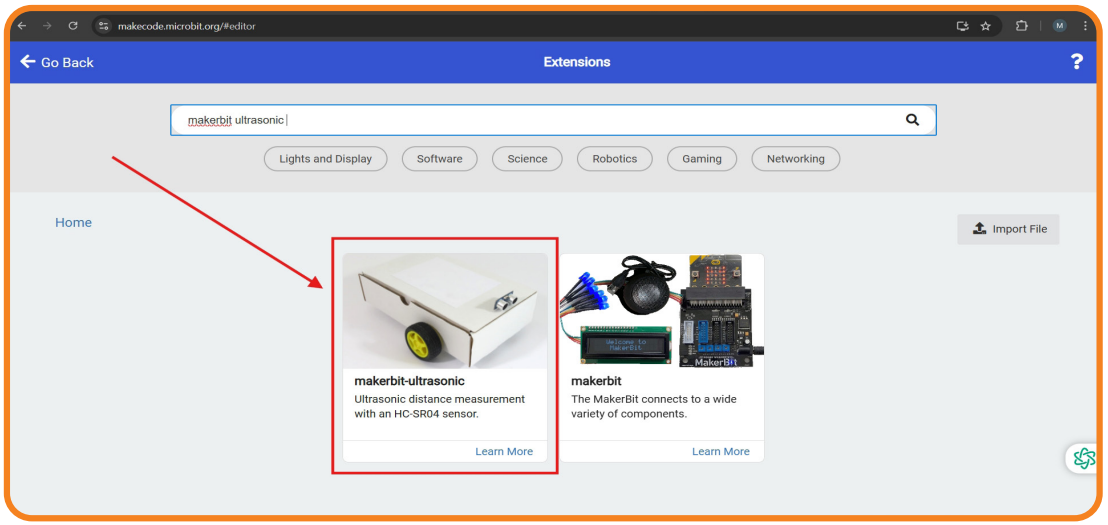
Step 3: Search for an Extension:

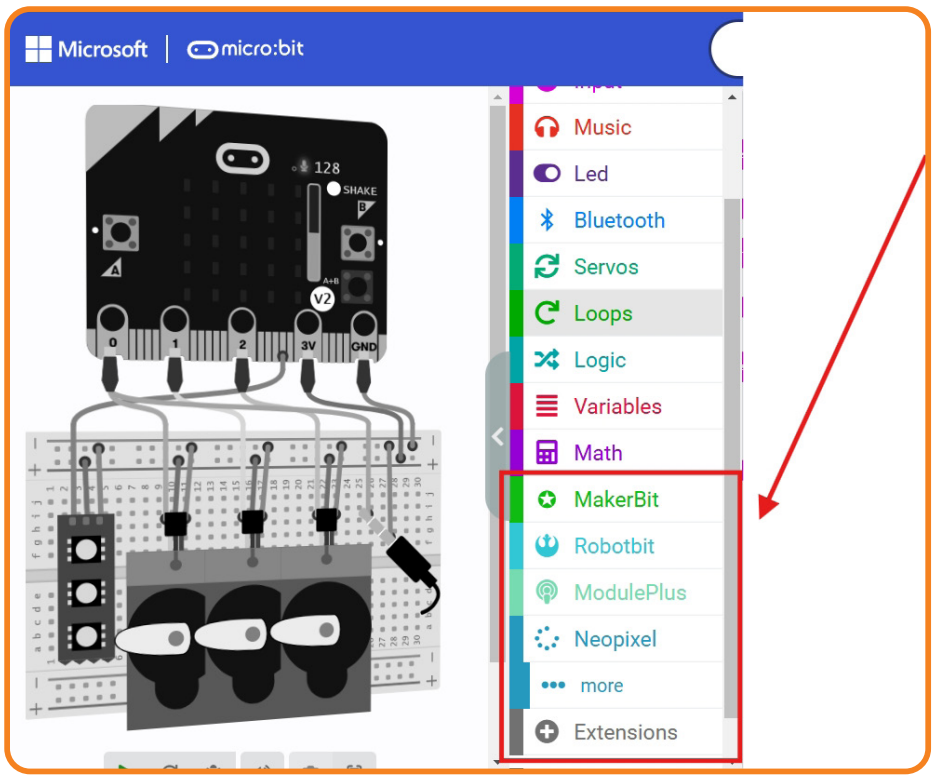
1. In the search bar, type the name of the extension you need (e.g., "MakerBit Ultrasonic").
2. Press Enter to search.



Step 4: Add the Extension:

1. Click on the correct extension from the search results.
2. Wait for MakeCode to load the new blocks into your workspace.

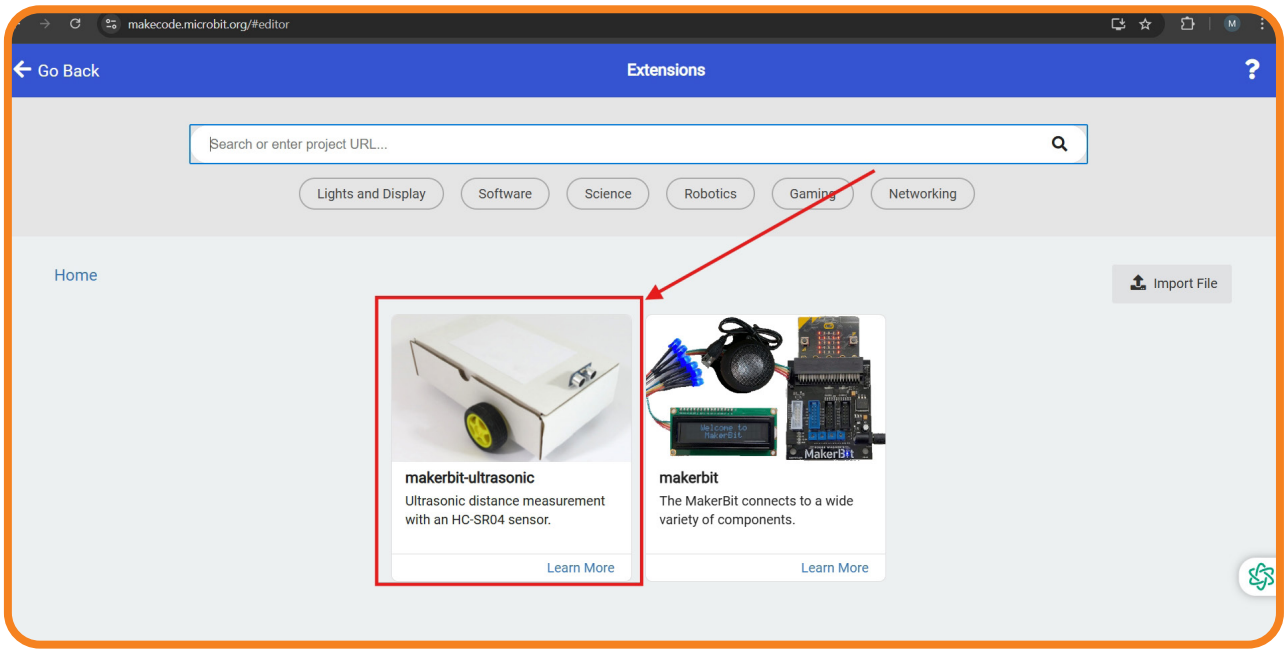




TIP: You'll now see new categories appear in the block palette, such as *MakerBit, Servos,* or *NeoPixel.*

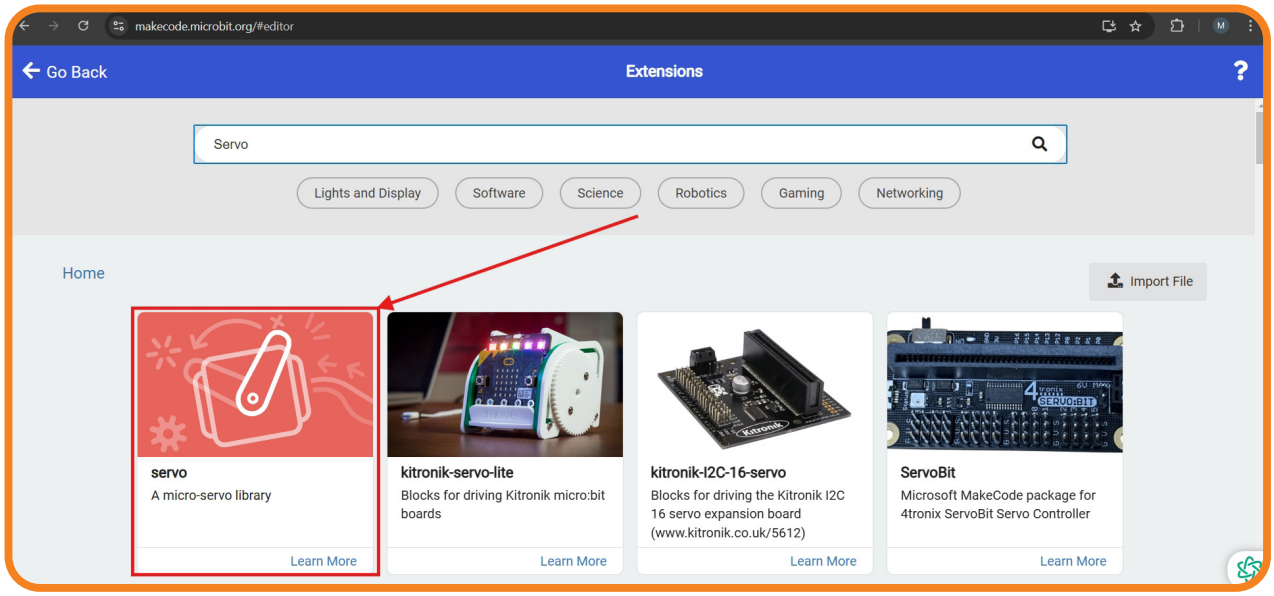
5. Adding the Specific Extensions for JitterBit:

MakerBit Ultrasonic Sensor Extension.



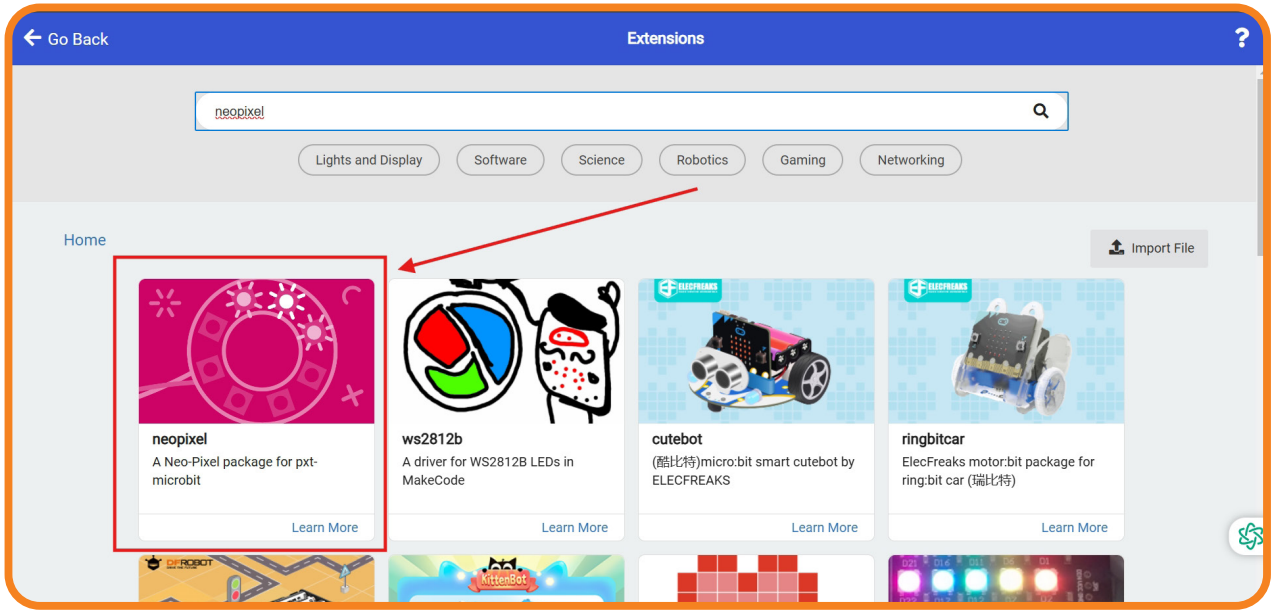
- Search for "*MakerBit*" in the *Extensions* menu.
- Select the *MakerBit* package to add blocks for ultrasonic sensors.

Servo Extension:



- Search for "*Servo*" in the *Extensions* menu.
- Add the package to allow *servo motor control* blocks.

Neopixel Extension



- Search for "*NeoPixel*" in the *Extensions* menu.
- Add the package to control the onboard LEDs.

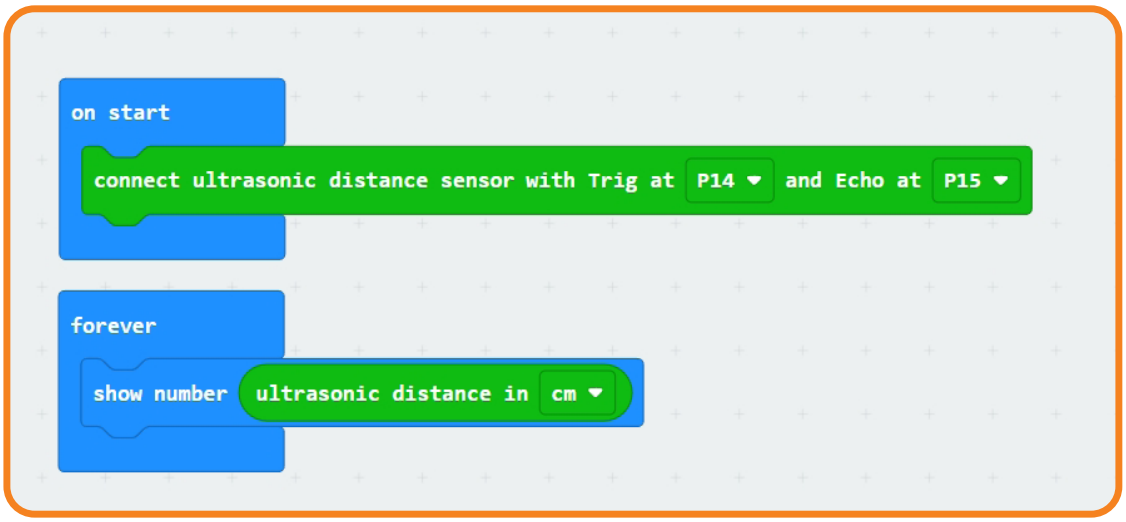
6. Testing the Extensions:

Once you've added the extensions, it's a good idea to test them to ensure they're working correctly.

Test 1: MakerBit Ultrasonic Sensor:

1. Use the *MakerBit* measure distance in cm block to display the distance on the micro:bit's LED matrix.
2. Place an object in front of the sensor and move it closer or farther away.
3. Confirm that the distance updates correctly.

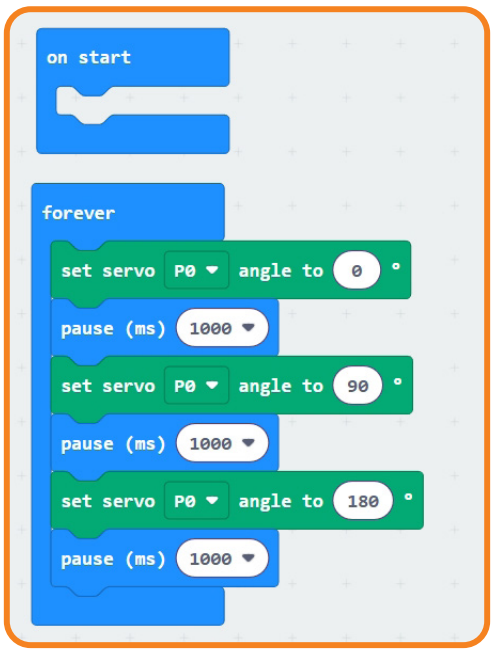
Example Block Code:



Test 2: Servos:

1. Connect a servo motor to pin P0.
2. Use the servo write pin P0 to (angle) block to move the servo to different angles (e.g., 0°, 90°, 180°).
3. Observe the servo's movement.

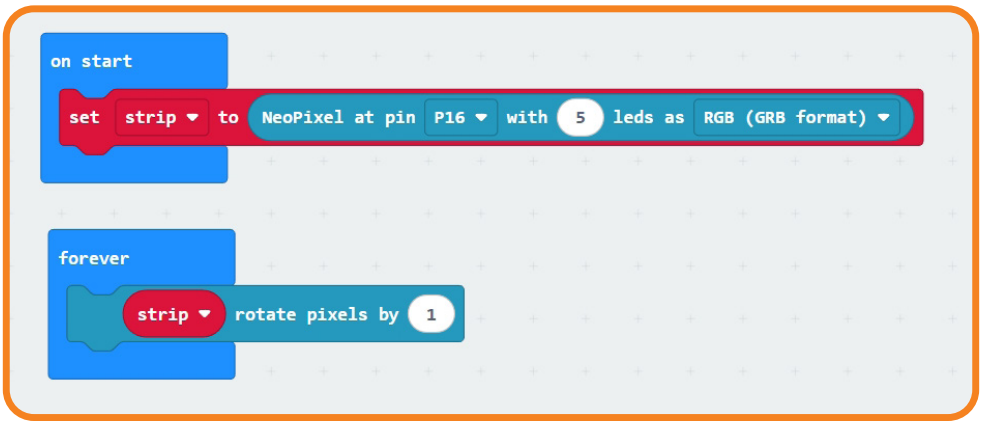
Example Block Code:



Test 3: NeoPixels:

1. Use the *strip show color (red)* block to light up all LEDs in red.
2. Add a *strip rotate pixels by 1* block inside a loop to create an animation.

Example Block Code:



Tips for Managing Extensions:

3. **Keep Extensions Updated:**
   - Check for updates in the *Extensions* menu to ensure you're using the latest version.
4. **Avoid Conflicts:**
   - Only add extensions you need for your project to avoid clutter or compatibility issues.
5. **Remove Unused Extensions:**
   - If an extension is no longer needed, you can remove it by deleting its associated blocks from your project.

What You've Learned:

- What extensions are and why they're important in *MakeCode*.
- The specific extensions needed for the JitterBit: *MakerBit Ultrasonic*, *Servos*, and *NeoPixel LEDs*.
- How to add and test new extensions in *MakeCode*.

7. Programming Servo Motors in MakeCode:

Controlling servo motors in *MakeCode* is simple and fun. The platform provides blocks that let you specify the pin and angle for each motor.

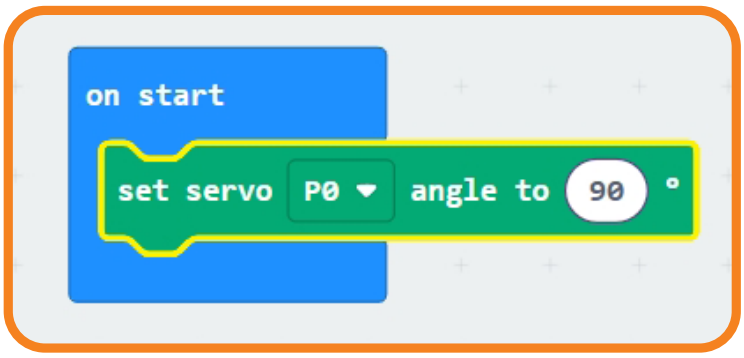
Basic Servo Control Block:

In *MakeCode*, you'll use the servo write block to control your servos.

Block Overview:

- **servo write pin (P0) to (angle)**
  - This block tells the servo connected to pin P0 to move to the specified angle.

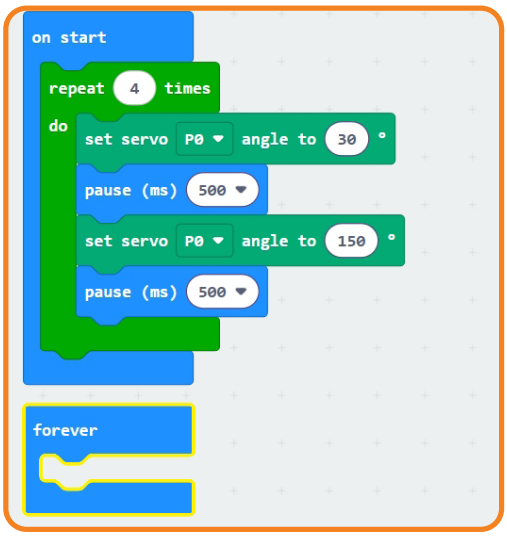
Example 1: Move a Servo to 90°:



Creating Movements with Servos:

You can use loops and pause blocks to create movements.

Example: Waving Motion:



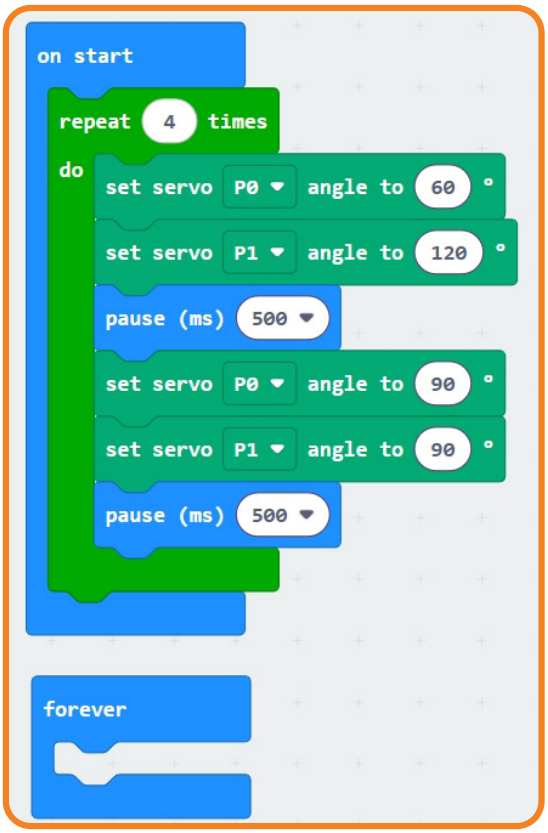
This program makes the servo connected to P0 wave back and forth five times.

8. Advanced Movements:

Working Multiple Servos together:

You can control multiple servos together to perform complex actions.

Example: Dance Moves:



9. Troubleshooting Servo Motors:

Even the best programmers encounter issues!

Here are some common problems and how to fix them:

Problem 1: Servo Isn't Moving

- **Check the wiring:** Ensure the *GND*, *Power*, and *Signal* wires are properly connected.
- **Verify the code:** Double-check the pin assignment in your program.
- **Test the power supply:** If you're using multiple servos, the micro:bit's power might not be enough. Use an external power source if needed.

Problem 2: Servo Doesn't Hold Position

- **Calibration needed:** Reset the servo to 90° using the Calibration Function by holding down button *A + B* Simultaneously.
- **Power fluctuations:** Ensure the power supply is stable.

Real-World Applications of Servo Motors:

Now that you know how servo motors work, imagine where else they're used!

- **In Prosthetics:** Servo motors help move robotic limbs.
- **In Animatronics:** Used to create lifelike movements in movies or theme parks.
- **In Drones:** Control propellers for stable flight.
- **In Robots Like JitterBit:** Make them dance, walk, or jump!

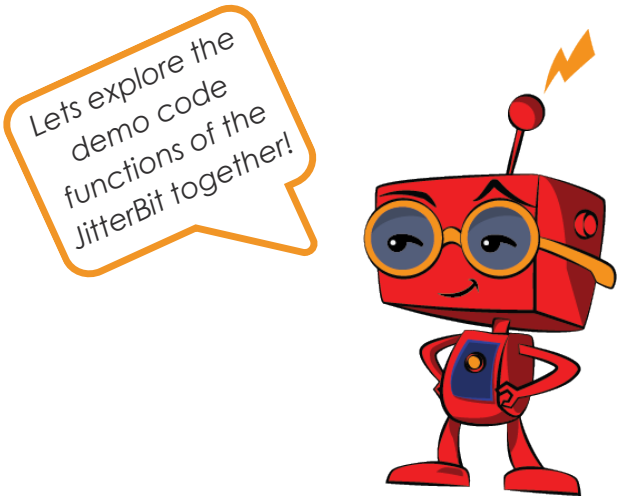
Challenge: Think of a real-world problem where servo motors could help. Could you design a robot arm or a waving hand?

10. Demo Code Functions of the JitterBit

The Demo code for the JitterBit, that you downloaded have many functions so you can start to have fun right away.

These functions are like built-in "mini-programs" that make the robot respond to specific inputs, such as button presses, sound detection, or touch.

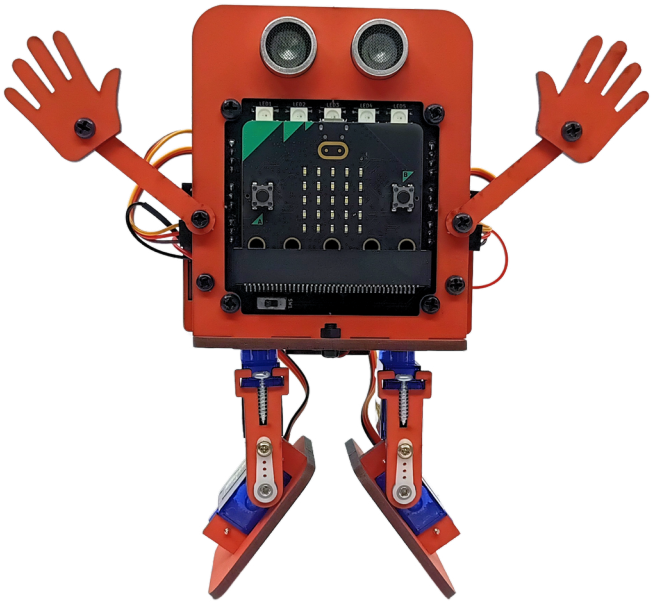
Each function showcases the robot's ability to interact with its environment and perform unique actions.



10.1 The Scare Function:

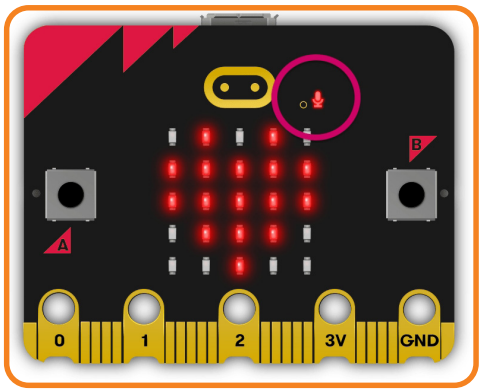
Overview:

The **Scare Function** is one of the JitterBit's most entertaining features. When a loud noise is detected—like a clap, snap, or shout—the JitterBit reacts by lifting its legs or arms, and letting out a scream!



How It Works:

1. **Input Detection:**
    - The micro:bit is equipped with a built-in microphone, located near the LED matrix as can be seen in the image below:



    - When the microphone detects a sound above a certain loud noise, it triggers the Scare Function.

2. **Robot Reaction:**
    - The servo motors quickly move the robot's legs into a "startled" position.
    - The micro:bit plays a screaming sound using its speaker.
    - The LED matrix displays a startled face, such as wide-open eyes or an animated flash.

What Happens in the Code?

In block-based programming, the Scare Function uses several key components:

- Input Blocks to monitor the sound level from the microphone.
- Conditional Statements to check if the sound is loud enough to trigger the scare.
- Servo Control Blocks to move the robot's legs into a startled pose.
- Sound Blocks to play a scream.
- LED Display Blocks to show the startled expression.

Example Block Code for Scare Function:

How to Trigger the Scare Function:

1. **Make a loud sound, such as:**
   - Clapping your hands.
   - Snapping your fingers.
   - Shouting near the robot.
2. Watch the JitterBit respond with its dramatic startled pose and scream.

What Can You Learn from the Scare Function?

- **How sound sensors work**: Learn how the micro:bit detects changes in sound levels.
- **Real-world applications**: Sound sensors are used in security systems and voice-activated devices.
- **Programming logic**: The Scare Function teaches conditional logic (e.g., "If sound is loud, then scare").

10.2 Button A - The Jitter Function

Overview:

Pressing Button A on the micro:bit activates the Jitter Function, where the robot shakes rapidly back and forth, creating a playful and energetic motion.

How It Works:

1. **Button Detection:**
   - The micro:bit detects when Button A is pressed.
   - This input triggers the Jitter Function.

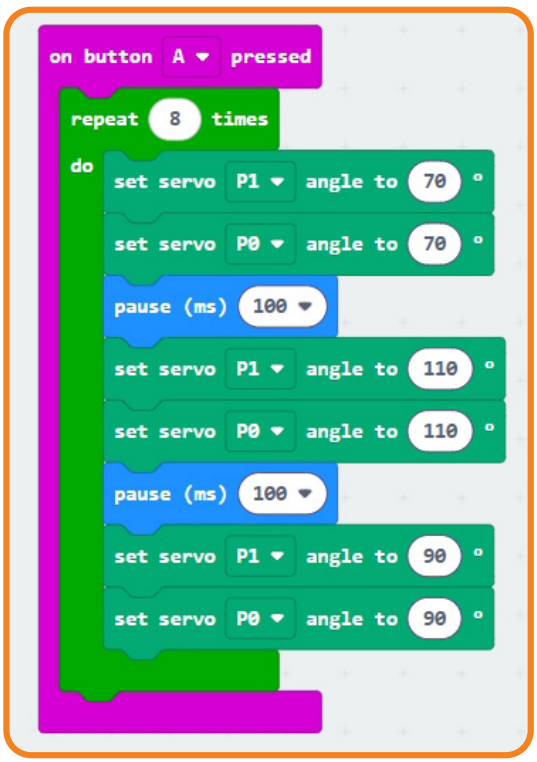
2. **Robot Reaction:**
   - The servo motors move back and forth in quick, small increments, creating a jittery effect.

What Happens in the Code?

The Jitter Function uses:

- **Input Blocks** to detect when Button A is pressed.
- **Loop Blocks** to repeat the shaking motion several times.
- **Servo Control Blocks** to create small, rapid movements.

Example Block Code for Jitter Function:



How to Trigger the Jitter Function:

1. Press Button *A* on the micro:bit.
2. Watch the JitterBit shake in a jittery motion.

What Can You Learn from the Jitter Function?

- **Looping in programming**: Repeating a block of code to create continuous motion.
- **Precision control**: Adjusting servo angles to create desired movements.
- **Fun customizations**: Experiment with different angles or speeds to modify the jitter effect.

10.3 Button B - The Dance Function

Overview:

When you press Button *B*, the JitterBit starts dancing! This function allows the robot to perform a series of coordinated movements, which can include waving its legs, moving side to side, or even spinning in place.

How It Works:

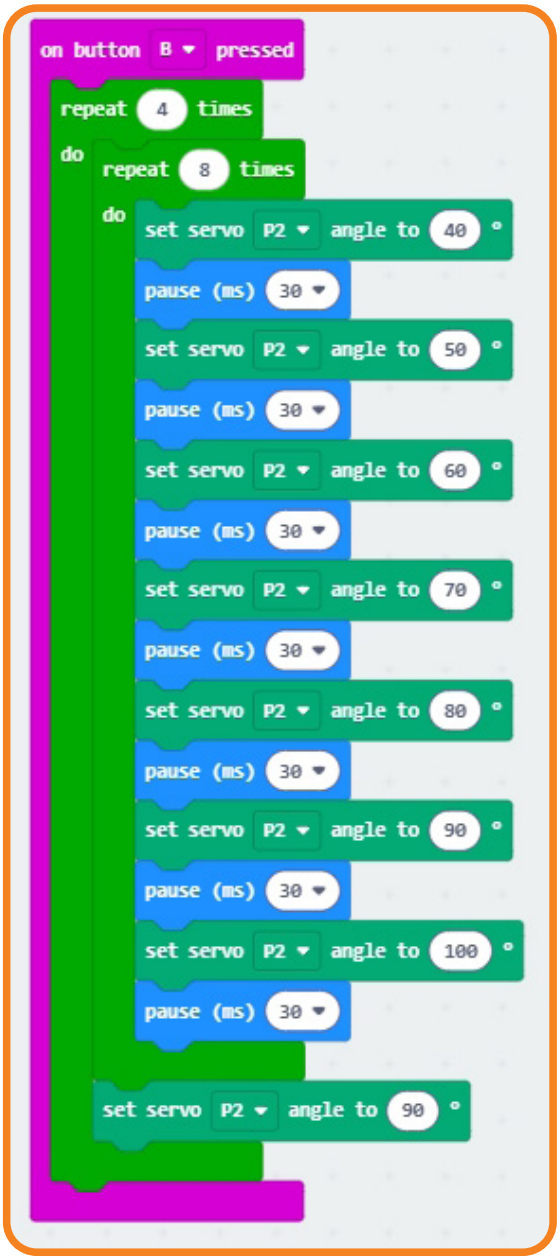
1. **Button Detection:**
   - The micro:bit detects when Button *B* is pressed.
   - This input triggers the Dance Function.
2. **Robot Reaction:**
   - The servos move in a pre-programmed sequence to simulate dancing.
   - You can add music or light effects to enhance the performance.

What Happens in the Code?

The Dance Function uses:

- *Input* Blocks to detect when Button *B* is pressed.
- *Servo Control* Blocks to move the legs in different directions.
- *Sound and Light* Blocks to add music and LED effects.

Example Block Code for Dance Function:



How to Trigger the Dance Function:

1. Press Button *B* on the micro:bit.
2. Watch the JitterBit perform its fun dance routine.

What Can You Learn from the Dance Function?

- **Sequencing in programming**: Organizing steps in a specific order to create a routine.
- **Combining features**: Learn how to synchronize movements, sounds, and lights.
- **Creativity**: Experiment with new dance moves or customize the sequence.

10.4 Buttons A + B - The Calibration Function

Overview:

Pressing both Button *A* and Button *B* activates the Calibration Function, which resets the servo motors to their neutral (90°) position.

This ensures that the robot stands upright and performs movements correctly.

It is important to calibrate your servo motors before assembling the JitterBit.

How It Works:

1. **Button Detection:**
   - The micro:bit detects when both buttons are pressed simultaneously.
   - This input triggers the Calibration Function.

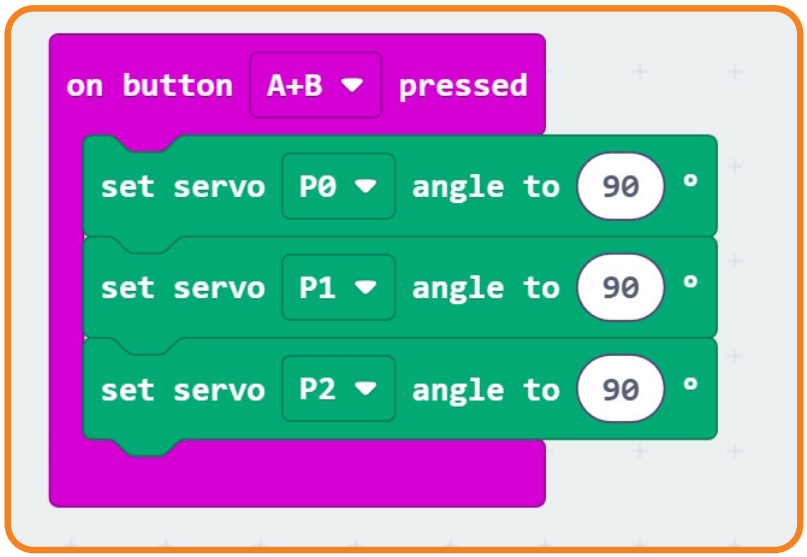
2. **Robot Reaction:**
   - The program moves all servos to 90°, aligning the legs and arms to their default positions.

What Happens in the Code?

The Calibration Function uses:

- *Input* Blocks to detect when both buttons are pressed.
- *Servo Control* Blocks to set all servo angles to 90°.

Example Block Code for Calibration Function:



How to Trigger the Calibration Function:

1. Press Button *A* and Button *B* together.
2. Watch the JitterBit reset its stance to the neutral position.

What Can You Learn from the Calibration Function?

- **Default settings**: How to reset components to their original state.
- **Preventing errors**: Calibration ensures movements are accurate and consistent.

10.5 The Tickle Function (Pressing the Logo)

Overview:

The Tickle Function is triggered by touching the micro:bit logo, making the JitterBit wiggle and giggle.

How It Works:

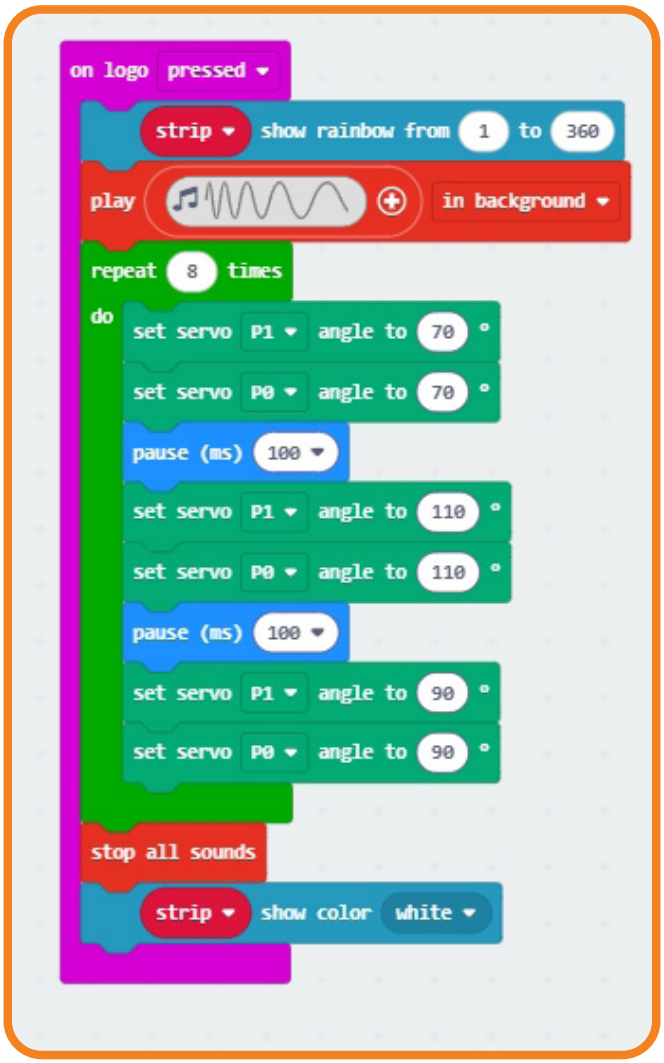
1. **Touch Detection:**
   - The micro:bit logo acts as a touch sensor.
   - When touched, the function is triggered.
2. **Robot Reaction:**
   - The servos wiggle in small, repetitive motions.
   - A giggling sound plays through the speaker.

What Happens in the Code?

The Tickle Function uses:

- *Touch Input* Blocks to detect logo presses.
- *Loop* Blocks to repeat the wiggling motion.
- *Sound* Blocks to play the giggling effect.

Example Block Code for Tickle Function:



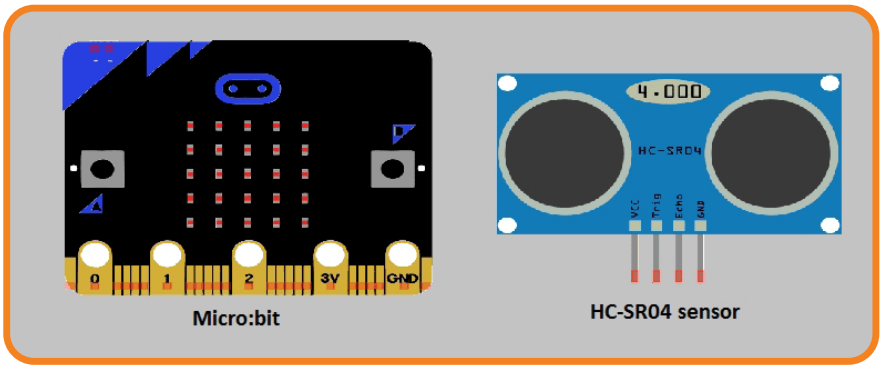
How to Trigger the Tickle Function:

1. Press the micro:bit logo gently.
2. Watch the JitterBit wiggle and hear it giggle.

What Can You Learn from the Tickle Function?

- **Touch sensitivity**: How touch sensors work in robotics.
- **Creative movement**: Customize the wiggle motion and sounds.

10.6 Ultrasonic Jump:



Objective:

In this activity, you will program the JitterBit to detect when someone walks by using its ultrasonic sensor and make it react by "jumping" (triggering the Scare Function).

This activity combines sensors and servo motor control to demonstrate how robots interact with their environment.

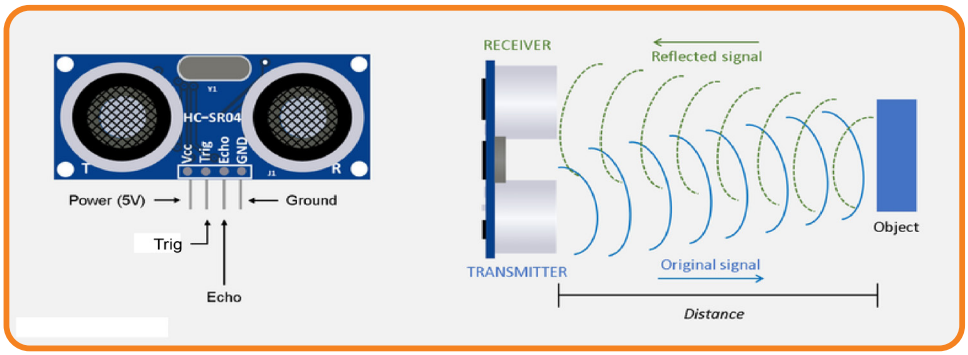
What You'll Learn:

- How ultrasonic sensors work and how they measure distance.
- How to program the JitterBit to perform actions based on sensor input.

What You Need:

- A fully assembled JitterBit with an ultrasonic sensor connected to the micro:bit.
- Access to a computer with the *MakeCode* editor.

How Does the Ultrasonic Sensor Work?



The ultrasonic sensor acts like the robot's eyes. It sends out high-frequency sound waves that bounce off objects and return to the sensor.

By measuring the time it takes for the sound to return, the sensor calculates the distance to the object.

Key Components of the Ultrasonic Sensor:

1. **Trigger Pin**: Sends the sound waves.
2. **Echo Pin**: Receives the reflected sound waves.

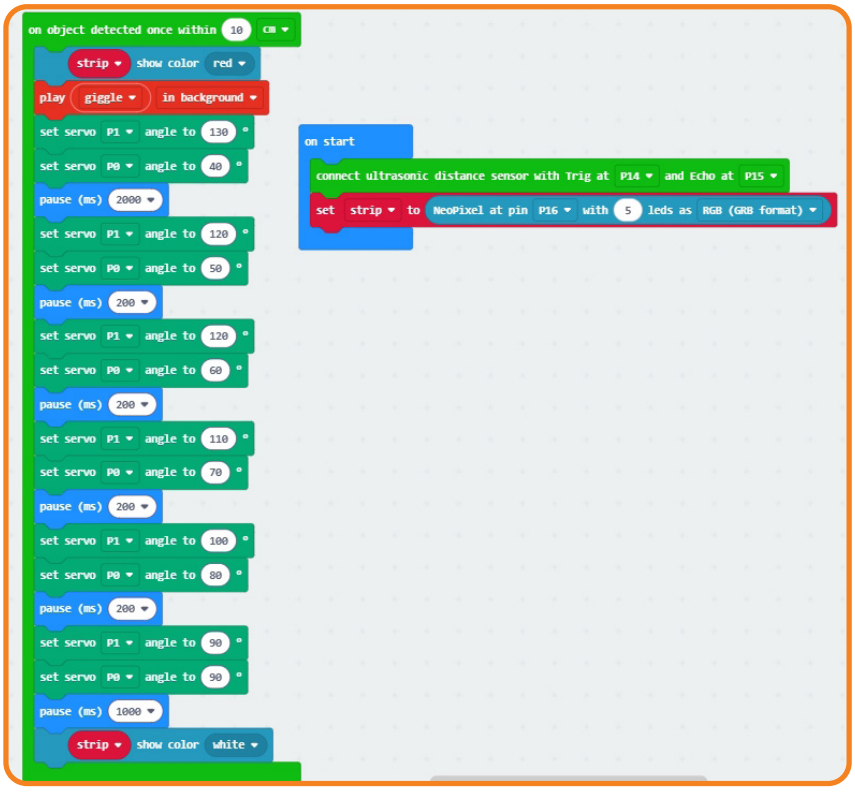
Connecting the Ultrasonic Sensor:

- Trigger Pin → P14 on the micro:bit.
- Echo Pin → P15 on the micro:bit.

Programming the Scare Behavior with Ultrasonic Sensor:

Here's how to program the JitterBit so it reacts to the ultrasonic sensor by "jumping" when an object gets close.

Complete Block Code Example:



Explanation of the Code:

1. **Initialization (on start):**
   - The *on start* block sets up the ultrasonic sensor (connected to pins P14 and P15) and resets the servos to their neutral position (90°).

2. **Continuous Monitoring (on object detected once within):**
   - The on object detected function continuously measures the distance in front of the robot.

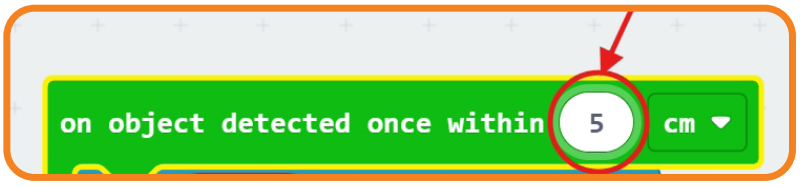
3. **Triggering the Scare Behavior:**
- If the distance is less than 10 cm, the robot:
   - Moves its legs to a "startled" position.
   - Plays a screaming sound.
   - Pauses for 1 second to complete the scare action.

4. **Resetting to Neutral Position:**
   - After completing the scare, the servos return to their neutral position (90°).

5. **Idle State:**
   - If no object is detected within 10 cm, the robot stays in its neutral position.

Customizations:

6. **Adjust the Distance Threshold:**
   - Change on object detected within 10cm to a higher or lower value depending on how sensitive you want the scare reaction to be.



7. **Change the Servo Angles:**
   - Experiment with different angles to modify the scare pose.

8. **Add LED Effects:**
   - Use the NeoPixel extension to flash red lights during the scare.

9. **Add a Different Sound:**
   - Replace the scream with another sound effect, like laughter or a melody.

Testing and Debugging:

1. **Sensor Doesn't Detect Objects:**
   - Ensure the ultrasonic sensor is properly connected (Trigger → P14, Echo → P15).
   - Test with a reflective object for better detection.

2. **Servo Movements Are Incorrect:**
   - Verify the servo connections (e.g., left leg on P0, right leg on P1).
   - Check the angles to ensure noticeable movements.

3. **Sound or Lights Don't Work:**
   - Make sure the sound and NeoPixel blocks are placed correctly within the if block.

Experiment and Customize:

4. **Step 1: Adjust the Threshold**
   - Change the detection distance from 10 cm to another value, like 5 cm or 20 cm.
   - Observe how the robot's behavior changes.

5. **Step 2: Add LED Effects**
   - Make the robot's LED matrix display a startled face when it jumps.

6. **Step 3: Add Lights**
   - Use *NeoPixel* blocks to flash colorful lights when the robot jumps.

What You've Learned:

- How to use an ultrasonic sensor to detect distance.
- How to program the JitterBit to react to its environment.
- How to create and use custom functions in *MakeCode*.

10.7 Dance Moves:

Objective:

In this activity, you'll program your JitterBit to show off its dance moves. You'll learn how to create a custom dance routine and add your own unique touches by controlling the robot's servo motors, adding sound effects, and using colorful lights.

What You'll Learn:

- How to program coordinated servo motor movements.
- How to synchronize lights and sounds with robot actions.
- How to use loops and sequences to create smooth, repetitive movements.

What You Need:

- A fully assembled JitterBit.
- Access to a computer with the *MakeCode* editor.

Step 1: Plan the Dance Routine:

Before programming, think about what movements you want your JitterBit to perform.

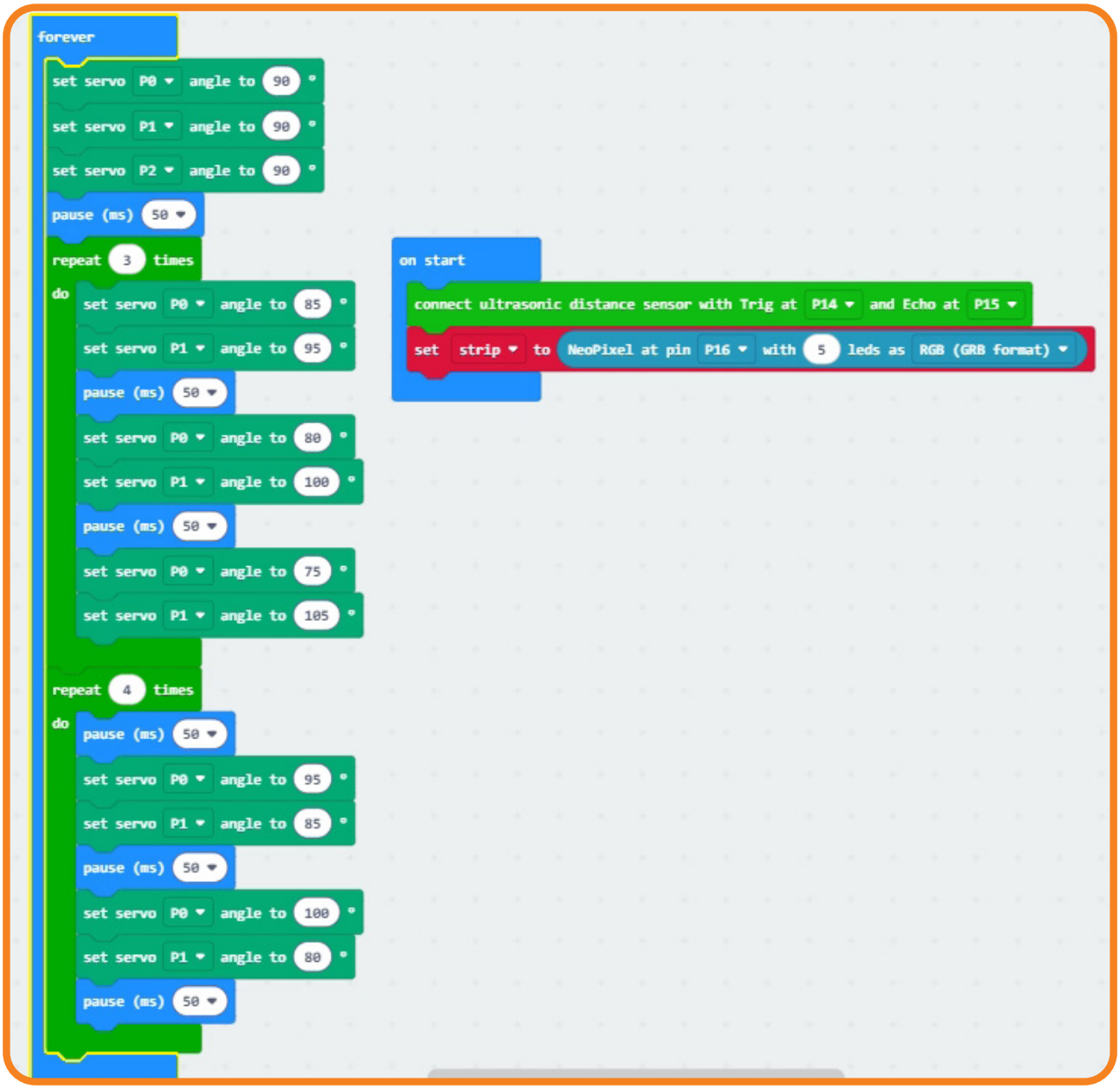
A simple dance routine might include:

- Moving its legs up and down.
- Wiggling side to side.
- Flashing lights in a rainbow pattern.
- Playing music or sound effects.

Step 2: Create the Dance Function:

Let's program a basic dance routine by moving the servos in a coordinated sequence.

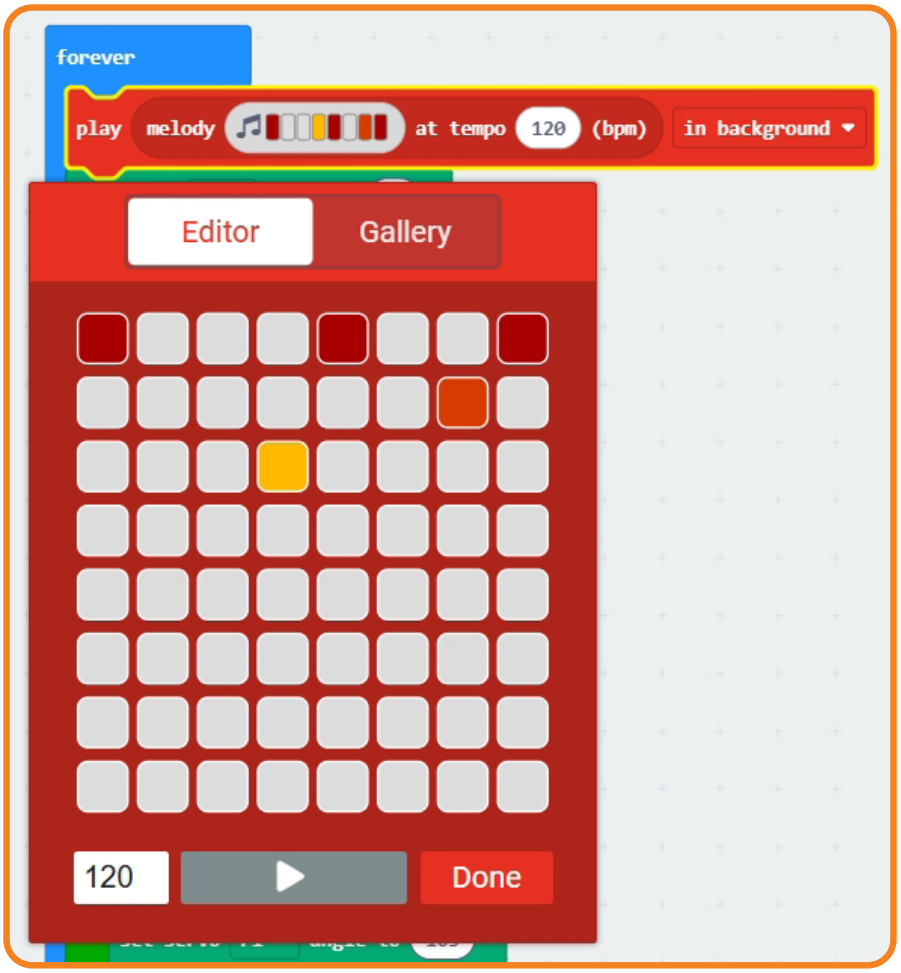
Block Code for the Dance Function:



Step 3: Add Music:

Make the dance routine more exciting by playing a melody during the movements.

Block Code to Add Music:

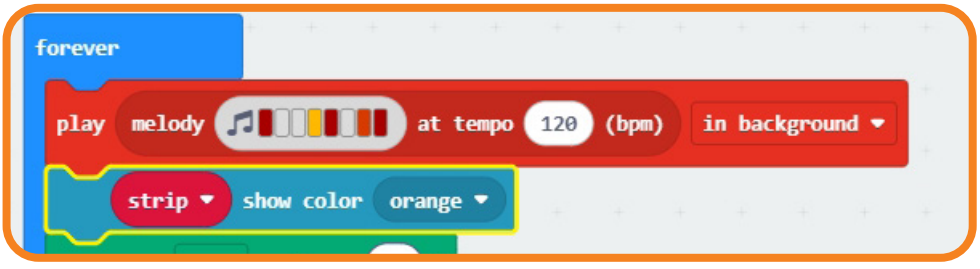


Add this block before or inside the dance() function to synchronize the music with the movements.

Step 4: Add NeoPixel Light Effects:

The NeoPixel lights can flash in sync with the dance. Create a colorful display using the rainbow or color-changing blocks.

Block Code for Lights:



Step 5: Test the Routine:

1. **Upload the Code**: Connect your micro:bit and upload the program through *MakeCode*.
2. **Press Button B**: Watch the JitterBit perform its dance routine.

Experiment and Customize:

Step 1: Add More Movements:

Expand the routine by incorporating additional servo angles or movements.

For example:

- Make the robot "wiggle" by alternating small servo angles (e.g., 80°, 100°).
- Add a "spin" by adjusting all servos simultaneously.

Step 2: Change the Melody:

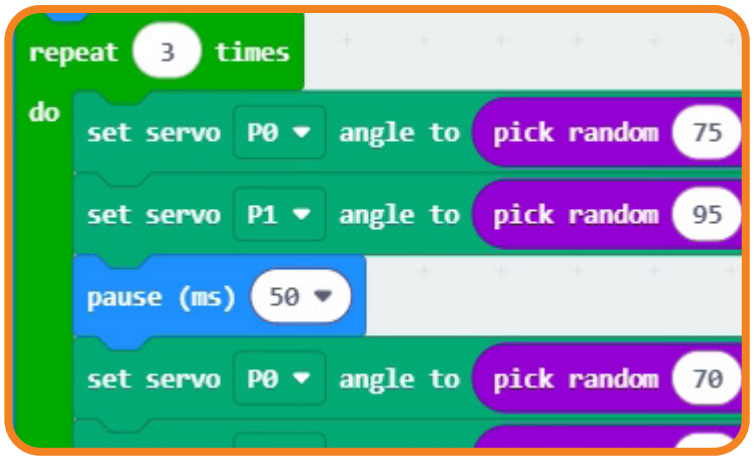
Replace the melody with your favorite song using *MakeCode's* music blocks.

Step 3: Add Randomness:

Make the dance unpredictable by introducing random angles or colors.

Step 4: Create a Dance-Off:

Pair two JitterBits together and program each to perform a different routine. Let them "compete" in a dance-off!



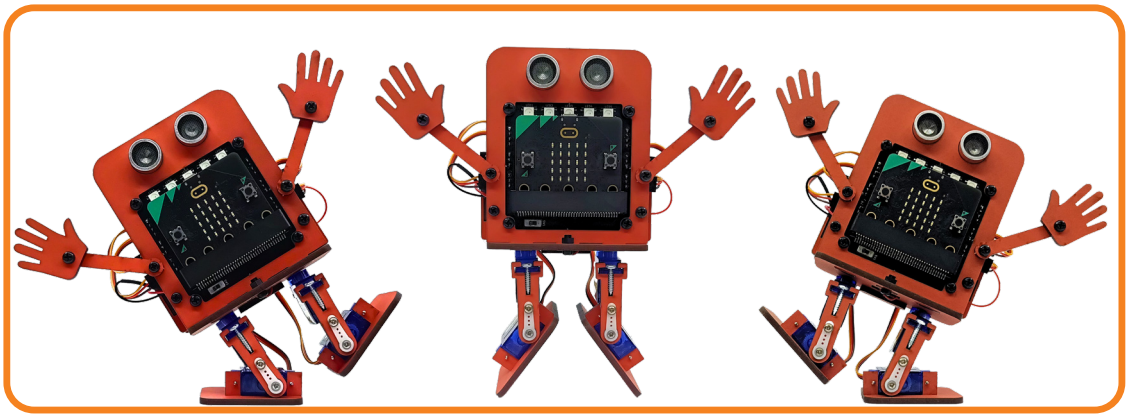
Troubleshooting Tips:

1. **Movements Look Jerky:**
   - Add small pauses between servo movements to smooth transitions.
2. **Lights Don't Sync:**
   - Double-check the placement of light blocks relative to the servo movements.
3. **Music Sounds Offbeat:**
   - Adjust the tempo or duration of pauses to better align with the routine.

What You've Learned:

- How to program servo motors for synchronized movements.
- How to combine lights, music, and movements to create a performance.
- How to use loops and sequences to build repetitive actions.

10.8 Make it shake:



Objective:

In this activity, you will program the JitterBit to respond to being shaken. When someone shakes the robot, it will perform a fun, wiggling motion combined with sounds or light effects. This activity will teach you how to use the shake gesture detection feature of the micro:bit.

What You'll Learn:

- How to use the shake gesture input from the micro:bit.
- How to program servos for quick, repetitive movements.
- How to enhance functionality by adding sounds and lights.

What You Need:

- A fully assembled JitterBit.
- Access to a computer with the *MakeCode* editor.

Step 1: Understand the Shake Gesture:

The micro:bit has a built-in accelerometer that can detect various movements, **including**:

- Tilting.
- Free-fall.
- Shaking.

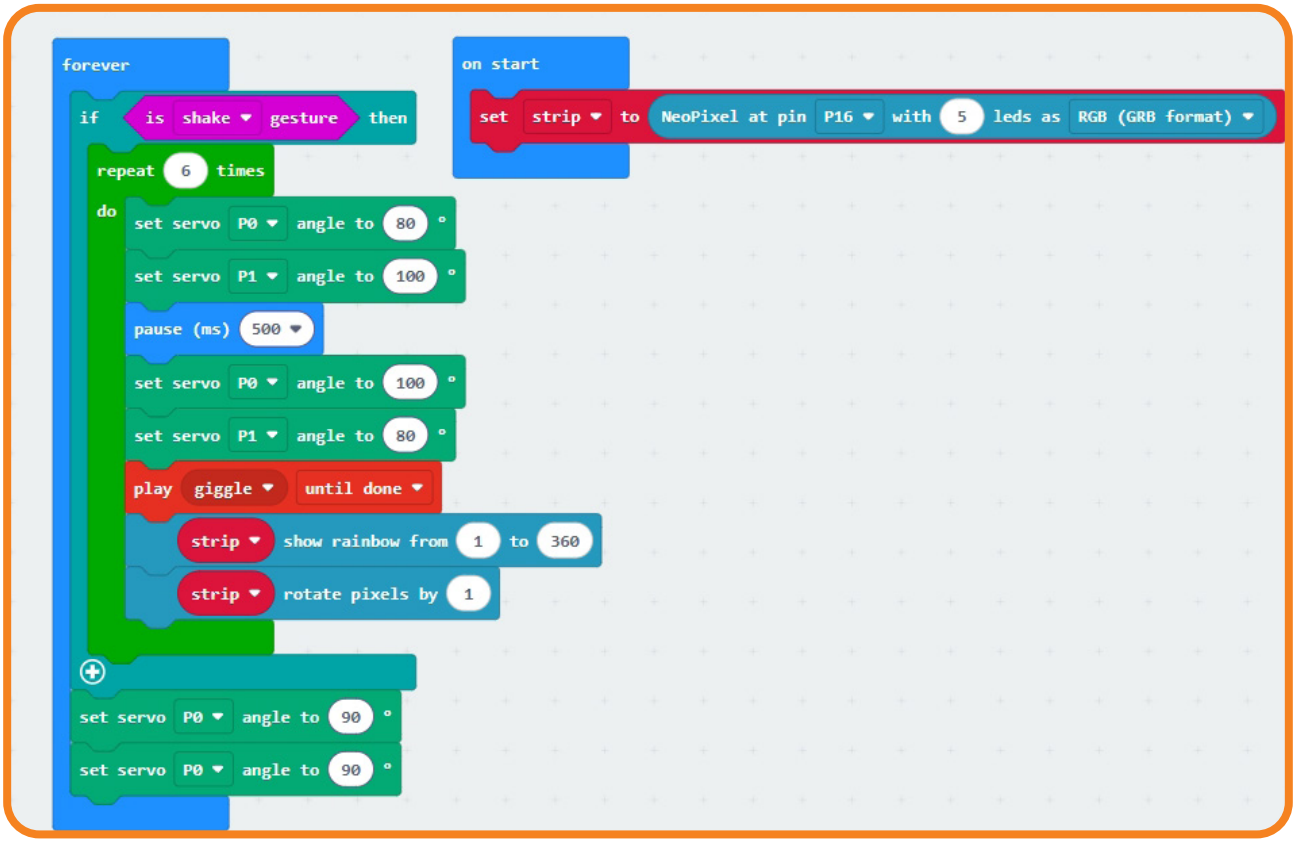
The shake gesture input triggers when the micro:bit is moved quickly back and forth.

You can use the on shake block in *MakeCode* to detect this movement.

Step 2: Create the Shake Function:

You'll program a shake() function that wiggles the servos back and forth quickly to simulate shaking. Add sound and LED effects to make it more exciting.

Block Code for the Shake Function:



Step 3: Test the Program:

1. **Upload the Code**: Connect your micro:bit to your computer and upload the program through *MakeCode*.
2. **Shake the JitterBit**: Gently shake the robot and watch it wiggle, giggle, and light up!

Experiment and Customize:

Step 1: Add More Wiggles:

- Increase the number of repetitions in the repeat block for a longer shake response.

Step 2: Change the Sound:

- Replace the giggle sound with another sound effect, like a laugh or a custom melody.

Step 3: Use Random Movements:

- Introduce randomness to the shake angles for a more unpredictable wiggle.

Step 4: Add Visual Effects:

- Use the LED matrix to display a playful face during the shake.

Step 5: Add a Stop Mechanism:

- To make the shake response stop after a certain time, add a stop condition using a pause block.

Troubleshooting Tips:

1. **Shake Gesture Isn't Detected:**
   - Ensure you're shaking the robot with enough force.
   - Double-check that the on shake block is included in the code.

2. **Robot Doesn't Wiggle Correctly:**
   - Verify the servo wiring and connections.
   - Check the servo angles in the shake() function for noticeable movement.

3. **Lights or Sounds Don't Sync:**
   - Adjust the placement of *sound and light* blocks relative to servo movements.

What You've Learned:

- How to use the micro:bit's accelerometer to detect gestures.
- How to program fun responses to physical interactions.
- How to combine movements, sounds, and lights to create engaging behaviors.

10.9 Day & Night Lamp:

Objective:

In this activity, you'll program the JitterBit to act as a Day & Night Lamp using its built-in light sensor.

The robot will:

- Glow brightly in the dark with warm colors (*Night Mode*).
- Dim its lights and display a "sunny" LED pattern during the day (*Day Mode*).

This activity will teach how to use the micro:bit's light sensor to create behaviors based on ambient light levels.

What You'll Learn:

- How to use the micro:bit's built-in light level detector.
- How to create conditional behaviors based on light intensity.
- How to program transitions between distinct states: *Day Mode and Night Mode*.

What You Need:

- A fully assembled JitterBit.
- Access to a computer with the *MakeCode* editor.

How Does Light Detection Work?

The micro:bit has a built-in light sensor embedded within its LED matrix. It can measure the surrounding light level, allowing the robot to react to changes in brightness.

Key Concepts for Light Detection:

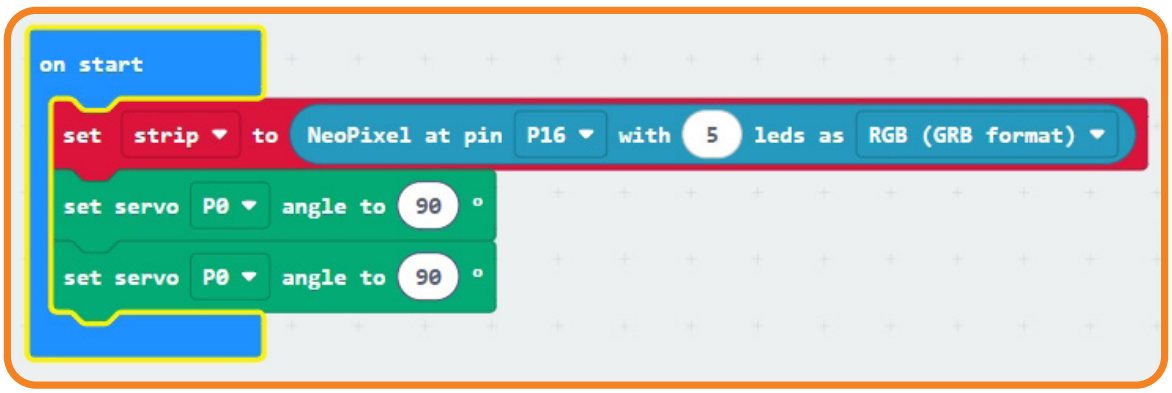
1. **Light Level:**
   - The sensor measures light intensity on a scale from 0 (dark) to 255 (bright).

2. **Thresholds:**
   - You can set a specific light level to determine whether it's "day" or "night."

Step 1: Initialize the Day & Night Lamp:

Block Code: Initialization:

Start by setting up the LEDs and servos for neutral states in both *Day and Night* Modes.

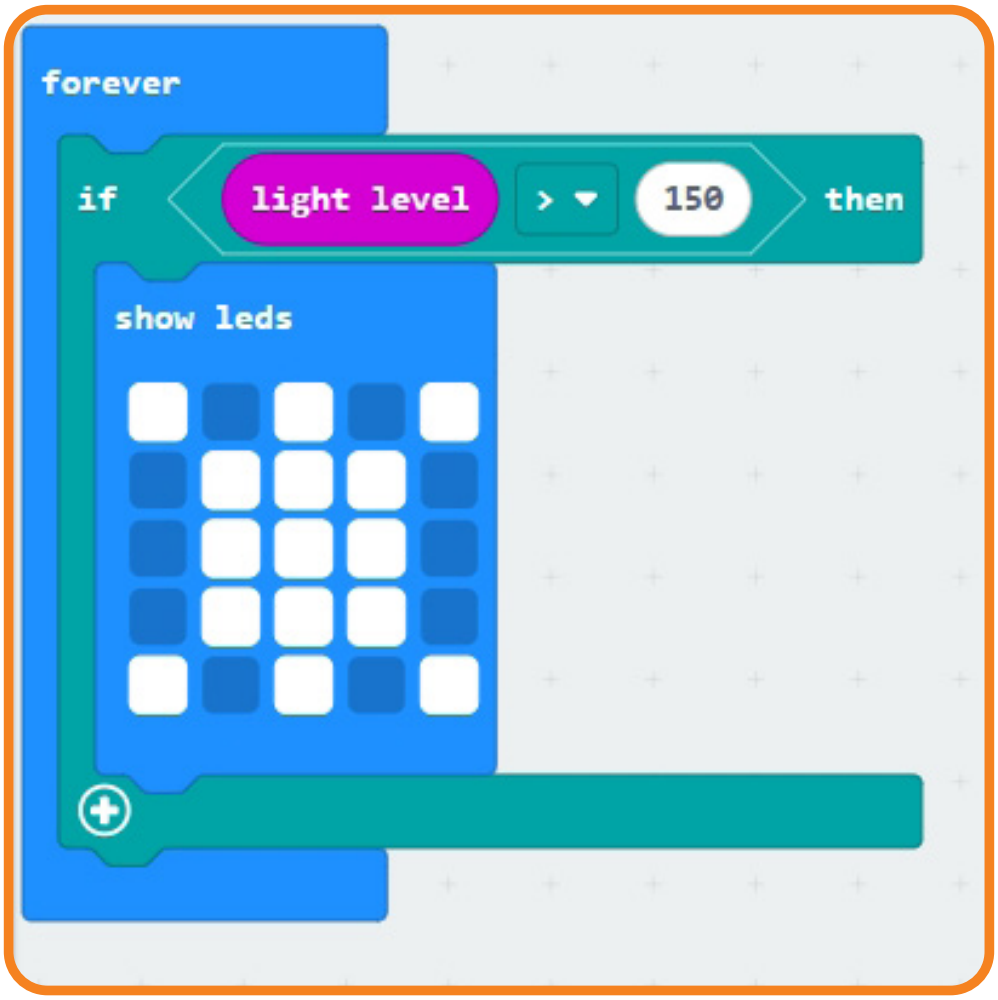


Step 2: Program Day Mode:

In Day Mode, the robot will:

- Display a "sunny" face on the LED matrix.
- Keep its servos at rest.

Block Code: Day Mode:

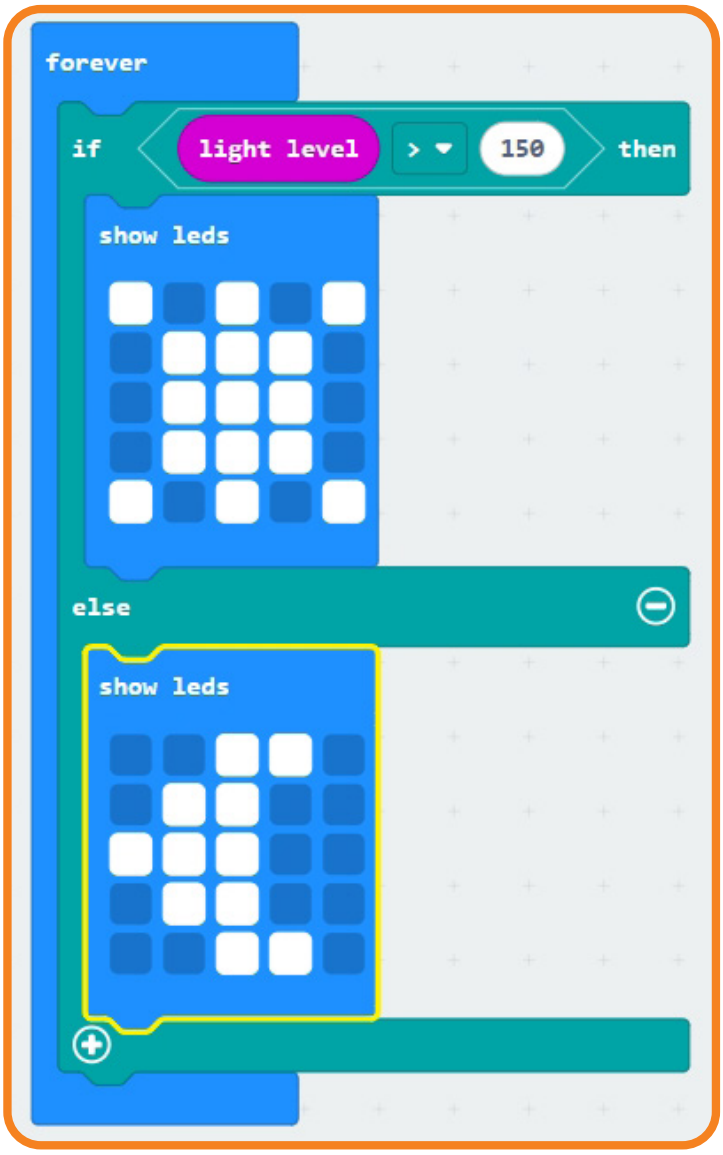


Step 3: Program Night Mode:

In Night Mode, the robot will:

- Brighten its lights with warm colors.
- Display a "starry" face on the LED matrix.
- Move its legs slightly to simulate a "breathing" effect.

Block Code: Night Mode:



Step 4: Test the Program:

1. **Upload the Code:** Connect your micro:bit and upload the program via *MakeCode*.

2. **Test Day Mode:** Shine a light on the micro:bit's LED matrix to simulate daytime. Observe the LED matrix and the robot's neutral state.

3. **Test Night Mode:** Cover the LED matrix to simulate nighttime. Observe the bright lights and leg movements.

Experiment and Customize:

Step 1: Add More Colors

- Introduce additional colors for different light levels.

Step 2: Adjust the Thresholds

- Change the light level threshold to fine-tune the transition between *Day and Night* Modes.

Step 3: Add a "Breathing" Light Effect

- In *Night Mode*, make the brightness pulse smoothly.

Step 4: Introduce Sounds

- Play soothing sounds at night and cheerful melodies during the day.

Troubleshooting Tips:

1. **Light Detection Isn't Accurate:**
   - Test the light level using a show number block to confirm the sensor is reading correctly.

2. **Transitions Are Too Fast:**
   - Add a pause block to prevent constant switching between *Day and Night* Modes.

3. **Lights Don't Change:**
   - Double-check the strip show color and strip set brightness blocks are placed in the correct part of the code.

What You've Learned:

- How to use the micro:bit's built-in light sensor to detect ambient light levels.
- How to program a robot to switch behaviors based on light intensity.
- How to create a *Day & Night* Lamp with lights, movements, and LED patterns.

Section 10: Questions

1: What is the JitterBit?

2: In previous sections, did we just build the JitterBit, or did we code it too?

3: Which parts of the JitterBit robot will we use to learn new coding concepts in this section?

4: Does this section focus only on building robots or coding as well?

5: Which sections prepared the learner for coding this robot?

6: What two parts of the robot will we use to explore more coding?

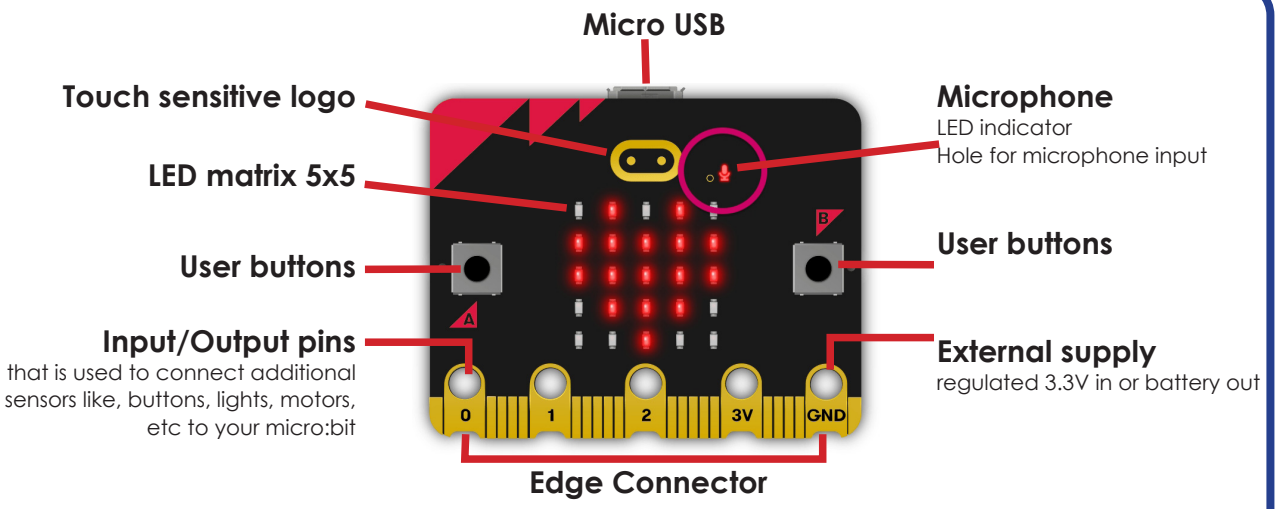
Section 11:

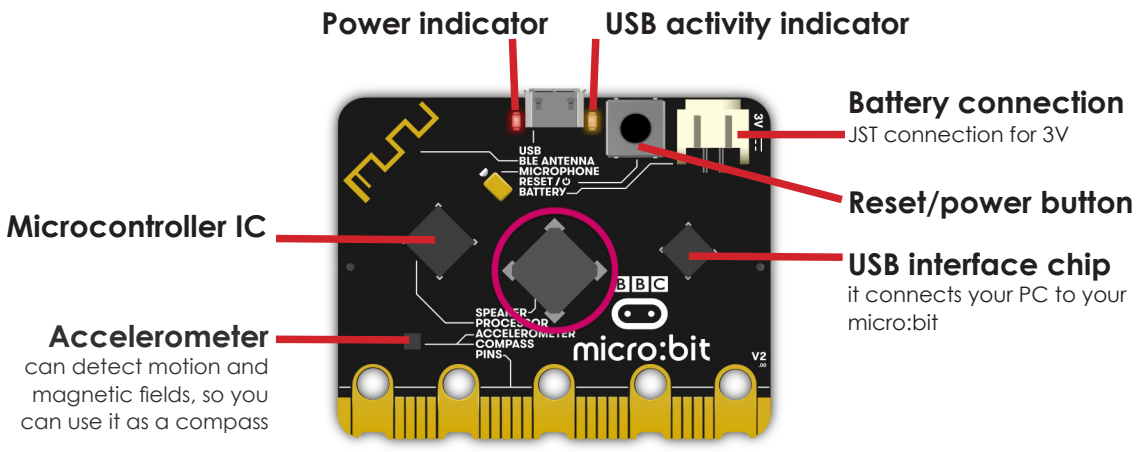
Microcontroller Parts and Its Blocks

Section 11: Microcontroller parts and its blocks.

Outcome

- micro:bit uses USB or batteries, with LEDs showing power and activity.
- LEDs are small, efficient, and controlled to emit various colours and brightness.
- The 5x5 LED matrix displays text, numbers, and graphics.
- Matrix coordinates start at (0,0) and end at (4,4).
- The LED matrix is a versatile, programmable display





Most microcontroller boards have additional electronic components on the board to make them more useful.

We will look at the most often used in this session. You saw this image in an earlier section, but we want you to have it nearby for this section.

Power supply:

The micro:bit microcontroller can be powered with a USB cable from your PC or batteries. You will often use the USB cable, because you will change your code often while learning. You will also receive power from your PC to the micro:bit.

On the back of the board, you can see the USB connector and the battery connector.

You will also see a red LED next to the USB connector that shows you if the micro:bit has power.

On the other side of the USB connector is the USB activity LED, which flickers as you upload or download data from the micro:bit.

Challenge:

Plug the microcontroller with the USB cable into your PC. See if the red power LED comes on. Upload code to the micro:bit and see how the hello activity LED behaves.

LEDs (Light emitting diodes)

Understanding LED's:

These days, you see LED lights everywhere. They are used as light bulbs in houses, in cars, and many other places.

There are many different types of LEDs, some very big and bright, and some small and not as bright as those on the micro-bit boards.

LED stands for Light Emitting Diode. Here's how you can think about it:

Light Emitting: This means it makes light. When you turn it on, it glows!

Diode: That's a fancy word for a special kind of switch. This switch only likes to let electricity flow in one direction. If you try to send it the wrong way, it stops the electricity like a one-way street.

Here's what you need to know:

Size: LEDs are tiny. They can be as tiny as a grain of rice or even smaller.

Color: They can come in all sorts of colours—red, green, blue, white, and even changing colours!

Brightness: Some LEDs are super bright, and others are softer. You can make them brighter or dimmer by changing the amount of electricity they use.

Energy: They use very little energy, meaning they can light up for a long time without consuming all the battery or power.

Control: With LEDs, you can turn them on or off or make them blink fast or slow. In gadgets like the micro:bit, you can even control them with code to show pictures or patterns or just light up when you want them to.

Where You See Them: Think about the little lights on your TV remote, the backlight of your watch, or even the traffic lights. Those are all LEDs!

LEDs can be controlled by a microcontroller, making them very handy in robotics,

Micro:bit LED matrix X and Y coordinates:

The micro:bit does not come with one LED; it comes with 25! If we group them together, we can form a matrix of LEDs and display numbers, text, and even basic graphics like a heart or smiley face.

What is a matrix?

A matrix is like a special kind of grid or Chessboard. It consists of rows and columns of blocks, as shown below:

0	1	2	3	4
1				
2				
3				
4				

You can put things into each block, like chess pieces. In robotics, we can put little lights (LEDs) in each block or even use a matrix to show how a robot should move, for example, moving from block 1 to block 3 in row 2.

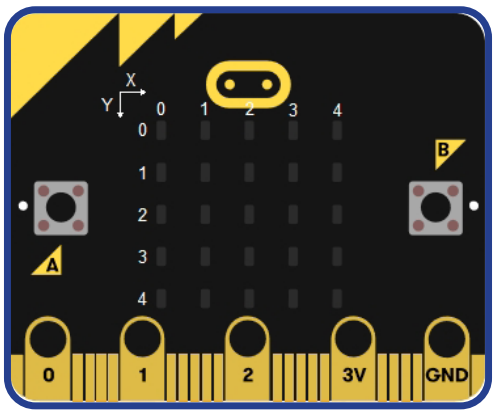
X & Y coordinates:

To know where the block is in the matrix, we need to understand the X & Y coordinates.

Rows are called X coordinates and columns are called Y coordinates.

In coding, you will often find that we start counting from 0 and not 1. So, a row of 5 blocks will count as 01234 instead of 1234.
In the image below, the LED matrix shows the numbers for you.

X0, Y0 will be the top left LED and X4,Y4 will be the bottom right LED

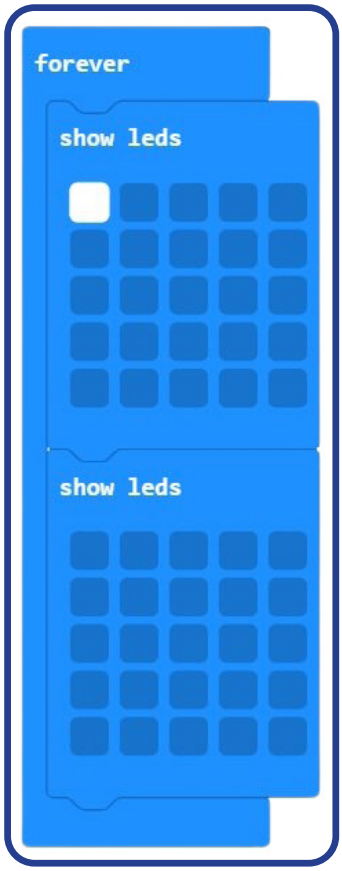


The *MakeCode* LED blocks allows you to light up the different LED's in anyway you want. The display blocks we already used show numbers and text on this matrix for us.

We can use a LED matrix like a display and we can even call this matrix an LED display.

Activity LED: Flicker one LED

Activity: Display LED's



1. In the Basic toolbox folder contains the show LEDs matrix. Clicking with your mouse on the LEDs will switch them on or off. The LEDs that is on will be white. This also indicate that specific LED will be on.

2. In the image there are 2 "show LEDs" blocks into the forever loop. The top "show LEDs" block shows there is one LED on.

3. In the second "show LEDs" block, there is no LED on.

4. The top left LED will start to flicker on and off.

Challenge:

1. Switch on any 3 LEDs and make them flicker on and off.

2. Create a light show where 3 LEDs is on and when they go off, 3 other LEDs go on.

3. Display an LED with the following coordinates X2 and Y3.

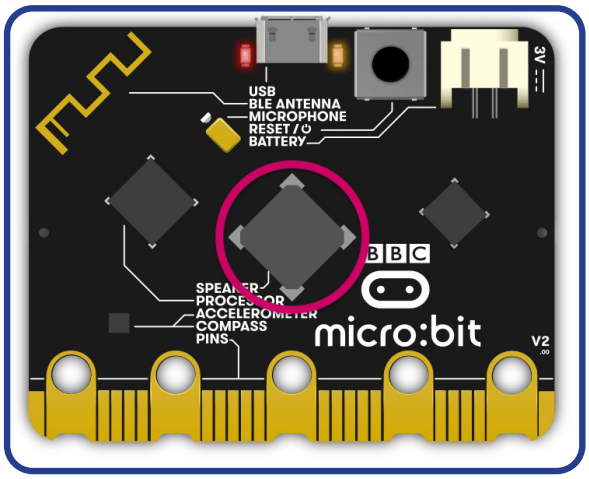
Speakers and buzzers:

A speaker can play very clear and realistic music. Microcontrollers can use add-on music amplifiers and play music, but we do not often use them in robotics.

A buzzer can only beep and play very basic tunes. In robotics, we are more interested in buzzers because buzzers can make all kinds of sounds like beeps and even play some basic tunes like "Happy Birthday."

These sounds are used to inform us of things like a beep when the microcontroller starts up or a beep when the temperature is too high.

Buzzers can indicate many things, such as buzzing when they find a coin or buzzing when they see an obstacle in front of them.



Activity Sound: Use the Speaker.

Instead of showing you step-by-step what blocks to use and how to use them, try to find the correct blocks yourself.

Tip: Look for a toolbox folder related to sound. The best option is to use the search box above the categories for words like play and pause.

Challenge:

Create a melody.

Section 11: Questions

1: What are some additional electronic components often found on microcontroller boards?

2: What are two ways that a micro:bit can be powered?

3: What does LED stand for?

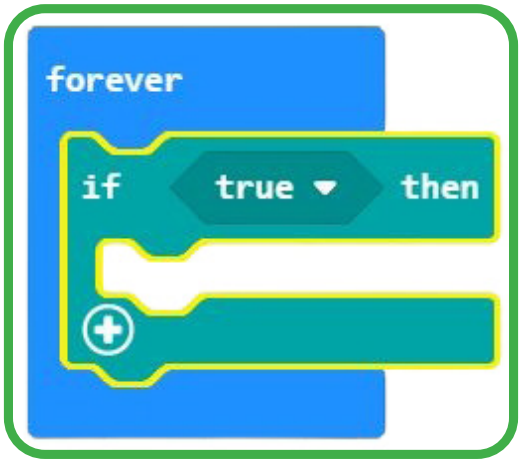
4: What is a matrix in the context of LEDs?

5: What are X and Y coordinates used for in the LED matrix?

6: What can you use a speaker or buzzer for in robotics?

Section 12:

More Operators and the if Statements



Section 12: More operators and the if statements:

Outcomes

- Understand operators
- Use math and comparison operators
- Use the "If" statement

We will show you in this section more ways to control the micro:bit microcontroller.

What are operators in robotics and coding?

In coding, operators are symbols or keywords that tell the computer to do something with the numbers or information (which we call "data") you give it. Here are some basic types:

Math Operators:

- + (plus) for addition, like in 3 + 5 which means "3 plus 5".
- - (minus) for subtraction, like 10 - 4.
- * (asterisk) for multiplication, like 2 * 3.
- / (slash) for division, like 8 / 2.

Comparison Operators: (great than / less than / equals)

- == (two equals signs) to check if two things are equal, like seeing if 5 == 5.
- > (greater than) or < (less than) to compare numbers, like 7 > 3 (is 7 greater than 3?).

Logical Operators:

- and or &&, which means "both must be true", like "Is it sunny AND warm?"
- or | |, which means "either can be true", like "Do you want ice cream OR cake?"

So, in coding, operators help you make decisions or do calculations. It's like telling the computer, "Hey, if this is true, do this; otherwise, do that."

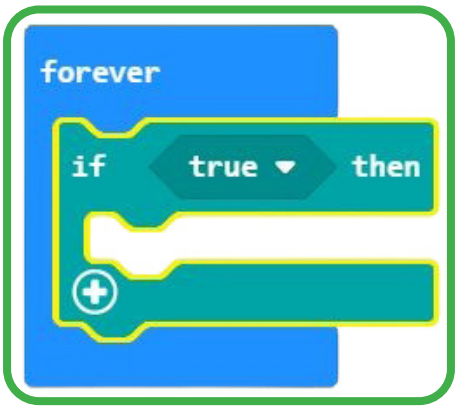
“If then” coding:

Imagine playing a game where you must decide what to do based on some conditions. The “*if then*” command works like this:

If something is true, then do something. For example, if a robot detects an obstacle in its way, it must stop.

In coding, we say: “If the obstacle in the robot’s way is True, then stop the robot.

You will find the “*If*” block in the logic toolbox folder. Drag the block inside the “*forever*” block.



The “*if then*” block, as is does not check anything to be true or false yet. We need an operator for the “*if then*” block to work.

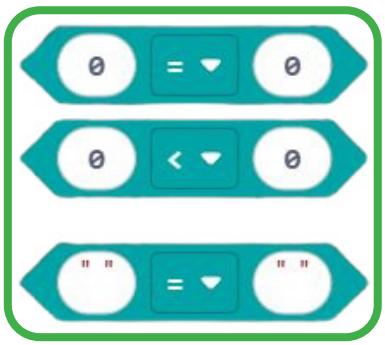
Comparison operators:

The type of operator that is often used with the “*if*” block is the comparison operator.

Comparison operators compare two or more values:

- = (equals) to check if two things are equal, like seeing if 5 = 5.
- > (greater than) or < (less than) to compare numbers, like 7 > 3 (is 7 greater than 3?).
- => (equal or greater than) or =< (equal or smaller than).

You will find all the comparison operators in the logic toolbox folder.



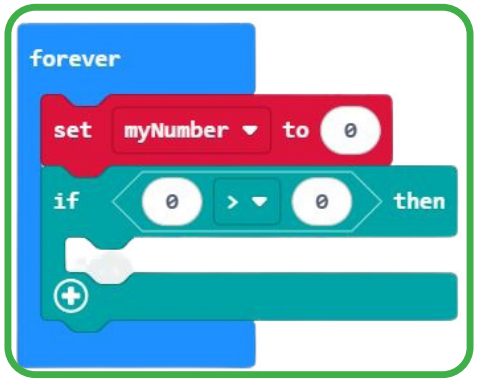
Above you will notice an operator to also match text (strings).

Activity: Using operators and the if block

In this activity, you will create a number variable to compare with another number.

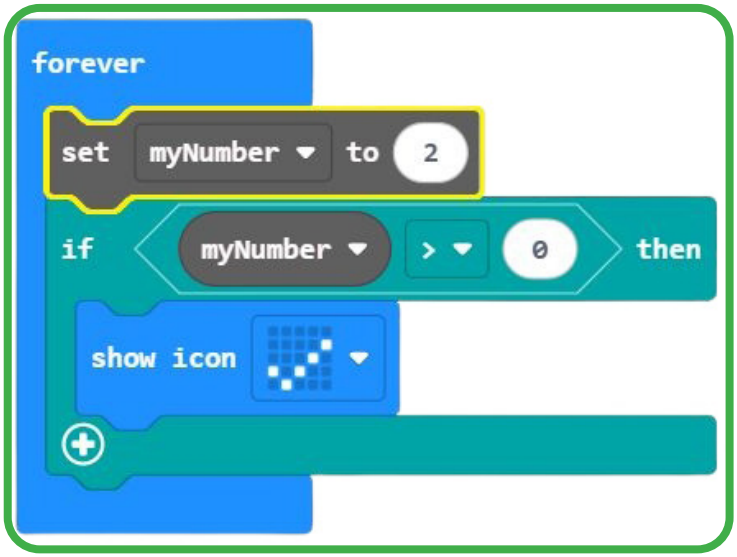
You can also compare different variables in coding.

1. Create a variable called *myNumber*.
2. Drag an "*if*" block inside the "*forever*" block
3. Drag the variable set block inside the *forever* block.
4. Drag a number comparison block over the "True" operator to change it with the comparison operator.



5. Next, we want to put an action inside the "*if*" block. If the comparison is true, we want the micro:bit to do something.
   Let's put a check mark icon on the display with a "*show icon*" block if the comparison is true.

   In the above image it will be false because 0 is not greater than 0. For this to be true we need to change the operator to "0" by clicking on the operator and choosing "=".
6. The last thing to do is to drag your variable from the variables toolbox folder to the first value in the "*if*" block.



In the above image, we set the variable to 2. The check mark icon will show on the screen because the variable *myNumber* is 2, which is greater than 0.

Section 12: Questions

1: What are operators in coding?

2: Name three types of operators.

3: What does the "if then" command do in coding?

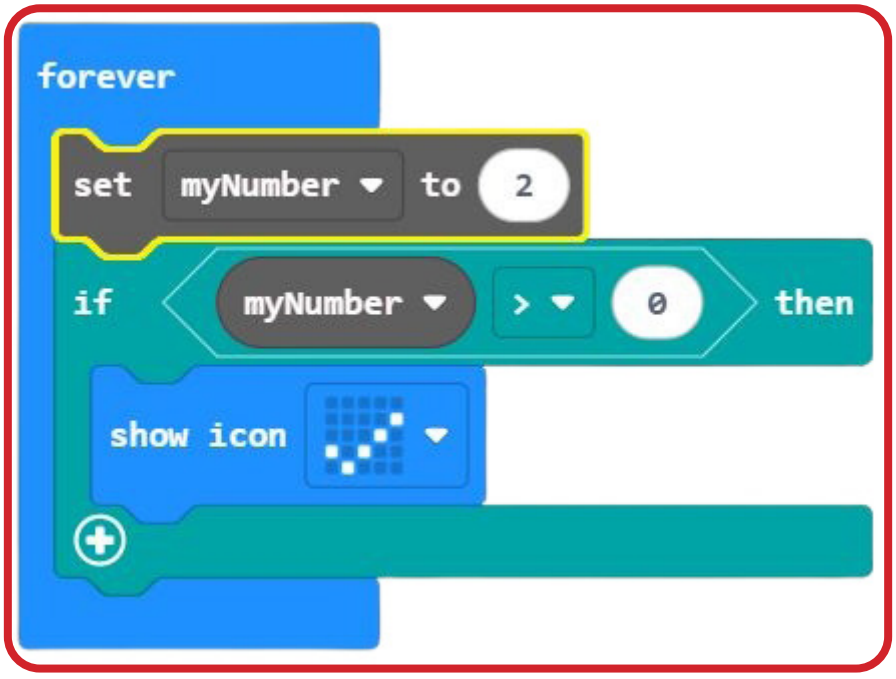
4: What are comparison operators used for?

5: What is an example of a comparison operator?

6: If you are using the "If" block, where do you find the correct operator?

Section 13:

More Control Blocks



Section 13: More Control Blocks:

Outcome

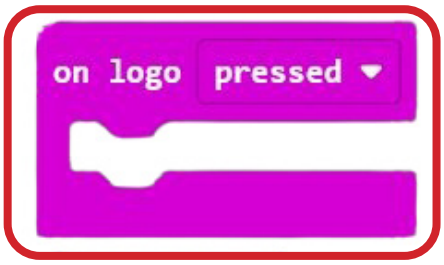
- Event triggers detect actions and respond immediately (e.g., display a heart).
- Repeat blocks simplify tasks by repeating actions a set number of times.
- Repetitions can be customized.

The Touch sensor block:

An event trigger is code that is used when a particular event happens. The micro:bit comes with a touch sensor. A touch sensor pictures up your finger when you press on it. It is different from a button, which you have to press, while a touch sensor senses your finger when you touch the sensor.

You will find the touch sensor on the micro:bit, as shown in the image below.

In make code the touch sensor block code is called "logo press"



Event triggers:

An event trigger is code that is used when a particular event happens, such as when you press a touch sensor or a button.

A microcontroller monitors the touch sensor or button constantly, waiting for someone to touch it or press a button.

When the microcontroller detects that a touch sensor, for example, is pressed, it can trigger something else to happen.

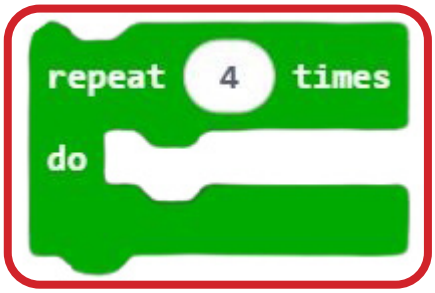
When you press a touch sensor, you want something to happen, such as displaying a heart on the LED display or making a sound.

IMPORTANT: Event trigger blocks like the touch and button blocks do not come in the forever loop. They are on their own in the workspace because they monitor constantly.

The repeat block:

This handy block can repeat the blocks you drag into it as many times as you want. The repeat block is in the loops toolbox folder.

The block below shows that all blocks inside the repeat block will run 4 times. You can change that number to any number you would like to repeat the blocks fewer times or more times.



Activity: Trigger the touch sensor and repeat actions

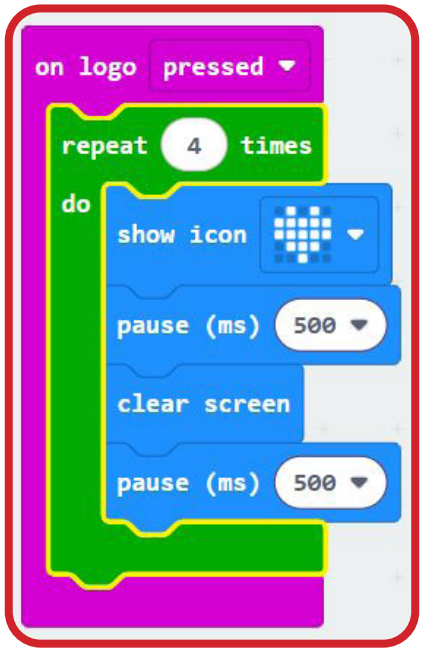
In this activity, we will use the touch sensor to trigger the repeat block, which will flash a heart on the screen 4 times.

1. Drag the "*on logo press*" block to the work area. It can not fit inside the forever block. It is on its own because it monitors the touch sensor all the time

2. Drag a repeat block into the "*on logo*" block, as below. Everything inside this block is repeated as many times as you like; in this case, it is four times.

3. Show a heart, then wait 500ms and clear the LED screen using the clear screen block so that all the LEDs go off. Then wait 500ms again.

4. This will repeat 4 times.



Section 13: Questions

1: What is a touch sensor?

2: What is an event trigger?

3: Why do touch sensor and button blocks exist outside the "forever" block?

4: What is the repeat block used for?

5: What does the "on logo press" block do?

6: What is an example of a practical way to use the repeat "forever" block?

About Bot Shop:

Vision: Bot Shop aims to make education fun, sparking curiosity and creativity rather than feeling like a chore.

Mission: Simple yet impactful, we make education fun and entertaining, turning **"giving up"** into **"what do I build next?"** Through the X-O-Skeleton series, we transform the micro:bit microcontroller into engaging learning tools.

Location and Production: Proudly made, designed, engineered, and manufactured educational robots in South Africa by Bot Shop.

Our products are tailored to meet the educational needs of South African learners and educators, combining advanced technology with innovative design.

Contact Us

Email: sales@botshop.co.za
Phone: +27 12 002 1651
Website: www.botshop.co.za