

LEDs



toolkit

Learning Digital Skills through Arts and Performance

A comprehensive approach to STEAM learning, where the Arts represent the beginning, middle and end of the learning experience.



Co-funded by
the European Union

2021-2-PT01-KA210-SCH-000048809

Author of this document:

- Francisco Medeiros (InovLabs)
- Hugo Melo (InovLabs)
- João Oliveira (InovLabs)
- Mick Mengucci (InovLabs)
- Nuno Charneca (InovLabs)
- David Sousa (NUCLIO)
- Kostas Soutos (Ellinogermaniki Agogi)
- Petros Stergiopoulos (Ellinogermaniki Agogi)

Artwork: Marco Martins

Partners in the project:

InovLabs (Portugal)



NUCLIO - Núcleo Interativo de Astronomia e Inovação em Educação, Portugal.

NUCLIO

Ellinogermaniki Agogi - Greece



ELLINOGERMANIKI AGOGI

AcroFit /Perfolie Arte Circus academy, Portugal



LeDS digital toolkit

The LeDS toolkit was developed to guide anyone interested in designing their digitally enhanced performances, and, at the same time to guide them in their STEAM journey to learn and apply wearables and electronics and programming for manipulation of sound, light and movement. The goal is to potentiate artistic expression and at the same time to increase the understanding of STEAM concepts and their applications in an enjoyable manner.

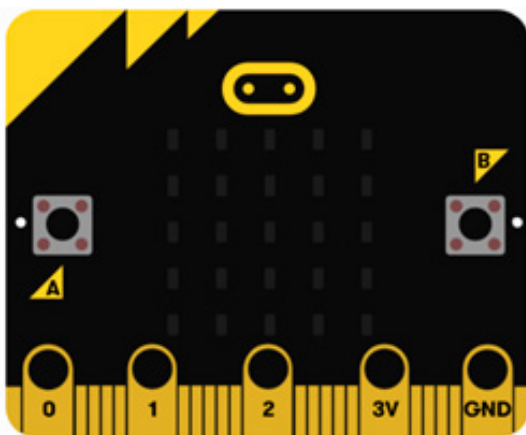
This toolkit was designed based on the continuous exploration and feedback from the students and teachers involved in the project..

Programmable LEDs

The micro:bit

To control LED effects or any other electronic component automatically, a microcontroller is required. It differs from a microcomputer in that it does not include an operating system, functioning more like the brain in the circuit. Microcontrollers possess a small processor and sufficient memory to receive digital and analog inputs in a circuit, process them based on the provided code, and generate outputs for the circuit.

In LeDS we use micro:bit, a pocket-sized microcontroller, designed by Microsoft for educational purposes. It features various sensors for input, such as light, sound, radio and accelerometer sensor, LED displays, and connectivity options, making



it a versatile tool for teaching coding and electronics¹.

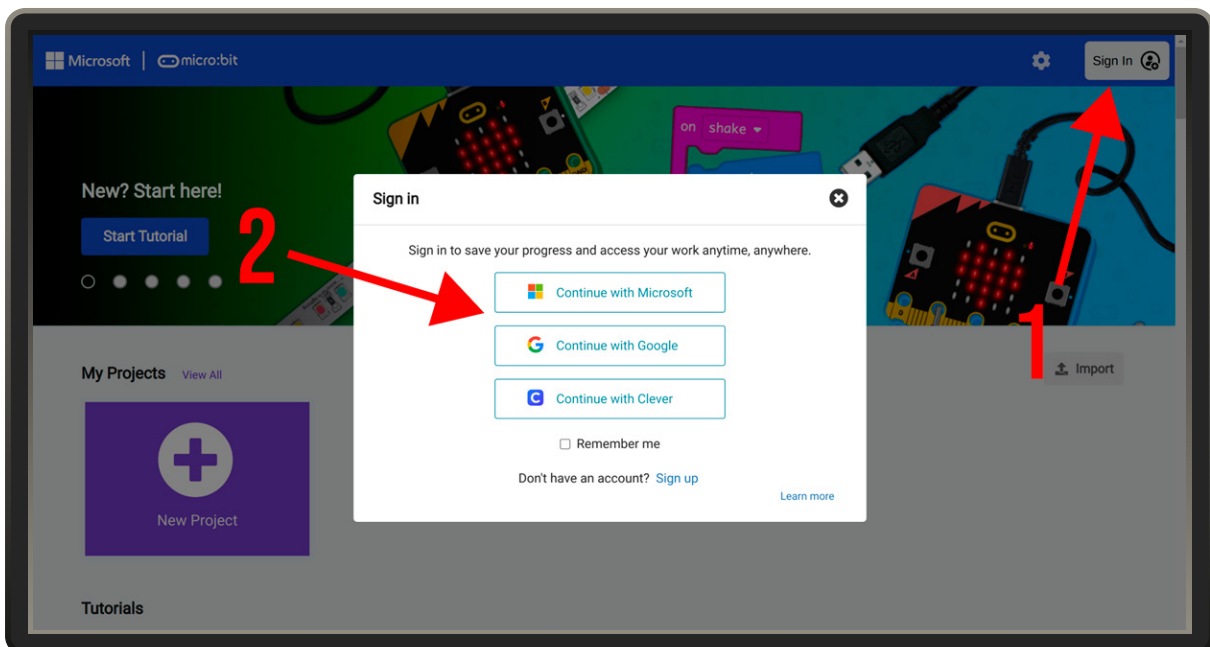
When teaching this to students it makes it more intuitive to use biological analogies. Don't mistake living beings with machines. Life is far more complex and mysterious than anything humans have invented until now! This is an important remark, as it teaches us to stay humble and open to new scientific

¹ More information about the micro:bit in the official website: <https://microbit.org/>

findings and perspectives. However, only for the sake of simplifying the understanding of microcontrollers, it is useful to compare machine functions with biological phenomena. For instance, you can think of the microphone in the micro:bit as “its ears”, the light sensor as “its eyes” and the accelerometer as “its vestibular system”. All these systems receive environmental input which the micro:bit can process, when programmed to do it, and eventually produce some kind of output. For instance, to its screen, in the form of drawings, or to its speaker in the form of sounds. When connected in a circuit, the micro:bit can receive and produce other kinds of input and output from additional sensors and electronic devices/components through its digital/ analog ports.

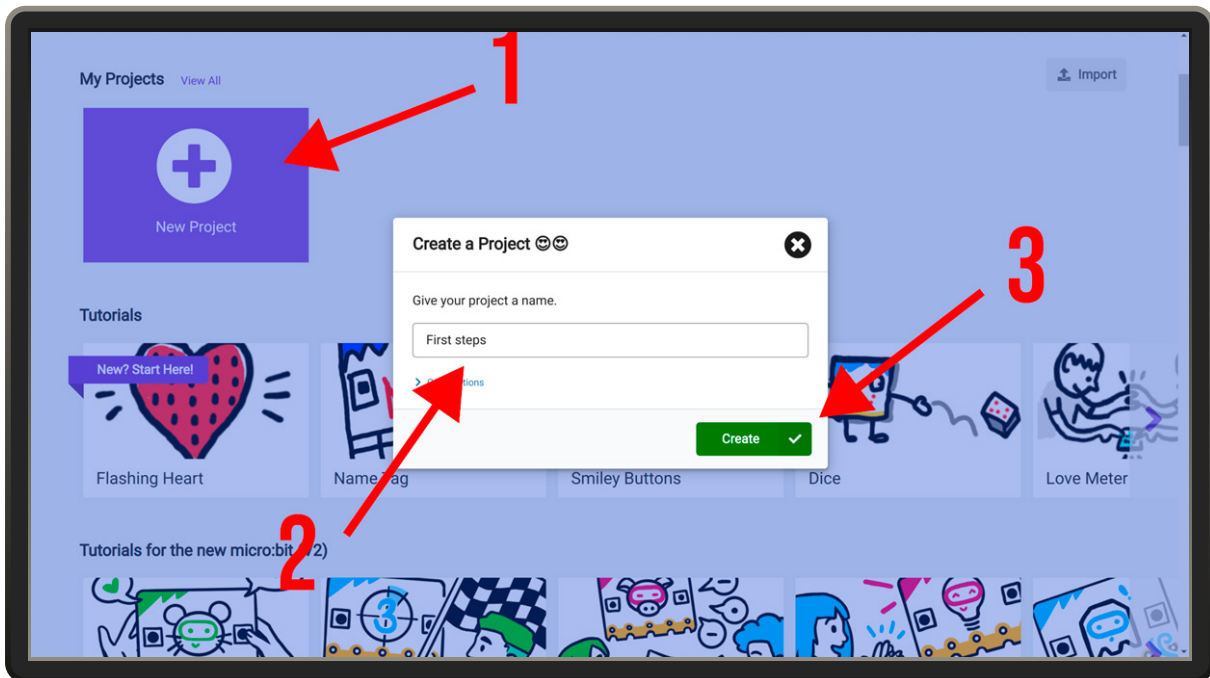
First steps

The micro:bit can be programmed by blocks, javascript or python from the Google Chrome Browser (for better compatibility) in the online platform makecode.microbit.org. It is free to create an account. You can authenticate with your Google or Microsoft accounts. This has the benefit of saving your projects in the cloud, becoming accessible from any device.



When creating your first project a tutorial will get you introduced to the platform and how to use it. It is highly recommended to follow it to clearly understand all steps of the process. After that, you can start right away with your own programming experiments. Or, you can go back to the makecode main page and follow the block

programming tutorials, namely, the “Flashing Heart”, “Name Tag”, “Smiley Buttons”, “Dice”, etc. Every change you make to your projects will be automatically saved, and every time you go back to the main makecode page, the projects you have created, including the tutorials you have followed, will appear in the “My Projects” section.



Programming with blocks is a great way to get introduced to computational thinking and to start learning the logic behind coding in a very intuitive manner. Since it is possible to simulate the micro:bit behavior in the platform it is not necessary to connect the micro:bit yet when experimenting the platform for the first time. Notice that everytime you change your program, the simulation is updated. When you are ready to test it for real, connect the micro:bit to your computer via the USB cable and follow the instructions upon clicking Download to transfer the program to the micro:bit. Everytime you change your code you will have to transfer it again to the micro:bit.

Having programmed the micro:bit, you can unplug it from your computer and plug in a portable 3V energy source which comes within the “BBC micro:bit Go” kit to be able to move with it and still run your programs.

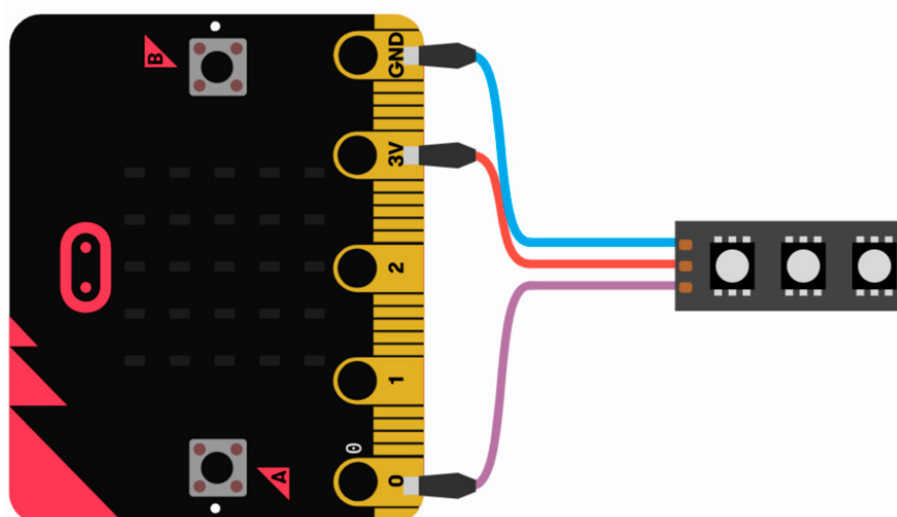
Setup a neopixel LED strip

Because of their great versatility, our toolkit for digitally enhanced performances is mostly based on LEDES effects. To produce LEDES effects with the micro:bit you need

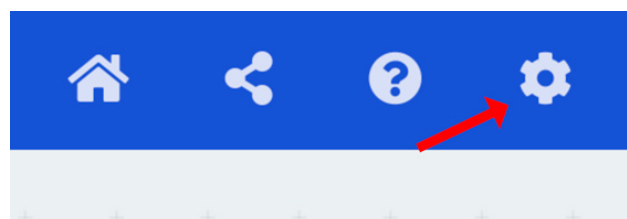
LEDS! However, it would not be practical to buy and connect individual LEDs and program effects for each one individually. Instead, you can use a neopixel LED strip.

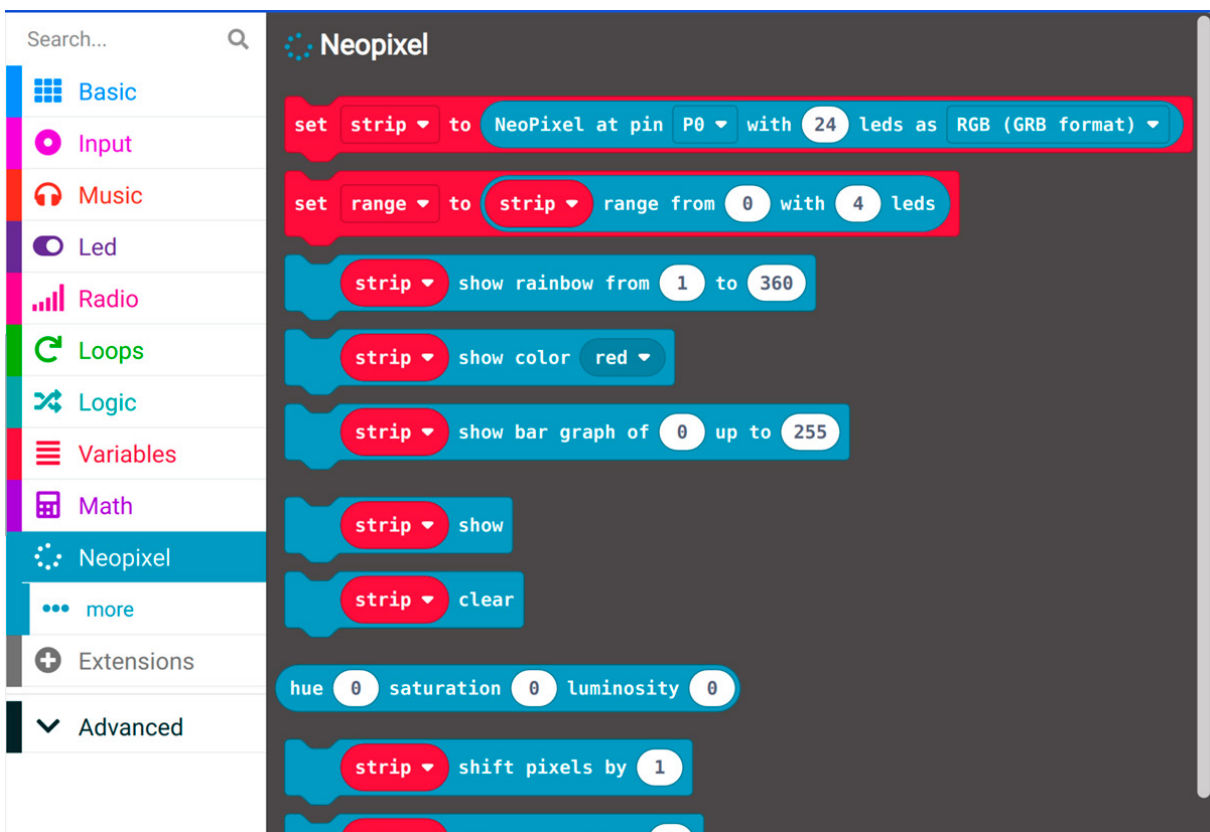
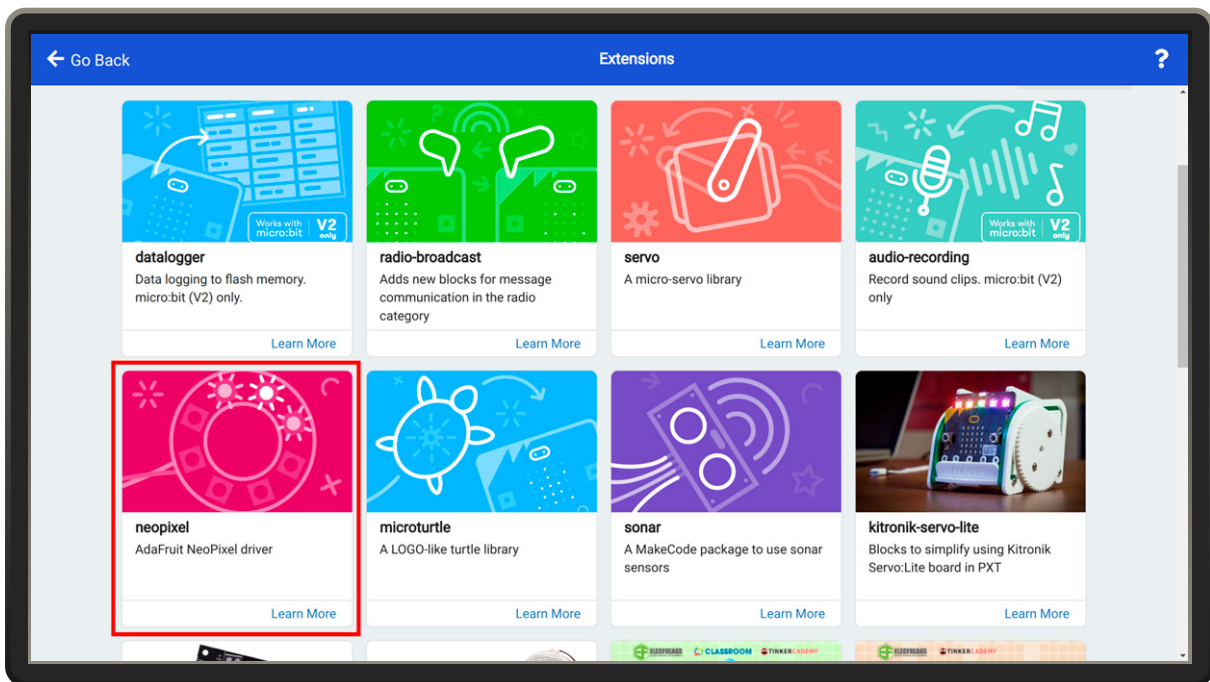
A neopixel strip is an addressable LED strip in which all LEDs are already connected to each other and whose color can be individually programmed. To connect and control a led strip, 3 wires are needed to:

1. connect the GND terminal at the LED strip to the GND terminal at the micro:bit;
2. connect the voltage input terminal at the LED strip to the voltage output terminal (3V) at the micro:bit;
3. connect the data input terminal at the LED strip to a digital port terminal (e.g., 0) at the micro:bit;



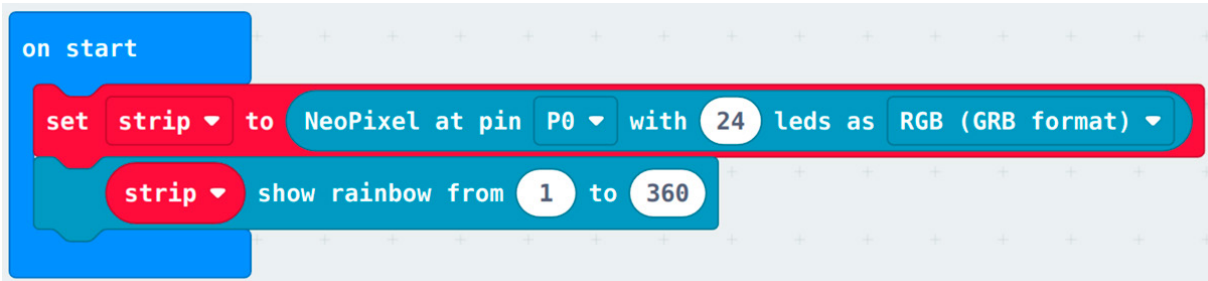
The next step is to program the micro:bit to control the LED strip. To do that, add the NeoPixel extension to MakeCode: inside a MakeCode project, click the settings wheel on the top bar in the right corner, choose Extension from the menu and select the NeoPixel extension from the list. After that, a new category of code blocks called “Neopixel” will appear in your category list, from where you can choose the code blocks necessary to program the micro:bit to control neopixel LED strips.



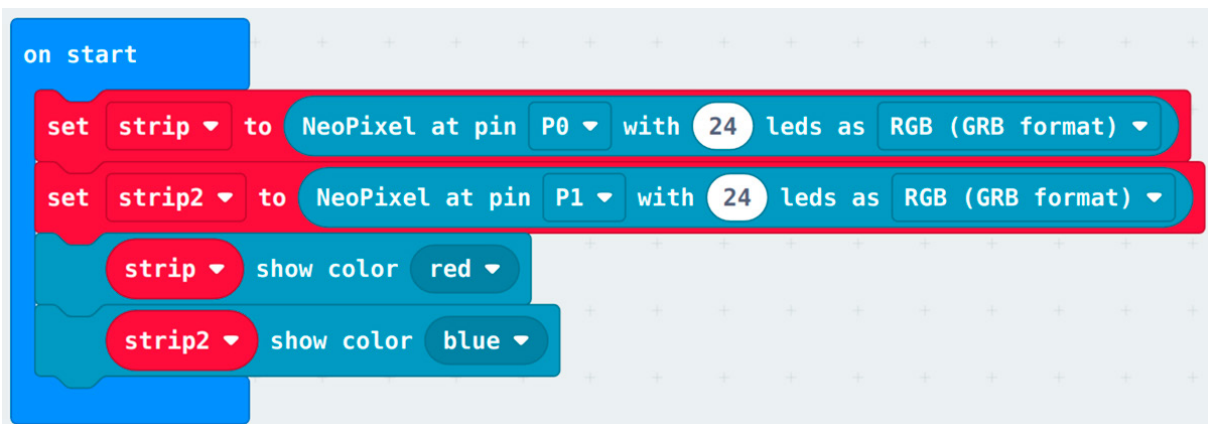
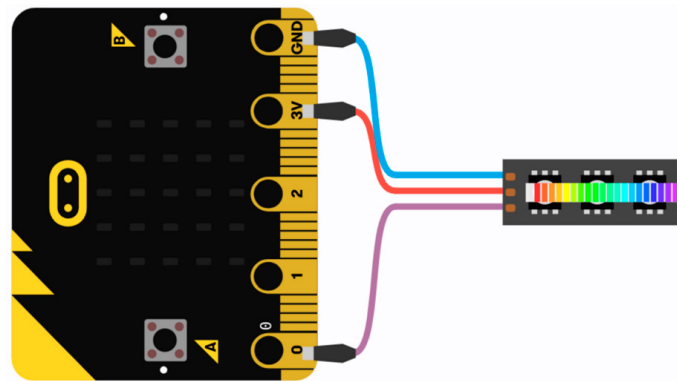


To be able to turn a NeoPixel LED strip on or off, you must first initialize the NeoPixel class. This means that you are telling MakeCode that a number of NeoPixels exist and that they are Red, Green, Blue (RGB) responsive. You can do this by adding

a set “strip to... block” to the “on start block”. Once you have done this, you can then call the show method on this class².

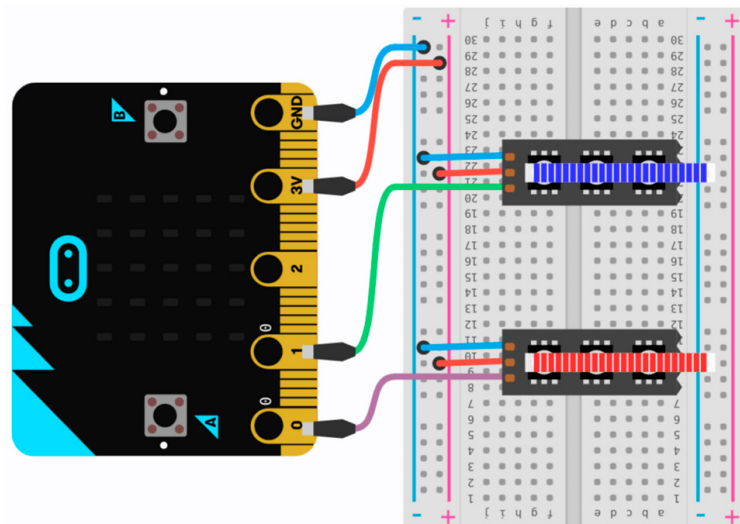


If you plan to control more than one strip simultaneously, you have to connect each one to its own digital output terminal on the micro:bit. You can then use variables to define each strip individually referring to the micro:bit digital output terminal the strip is connected to. This allows you to call each strip independently.



To make a neopixel setup portable and practical to use in performances, it's useful to use a micro:bit shield (more on this in the section “Wearables” of the toolkit, page XX) that allows both, to plug in batteries directly, and to connect multiple LED strips easily to the micro:bit digital output terminals.

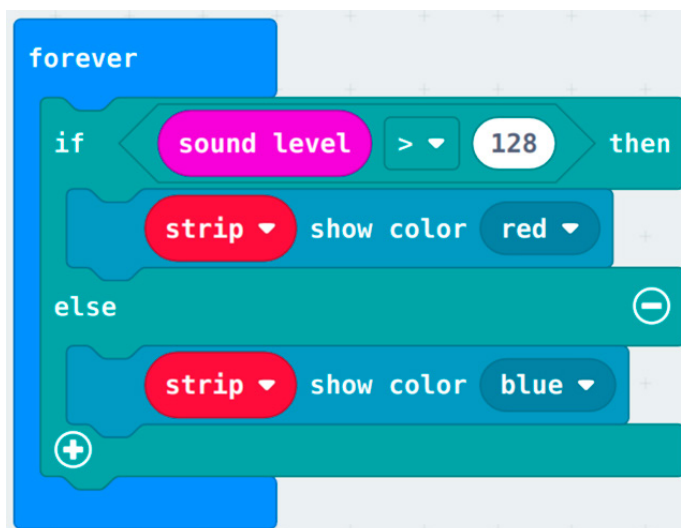
² Help and support on how to use Neopixels with the micro:bit: <https://support.microbit.org/support/solutions/articles/19000130206-using-neopixels-with-the-micro-bit>



Learning to program basic neopixel effects controlled by sound, light and acceleration.

The simplest examples are the best to get us started! What follows will serve as a skeleton and reference for you to start understanding and making your own micro:bit controlled neopixel effects. It all depends on logic. Logic is the basis of all programming. Would you be able to speak a new language just by adding up new words to your vocabulary? Without the grammar you would not know how to combine the words to make sense. It is exactly the same thing with programming languages! It does not matter whether you are learning Java, Javascript, C, Python or block programming. If you don't understand logic you can't speak to the micro:bit or any other microcontroller. But if you do, with enough experience you will be able to understand and reproduce any complex piece of code provided to you.

Let us start with one LED strip only. First thing is to set up the LED strip as in the first LED strip example. Don't forget to initialize the LED strip within the "on start" block! Now, consider a scenario where you intend to set all LEDs to red when the sound level detected by the micro:bit exceeds a prespecified threshold and to blue otherwise. Translating this to block language, you will want to



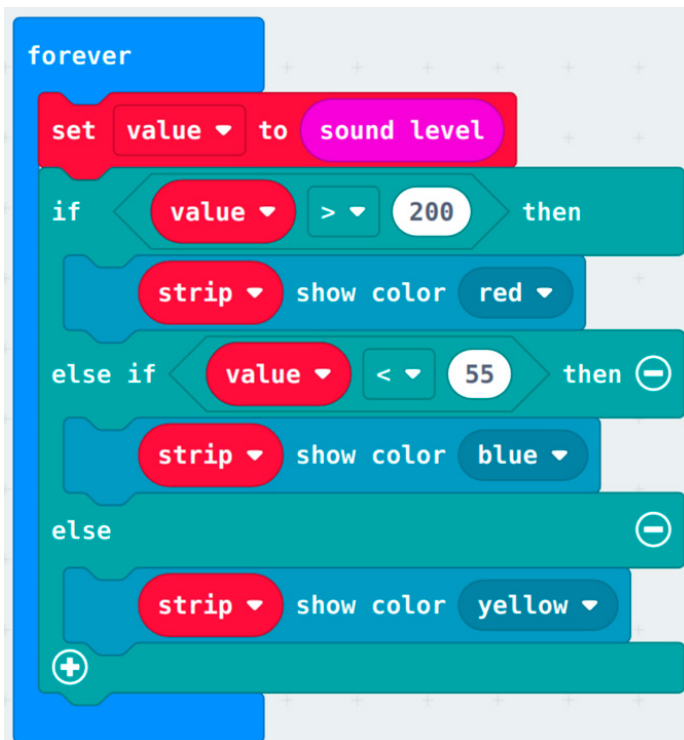
use the "forever", "If... then ... else" and comparison blocks to evaluate whether at each moment it is true that the sound level detected is above a certain threshold.

This means that, if the comparison between the measured sound level and the defined threshold returns true, that is, if the sound level is above 128, all LEDs are set to red, and the execution of the code inside the "else" part is skipped. If the comparison is false, that is, if the sound level is below or equal to 128, the execution of the show color red is skipped, and the "show color blue" block inside the "else" part is executed.

Note: Remember that blocks can only fit other blocks of complementary shape and that they have the color of the category to which they belong. At any moment, if you need help with a specific block, right click on it and click help.

Now, you might wonder why the number 128. This number was arbitrarily chosen to represent the midpoint of the binary precision scale of the sound level sensor, which ranges from 0 to 255.

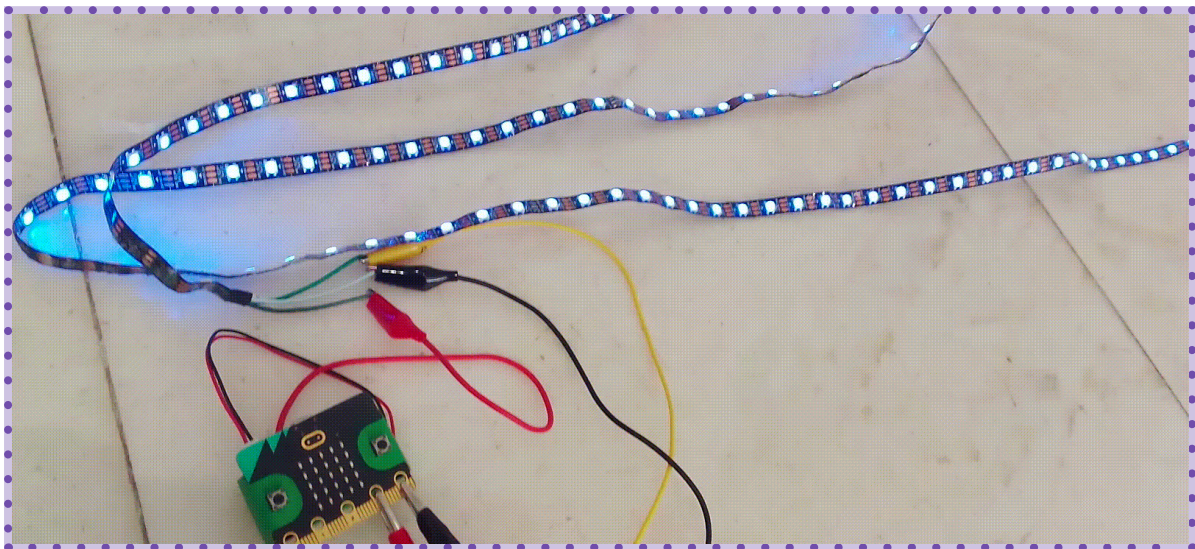
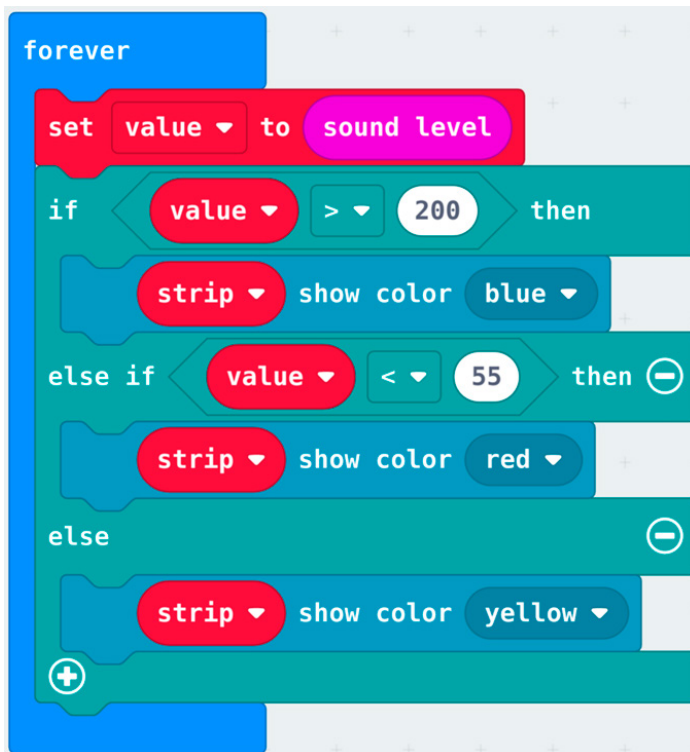
If we wish to set different colors for various intervals of the detected sound level, we need to click the plus sign at the bottom of the “if...then...else” block to add more things to evaluate. You can add as many as you wish. For the sake of simplicity, let’s define only three: values above 200 for loud sounds, values under 55 for quiet sounds, and values in between for sounds at moderate levels. For accuracy, it’s essential to define a variable representing the sound level measurement before any comparison. Otherwise, if you directly compare the sound level measurement at each step within an if block, you might end up comparing different values across evaluations.



This code means that, for a predefined sound level measurement set to a value, the “if” block begins by checking if the value is above 200. If true, it executes the “show color red” block, setting all LEDs to red, and skips the evaluation of the second comparison and the “else” part, restarting from the beginning. If the first condition is false, it skips the “show color red block” and evaluates the second comparison. If the second comparison is true, it executes “show color blue” and skips the “else” part, restarting from the beginning. If neither

comparison is true, the “else” part is true, and the “show color yellow” block is executed. The “if” block then restarts from the beginning.

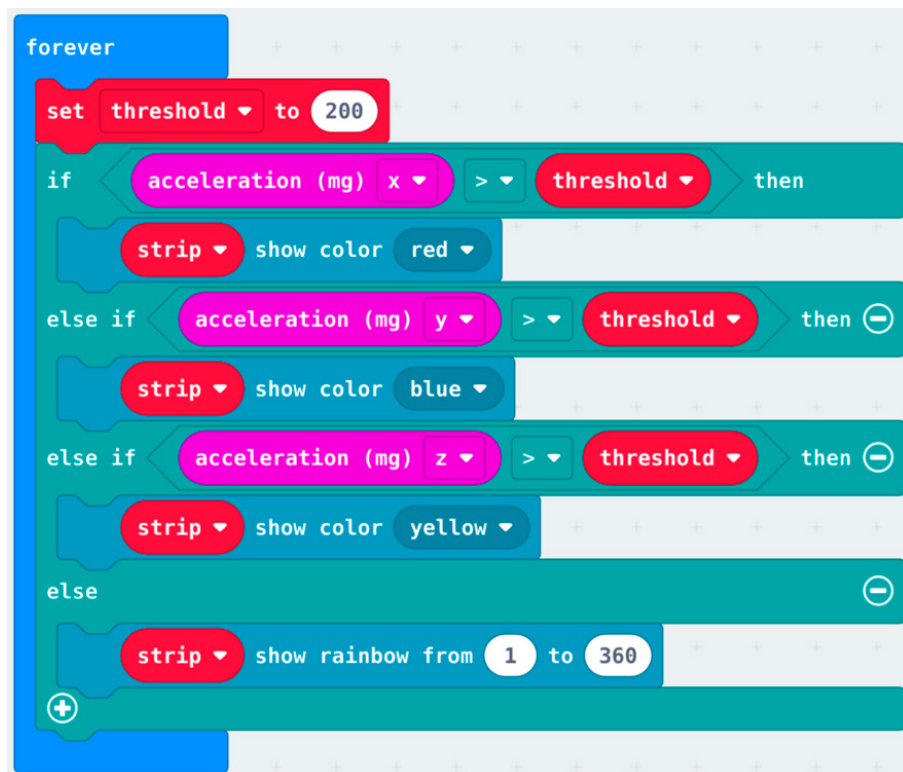
Because the light level sensor uses the exact same binary precision scale as the sound level sensor, the same blocks can be used to perform the same effects. It suffices to replace the sound level pink block by the light level block of the same category. Lets anyway swap the “show color red” and “show color blue blocks” to test your understanding of the logic flow.



You can also leverage the accelerometer to achieve similar effects based on the measure of the acceleration³ magnitude or in a specific direction. Simply replace the pink block light level with an acceleration block of the same category. Play with the acceleration strength and the three coordinates X, Y and Z. The binary precision scale of the accelerometer ranges between -1023 and 1023.

³ <https://makecode.microbit.org/reference/input/acceleration>

Another application of the accelerometer is to generate distinct effects based on different movement directions. You can achieve this by setting a common threshold value to compare the acceleration measured in each direction.



Test different thresholds values to manipulate the sensitivity of the micro:bit accelerometer and adjust it to your needs. You can also define different threshold values for different directions of movement.

One example of another way to use the accelerometer, developed by the Greek students for their performance, is to compare two acceleration values, for instance in the X and Y coordinates, respectively with two threshold values simultaneously by using the “and” comparison block. When one of a set of such comparisons holds true it executes a “strip show color ...” block with the intended color. This, according to the code, will change the color to yellow, red, green, or indigo depending on the type of movement performed, namely the ‘rotation,’ ‘fall and rebound,’ and ‘spirals’ movements.

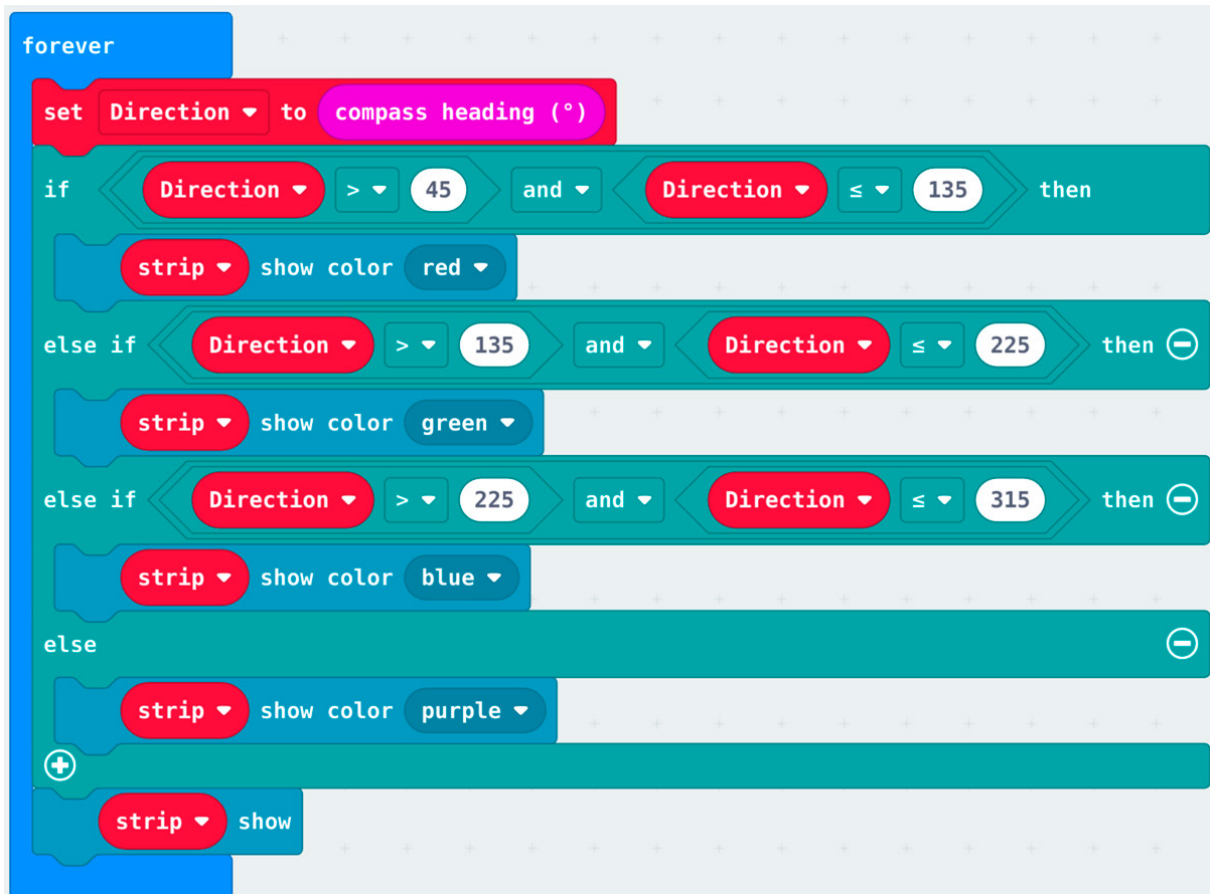


```
on start
  set strip to NeoPixel at pin P0 with 90 leds as RGB (GRB format)

forever
  if acceleration (mg) x < 0 and acceleration (mg) y < 500 then
    strip show color yellow
  else if acceleration (mg) x > 500 and acceleration (mg) y > 1000 then
    strip show color red
  else if acceleration (mg) x > 1000 and acceleration (mg) y < 2000 then
    strip show color green
  else
    strip show color indigo
  pause (ms) 100
```

The code starts with an 'on start' block that initializes a NeoPixel strip at pin P0 with 90 LEDs in RGB (GRB format). A 'forever' loop follows, containing four conditional blocks based on acceleration in mg. The first block checks if x < 0 and y < 500, showing yellow. The second checks if x > 500 and y > 1000, showing red. The third checks if x > 1000 and y < 2000, showing green. The fourth is an 'else' block showing indigo. A 100ms pause block is at the end of the loop.

Another example that is equivalently simple uses the compass heading as input.



```
forever
  set Direction to compass heading (°)
  if Direction > 45 and Direction ≤ 135 then
    strip show color red
  else if Direction > 135 and Direction ≤ 225 then
    strip show color green
  else if Direction > 225 and Direction ≤ 315 then
    strip show color blue
  else
    strip show color purple
  strip show
```

The code starts with a 'forever' loop. The first block sets a variable 'Direction' to the 'compass heading (°)'. This is followed by three conditional blocks: the first checks if Direction > 45 and Direction ≤ 135, showing red; the second checks if Direction > 135 and Direction ≤ 225, showing green; the third checks if Direction > 225 and Direction ≤ 315, showing blue. An 'else' block shows purple. Finally, a 'strip show' block is at the end of the loop.

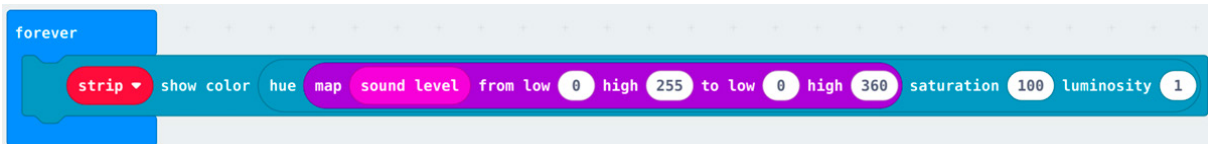
Now that you know the basics, try making your own effects using the micro:bit sensors and experimenting with the neopixel LED strip capabilities.

Brightness and hue effects

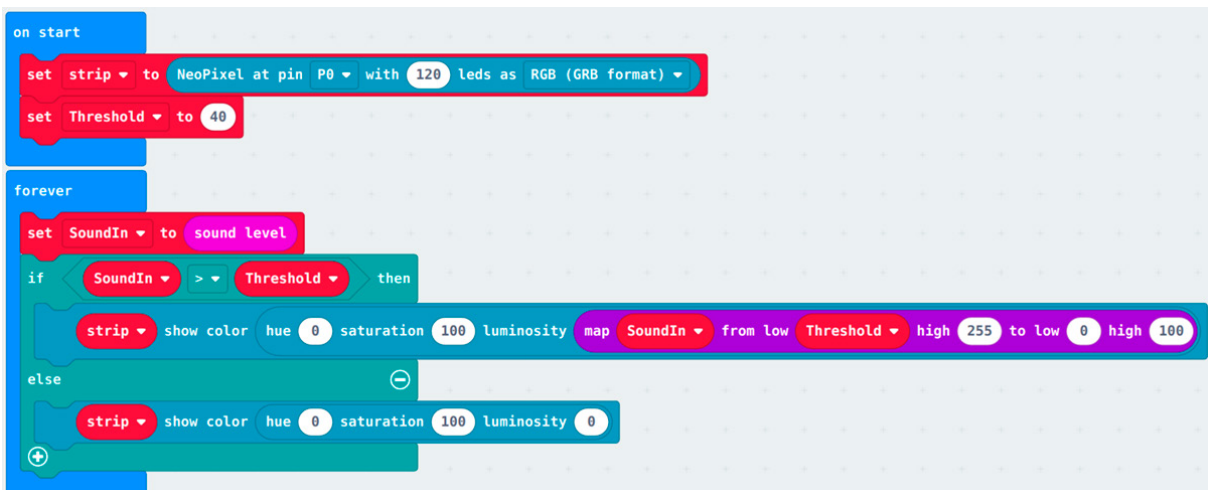
For a brightness and hue effect controlled by sound, light or any other kind of input data, the neopixel library offers a special block that lets you manipulate the hue, saturation and luminosity (brightness) with precision. You just need to map your input range to the hue range (0 to 360), or the luminosity range (1 to 100) you want to work with. You can experiment with different numbers to see how they change the LED strip. So, for instance, if you want the luminosity of a specific color, say red (hue = 0) to vary with the sound level you can replace the color in the “show color” block with the new color you have just defined:

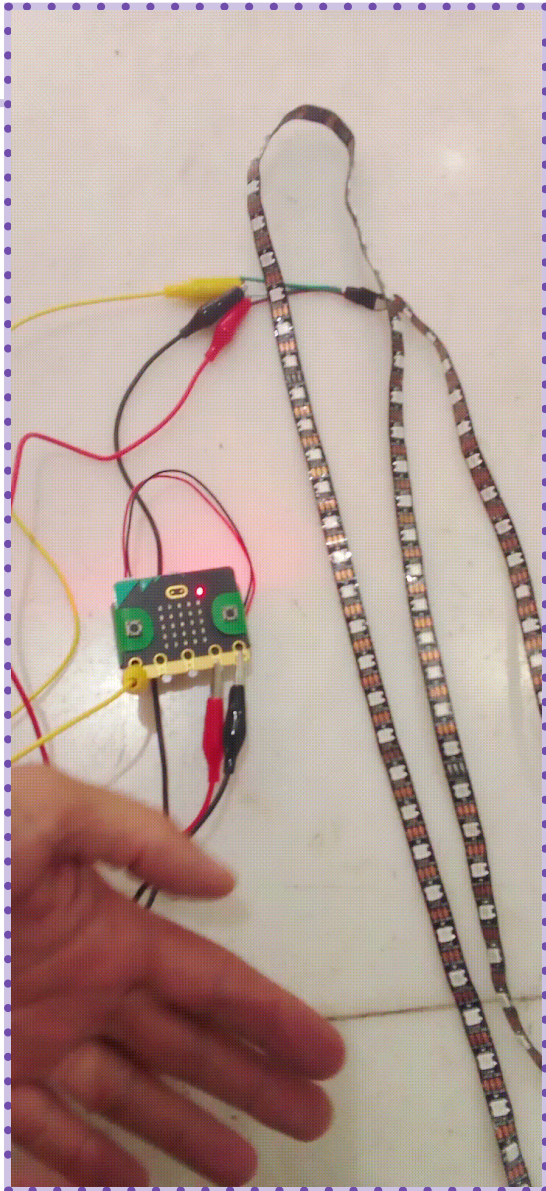


For a hue effect change the mapping number and move the block to hue:



It's also possible to combine both effects. We'll leave that for you to experiment with. Now, to go back to logic, you might want to generate the effect only when the sound level is above a certain threshold that you can define on start. Recall everything you have learned until now and put it all together in blocks.





As before, to replicate a similar effect for the light level, you only need to substitute the pink sound level block with the light level block from the same category. For acceleration, you would also need to remap the input based on the accelerometer's binary precision scale (0 to 1023).

Running LEDS effect

You can utilize sensor input data to generate a visually appealing running LED effect along the Neopixel strip. The colors and brightness of the LEDs dynamically change based on the magnitude of the input data. The LEDs will dynamically run and shine, their colors and brightness changing in response to the strength of your movements.

```

on start
  set strip to NeoPixel at pin P0 with 60 leds as RGB (GRB format)

forever
  pause (ms) 5
  strip shift pixels by 1
  strip show

on event from MICROBIT_ID_ACCELEROMETER with value MICROBIT_ACCELEROMETER_EVT_DATA_UPDATE
  set Force to acceleration (mg) strength
  if Force > 1200 then
    set force2hue to map Force from low 1200 high 1772 to low 0 high 360
    set LUM to 10
    for index from 0 to 4
      do
        set color to hue force2hue saturation 80 luminosity LUM
        strip set pixel color at 4 - index to color
        set LUM to LUM + 10
  
```

The core of the code here is the “strip set pixel color at” block inside the green “for” loop code block. Because the index of the “for” loop ranges from 0 to 4, this loop will set the LEDs with indexes 0, 1, 2, 3 and 4 to one of 360 colors, and a brightness (luminosity) which varies with the index of the LED. The color of the LEDs “force2hue” is chosen from mapping the “Force” value (between the threshold 1200 and its maximum possible value 1772⁴) to the range of 360 possible colors.

The control block in black is meant for the micro:bit to always pick up accelerometer events and override the current animation to create multiple strings of pixels flowing along the strip. The movement effect is generated from within the “forever” block by

⁴ Since, in each movement coordinate (X, Y, Z), the acceleration value is at most 1023, its magnitude can be calculated using the Pythagorean theorem: $\sqrt{X^2 + Y^2 + Z^2}$, resulting in a maximum magnitude of $\sqrt{3 * 1023^2} \approx 1772$.

simply shifting the “pixels” by 1 at each 5 ms, which requires a “strip show” block to take effect.

There are many different ways in which you can change this code to fit your performances. For example by modifying:

- the range of colors you want you use;
- the sensitivity of the micro:bit to movement by changing the threshold to which the “Force” value is compared in the “if” block;
- the brightness of the LEDs by changing their luminosity;
- how fast the LEDs “run” by changing the pause value;
- how many LEDs are used by changing the index range in the “for” loop;
- the triggering input sensor from acceleration to sound level or light level;

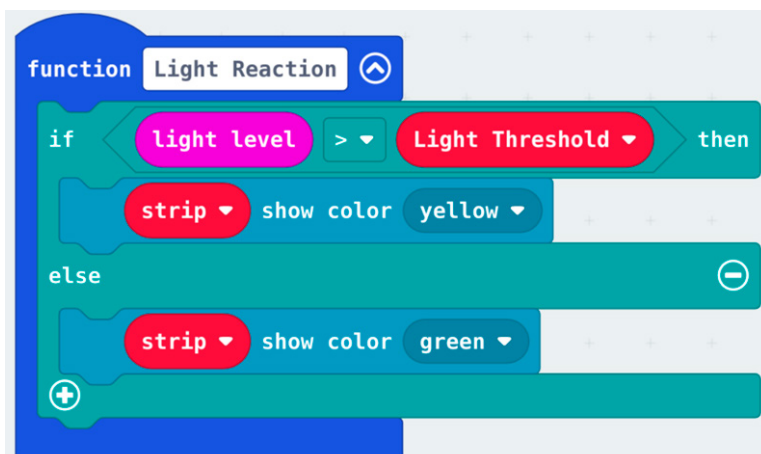
Visit the editable version at:

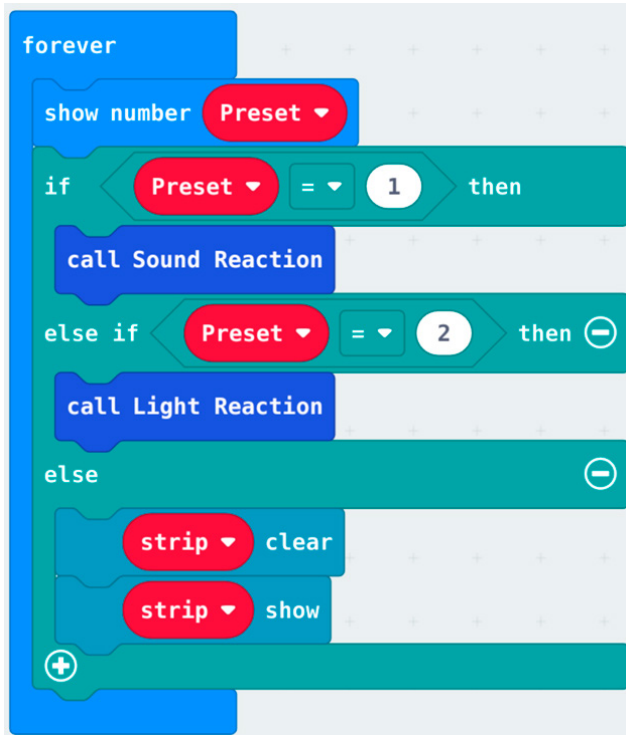
<https://makecode.microbit.org/S62918-34908-89381-60752>

Preset System

Instead of having individual micro:bits programmed for different effects, you can program one micro:bit for multiple effects and use the micro:bit buttons to switch between different effects during your performance. Let’s build a simple preset system consisting of an effect triggered by light and another one by sound.

To be able to choose from between different effects with the micro:bit buttons we need to define the effects as functions that we can call when pressing the buttons. Can you find the “function” block? To make your code easily readable, name your functions according to the effects they produce or the type of input they react to. Don’t forget to keep track of the variables which have not yet been defined!

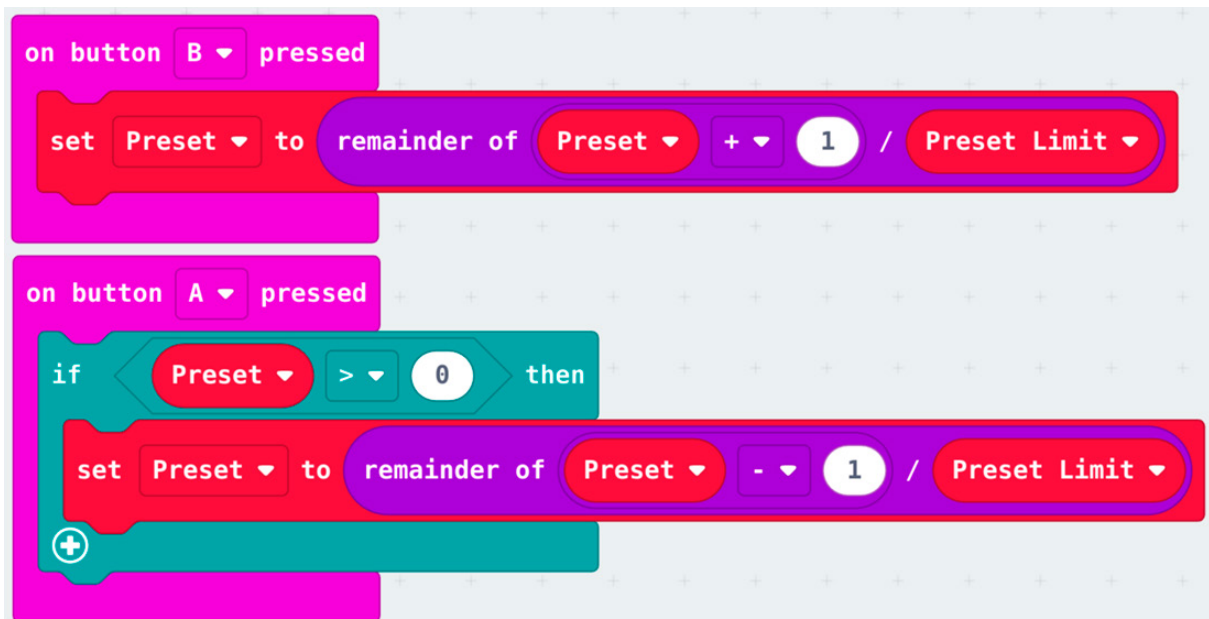




The “forever” block will be useful here to check what is the preset chosen and execute it.

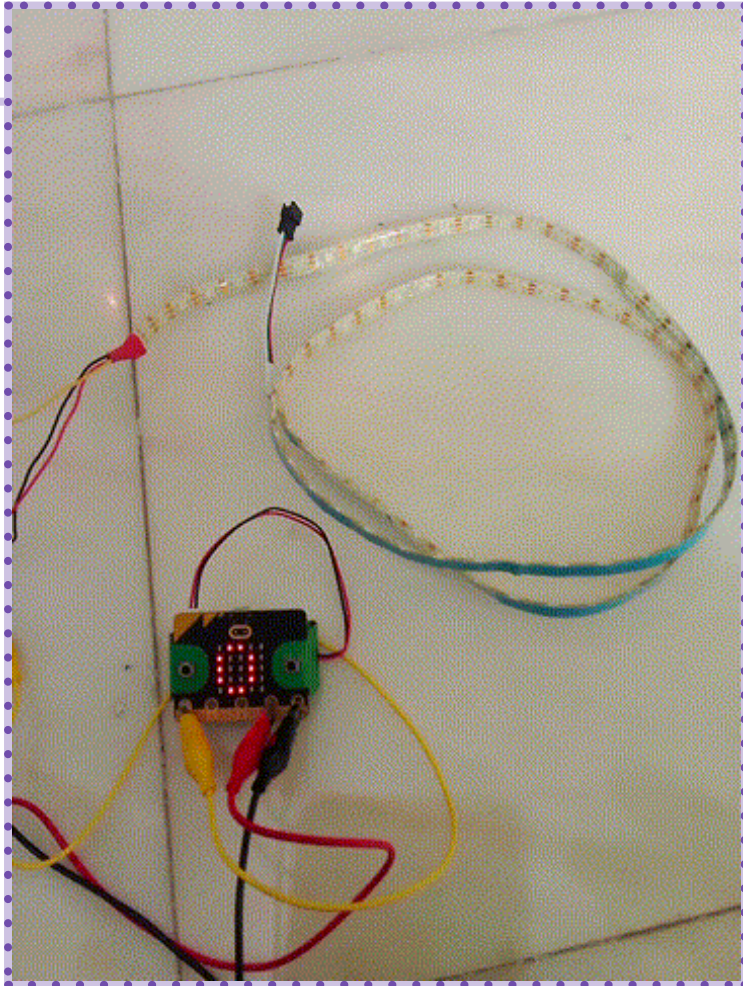
Note that no effects are generated when the Preset value is other than 1 or 2. To choose its value, we use the micro:bit buttons and a mathematical trick: the remainder of division! Have you ever wondered what is it useful for? A wonderful application is easily cycling through a predefined range of values. This is super handy in many programming situations.

To cycle the “Preset” value up we use the “on button B pressed” input block and set the Preset value as the remainder of the division between the “Preset” current value plus 1 and the “Preset Limit”. To cycle it down, we use the “on button A pressed” input block, and simply replace the plus sign with a minus sign in the equation, but use an if block to prevent negative values.



To finish, don't forget to define the variables you are using

You can access the editable version here and change it as you see fit: <https://makecode.microbit.org/S87691-40958-94224-33779>



Try building a preset system for your own LEDs effects. You can put in as many functions as you wish and cycle between them with only two buttons as long as you rewrite the “forever” block according to your functions’ names and redefine the “Preset Limit” value.

Radio Preset System

Making a radio preset system that allows you to control multiple micro:bits wirelessly with one micro:bit will allow you to switch between the different effects you wish the micro:bits to generate while watching the performance from the outside.

A straightforward and practical way for switching the preset wirelessly involves breaking the preset system code earlier into two separate programs, incorporate the radio functionality, and transfer these programs to separate micro:bits, the emitter and the receiver micro:bit: while the emitter micro:bit will use the A and B buttons to send the preset value over radio and that’s it, the receiver micro:bit will receive the preset value over radio and execute the corresponding effect. Let’s look at it step by step.

Emitter micro:bit

First, we need to program the emitter micro:bit. The “radio set group” block within “on start” instructs the micro:bit to communicate (send and receive radio signals) with all other micro:bits within the same group, so you always need to set the radio group, whether the micro:bit is a receiver or an emitter. A block “radio send number” will emit the radio signal.

```

on start
  radio set group 1
  set Preset to 0
  set Preset Limit to 3

forever
  show number Preset

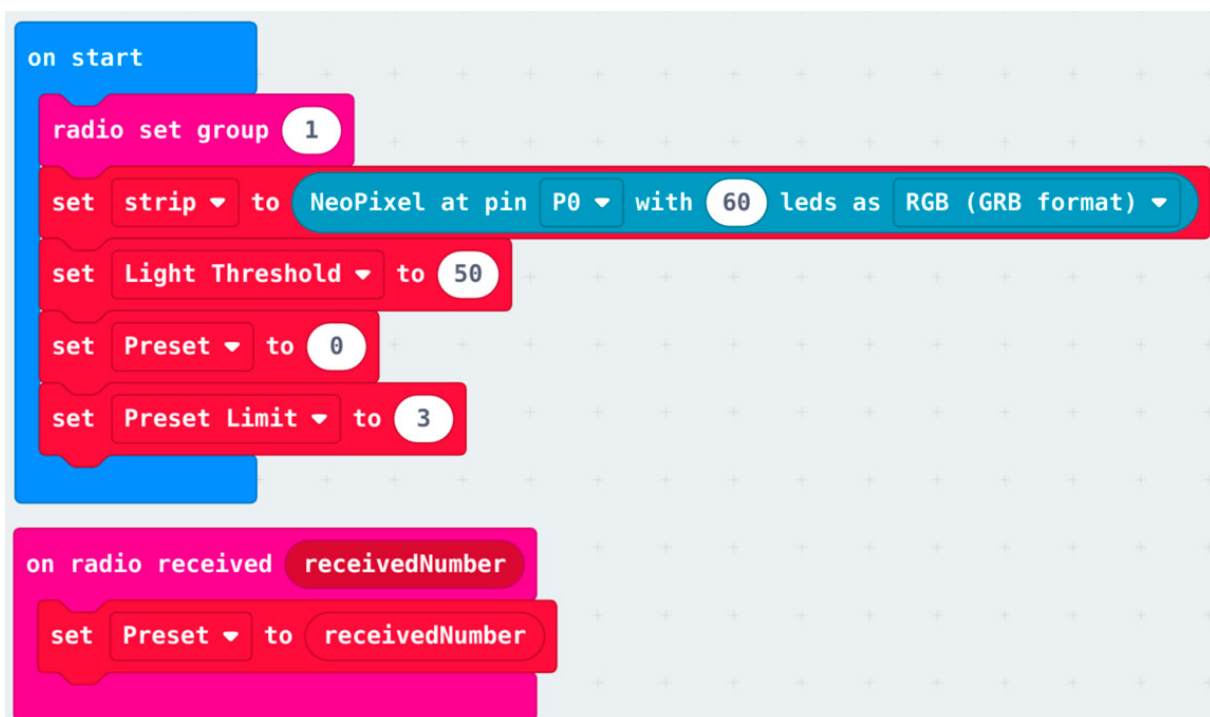
on button A pressed
  set Preset to remainder of Preset + 1 / Preset Limit
  radio send number Preset

on button B pressed
  if Preset > 0 then
    set Preset to remainder of Preset - 1 / Preset Limit
    radio send number Preset
  
```

Test it yourself in the following link: <https://makecode.microbit.org/S54410-58041-37683-04506>

Receiver micro:bit

Next, we need to program the receiver micro:bit to switch between effects using signals received through radio communication. To achieve this, simply place the receiver micro:bit within the same radio group as the emitter and instruct it to process the received radio signal with the on radio received block, which executes the code within its scope and makes the received signal available as a variable called “receivedNumber”. Use this variable to set the Preset value.



```
on start
  radio set group 1
  set strip to NeoPixel at pin P0 with 60 leds as RGB (GRB format)
  set Light Threshold to 50
  set Preset to 0
  set Preset Limit to 3

on radio received receivedNumber
  set Preset to receivedNumber
```

Except for the “on start” block, and the fact that the “on button pressed” blocks were replaced by the “on radio received” block, the rest of the code (“function” blocks and “forever” block) should look exactly the same as the preset system code without the radio functionality.

Test the whole thing here: <https://makecode.microbit.org/S04955-29546-19692-87374>

LeDS toolkit effects

Based on everything we have taught about programming LED effects, our team has developed its own Preset system with radio capabilities and various effects. Feel free to explore it and adapt it as you see fit.

Emitter micro:bit: <https://makecode.microbit.org/04838-45924-65091-31538>

Receiver micro:bit: <https://makecode.microbit.org/84429-78340-99227-35817>

Wearables

Integrating electronics into clothing or wearables is becoming ever more common nowadays. Wearables are accessories you can dress and use in your performance to create the effects you have programmed. Sparkfun for instance, has compiled [a set of tutorials available online](#) for anyone wanting to develop wearable electronics.

Everything you have learned until now can be integrated into wearables. This is a fun part of the process as you will have to choose the right materials and tools for your specific case, as well as to learn a whole new world of techniques, and competencies, such as soldering, gluing, sewing, wiring, etc. Stay open to ideas that bring you out of your comfort zone and push you to learn new things.

List of materials and tools

The LeDS wearable costumes were developed and designed with and for the students to use in their performances. For acrobatics however, as in the case of the Perfolie Arte students' performance, wearables are commonly subject to a great deal of mechanical stress when performers execute for instance aerials, floor movements or climb ropes, etc. So make sure you select the most suitable electronic components and materials for your specific case.

The costumes designed for the LeDS toolkit consist of the following components and materials:

- Lycra suit
- AAA Batteries or Li-Po batteries
- LED strip
- Micro:bit microcontroller
- Micro:bit shield, for holding the batteries and connecting the LED strips

- gaffer tape and grosgrain strip (portuguese prototype), or a light-colored fabric strip (greek prototype) for holding the led strips.
- wiring cable

Tools:

- sewing line and needle (or sewing machine)
- soldering iron and solder

Power consumption and Batteries

According to [NeoPixel — BBC micro:bit MicroPython 1.1.1 documentation](#) the maximum number of neopixels supplied directly from the 3V pin of the micro:bit should be no more than 8. However, in practice, it was demonstrated that it is feasible to draw significantly more power than the micro:bit's internal regulator provides, at least for a brief duration, without causing harm to the micro:bit. The Adafruit NeoPixel Überguide ([Powering NeoPixels | Adafruit NeoPixel Überguide](#)) recommends considering 20mA per LED for most applications. For the micro:bit board itself one should reserve roughly 30mA. Considering one micro:bit and 50 neopixel LEDs we can estimate the total current consumption to be 1030 mA, roughly 1A.

A typical 3.7V LiPo battery with 2000mAh capacity would supply the costumes for about 2 hours. 3 AAA batteries can yield a similar result. Both options allow for recharging but AAA cells can be directly replaced by standard AAA alkaline batteries widely. Using standard AAA batteries has advantages for stage costumes as they are readily available from specialty electronic stores as well as supermarkets or convenience stores. The [ring:bit shield for micro:bit](#) allows for the use of standard AAA batteries.

Putting everything together

A simple example of how to create a LeDS wearable circuit is to connect each strip input terminal directly to a digital terminal at the micro:bit. All strips' VCC and GND terminals must connect directly to 3V and GND terminals at the micro:bit. No micro:bit shield is necessary in this simple case, as a Li-Po battery can connect directly to the micro:bit. In this case it would be necessary to solder the LED strips' wires directly to the micro:bit board. An alternative to this, is to use a micro:bit shield, such as the ring:bit shield, for holding AAA batteries, and connecting the LED strips (without having to solder them).

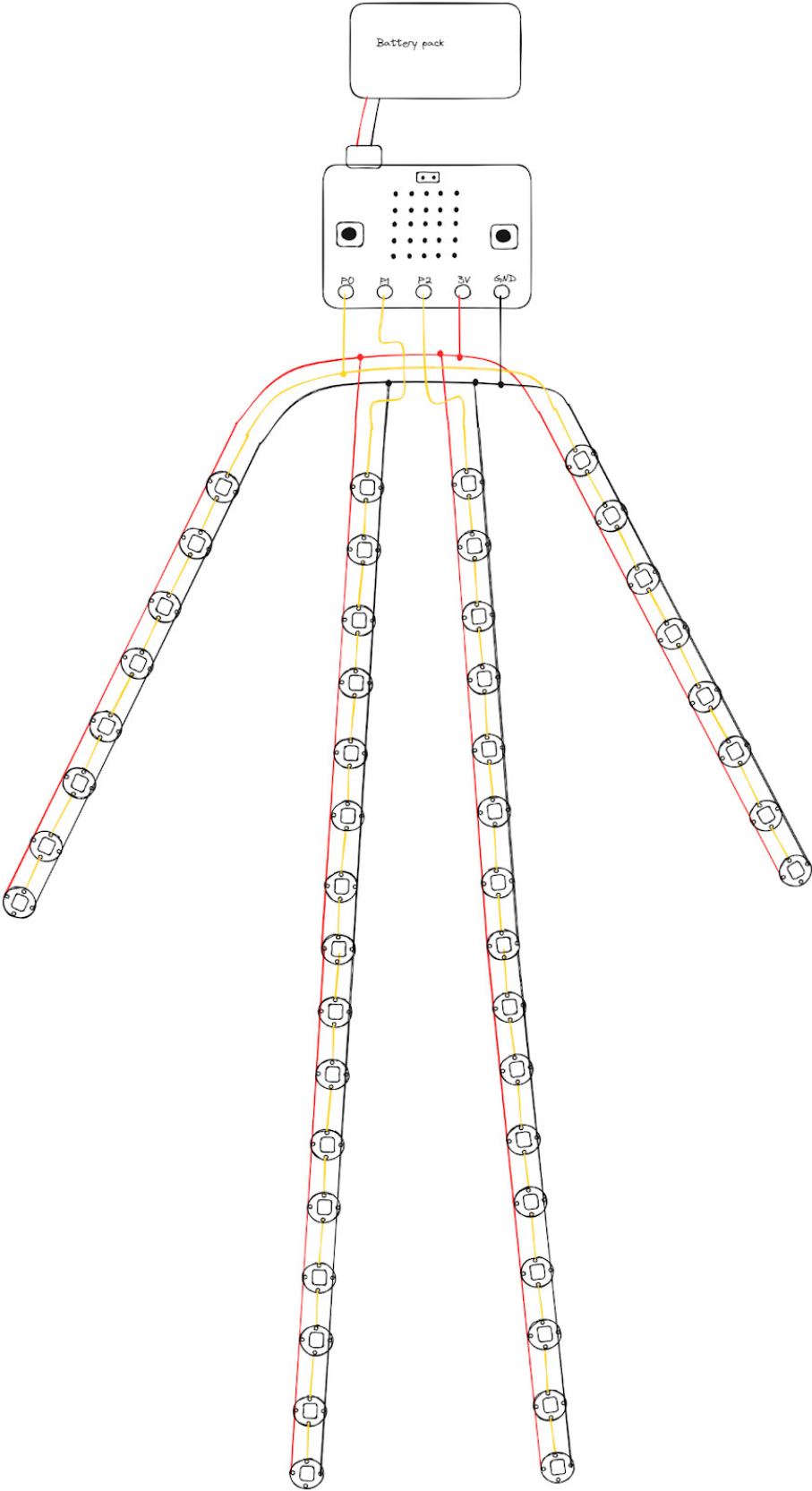
The costumes developed in Portugal are made of lycra fabric which is highly flexible but not suitable for sewing or gluing the LEDs to it. To allow the LED flexible strip to be attached, grosgrain strips were previously sewn along the arms, torso, and legs of the lycra suit. Over the grosgrain strips the flexible LED strips were fixed in place with hot glue and protected along the way with Gaffer Tape (Utility Duct Tape).

Another way to make the costumes, developed by the Greek team, is to use a special fabric strip and attach it to the costumes with the help of a special seam. The fabric chosen was light-coloured with a thin texture so that the light from the LEDs could pass through it. Two 2 cm thick strips of fabric were cut from it for each costume. The strips were then sewn left and right along the torso and limbs, (legs and arms).

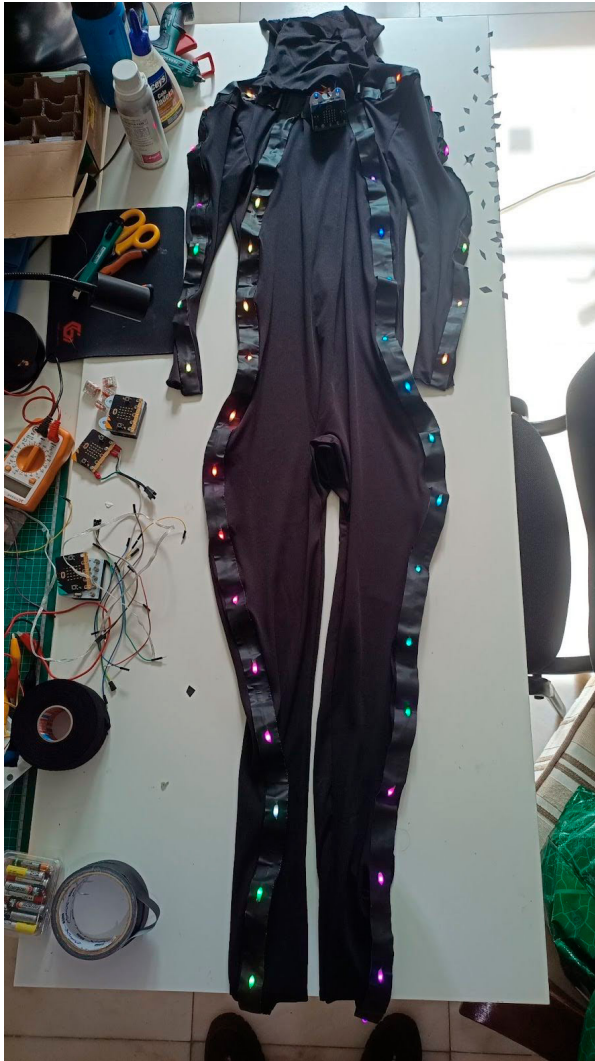
The LED strips attached to the suits were chosen to tolerate high traction forces and torsion, specifically, our kit used the [POMORONI 5m Flexible Addressable RGB LED Wire \(a.k.a. NeoPixel, WS2812B, SK6812\)](#). Regardless of the ones you choose, LED strips should be protected to avoid breaking the contact lines arriving or departing from each single LED. This can be achieved using hot glue to fix in place and protect each LED along the LED strip from extreme forces.

All the electronics, including the power source and the wire connections to power up and control the LED strips, were included in small lycra pockets located below the neck in the upper chest and specially designed to not interfere with the movements of the performers.





Wiring diagram. No micro:bit shield is necessary in this simple case. A Li-Po battery is connected directly to the micro:bit. Each strip input terminal connects directly to a digital terminal at the micro:bit. All strips' VCC and GND terminals connect directly to 3V and GND terminals at the micro:bit.



The costume prototype developed in Portugal.



The costume prototype developed in Greece.

Having put everything together, it is worth mentioning that even though the electronic components were chosen and tested to resist a significant amount of mechanical stress, some costumes failed to go through a complete rehearsal and performance and had to be repaired before the next one. Even though our method is good enough for small presentations and performances in schools (when they do not require a great deal of mechanical stress), it should be mentioned that further improvements are necessary to make the costumes suitable for any use case.

After testing the suits some LEDs may fail. This is due to the amount of mechanical stress applied to the suits during rehearsals or performances. Make sure you choose resistant LED strips to attach to your suits if you think your performance may damage the LED strips.

LeDS toolkit final remarks

You've made it this far! Now, equipped with all the cool stuff you've picked up, you're ready to dive into creating awesome effects for your performances and, most importantly, to spread the knowledge by teaching others.