Synthia coding - first steps

Introduction

Installation

Welcome to the Synthia coding tutorial

Thank you for supporting CircuitMess, and welcome to the Synthia coding tutorial.

We'll use **CircuitBlocks** for coding your newly-assembled digital music sampler.

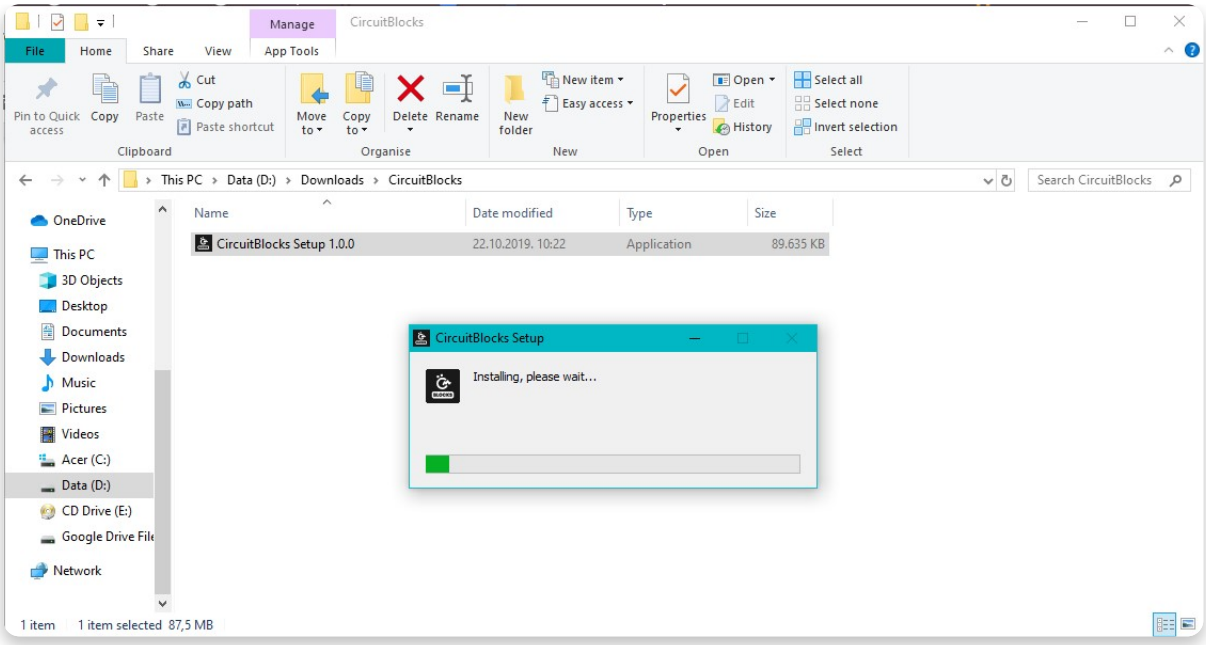
CircuitBlocks is a custom-made coding app that we've designed.
You will code your Synthia in CircuitBlocks' graphical block-based coding interface that will help you make your first steps in the world of physical computing.

Installation

CircuitBlocks currently runs on Windows, Linux, and Mac OS computers.

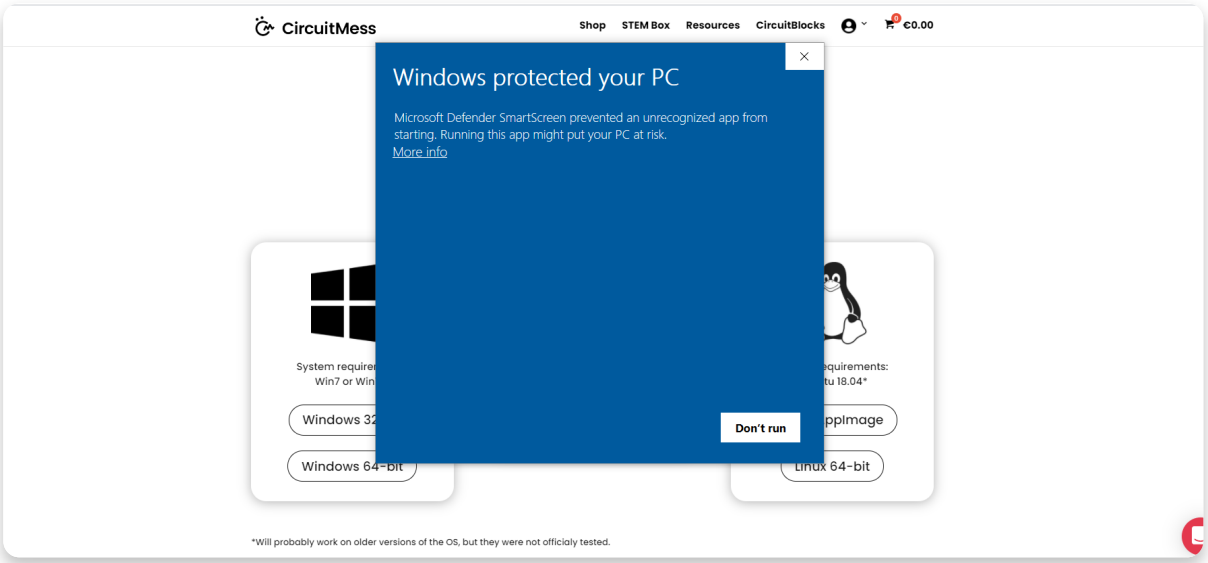
If you have a Windows computer

1. **Go to the CircuitBlocks download page**
2. **Download the latest version for Windows -** Check if you have a 32 or 64 version. Go to Settings on your PC, click on the System option and find the About section where you'll see the system type.
3. Double-click the downloaded file named "CircuitBlocks"
4. CircuitBlocks will automatically install and create a new desktop shortcut



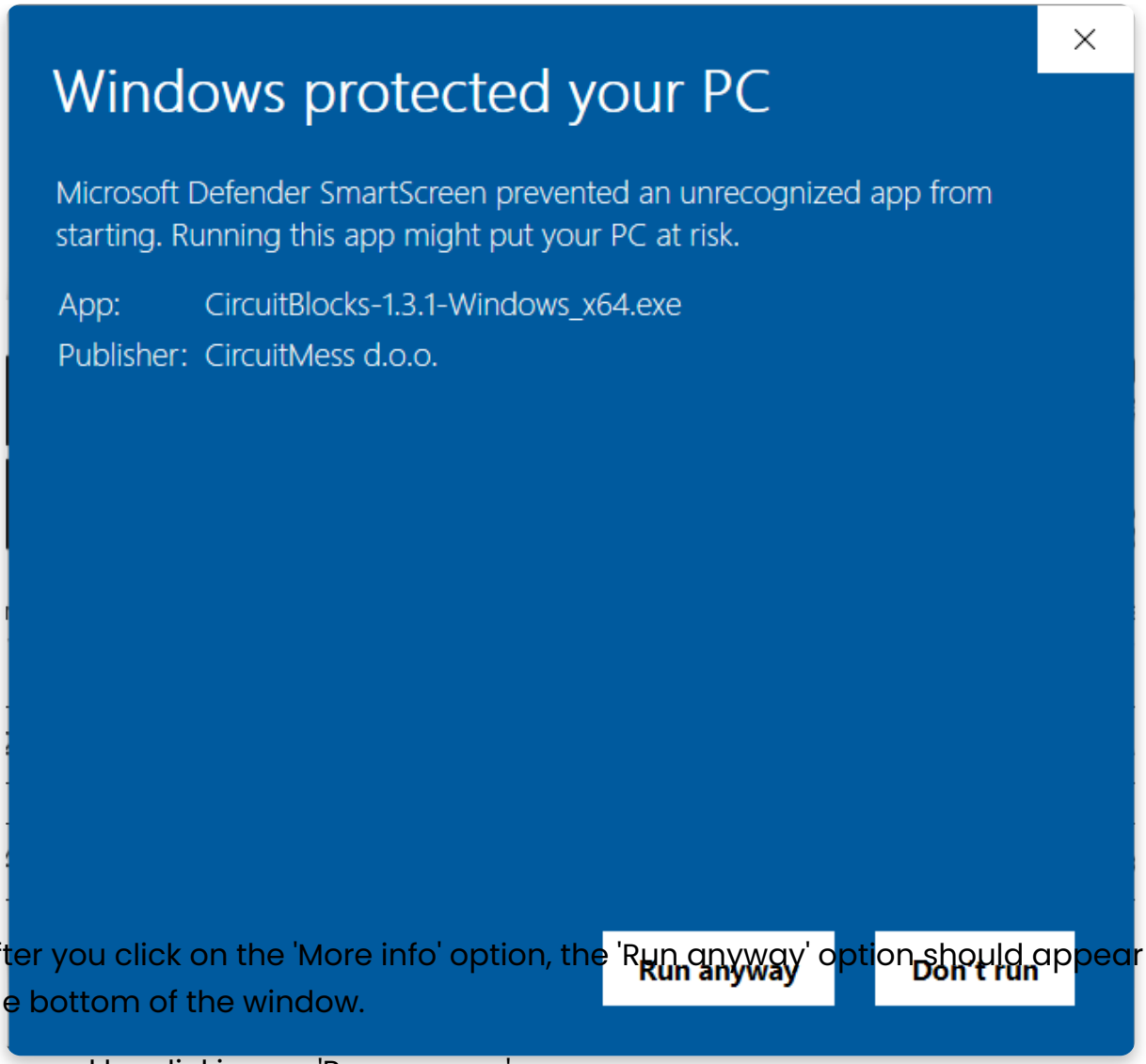
Your PC is not at risk!

There is a possibility that a notification that says your PC is at risk may pop up when you try to install CircuitBlocks. Don't worry; this happens regardless of CircuitBlocks being safe to run. See the instructions below on how to handle this notification.



This is the message you might get when installing CircuitBlock on your PC. Windows reports a threat despite the program being safe to download and run.

Please proceed with the installation by clicking on the 'More info' option.

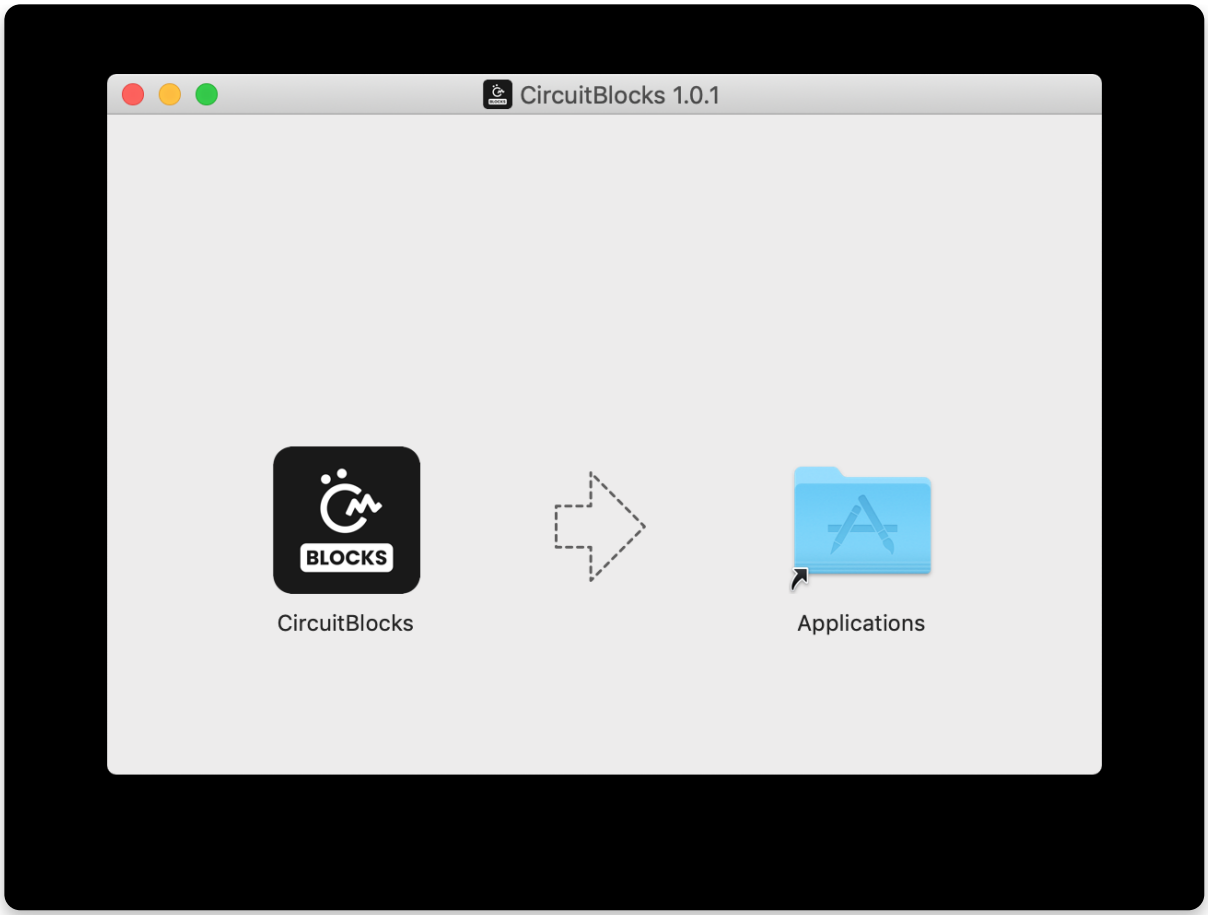


After you click on the 'More info' option, the 'Run anyway' option should appear at the bottom of the window.

Proceed by clicking on 'Run anyway'.

If you have a Mac computer

1. **Go to the** CircuitBlocks download page
2. **Download the latest version of CircuitBlocks** for Mac OS (the file named "CircuitBlocks-1.0.1-Mac.dmg" or similarly should be downloaded)
3. Move the files to the 'Applications' folder
4. CircuitBlocks will be installed automatically

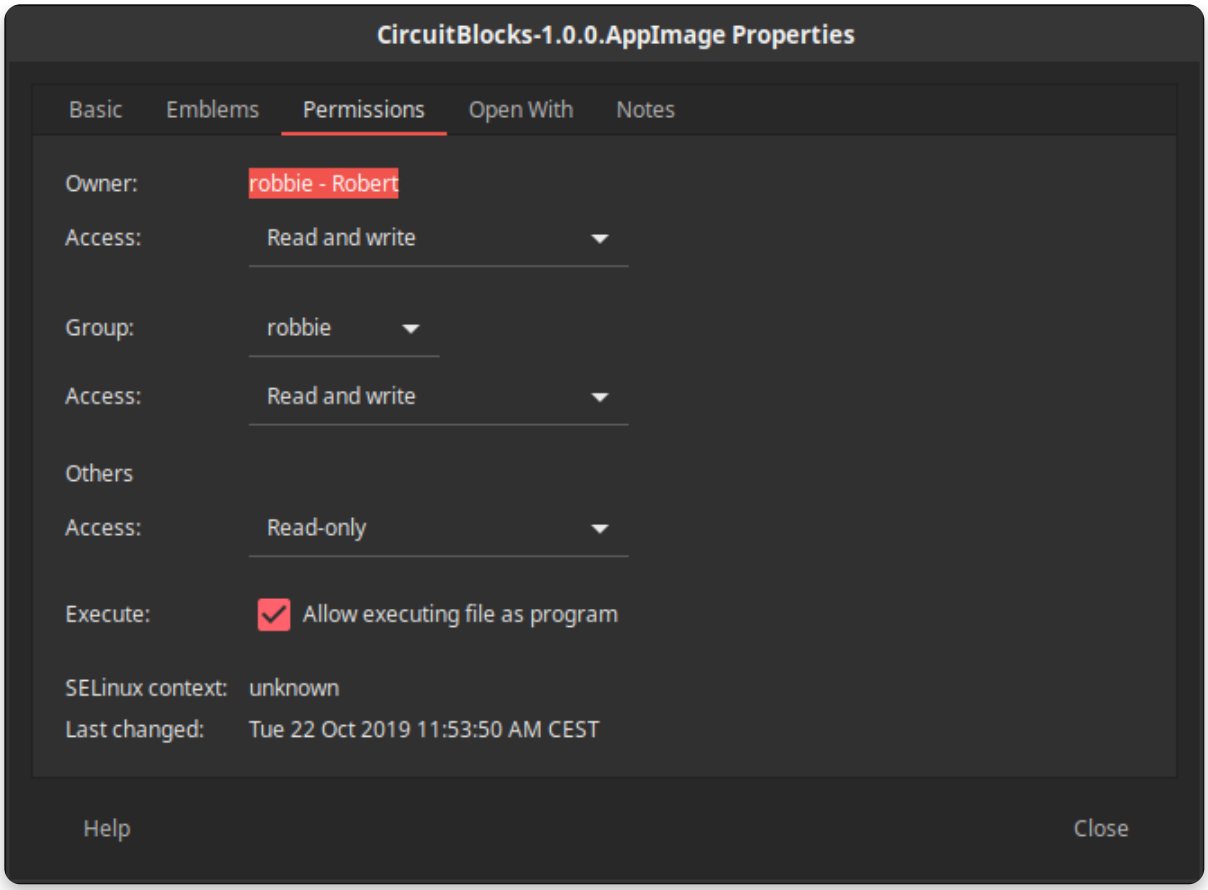


If you have a Linux computer

1. **Go to the [CircuitBlocks download page]**
2. **Press the "Linux 64-bit" download button**
3. Double-click the file to run the installation (Ubuntu)
   or
   Open the terminal and write `sudo dpkg -i <path to the downloaded file .deb>` (Other Linux distros)
4. CircuitBlocks will automatically install and create a desktop entry

Stand-alone (AppImage):

1. **Go to the [CircuitBlocks download page]**
2. **Press the "Linux AppImage" download button**
3. **Right-click on the file and select 'Properties**
4. Go to **'Permissions'** and tick **'Allow executing file as program'**
5. **Double-click the file, and the installation will complete automatically**



If you encounter any issues with the installation, please reach out to us via email at **contact@circuitmess.com** and provide a screenshot of your issue and any information you find relevant.

The basics

User interface

When you open CircuitBlocks, you will see a window that looks like this.

It's pretty simple – starting a **new project (we also call them "sketches")** can be done by clicking the 'New project' button.

Saved sketches will appear right next to that button and you can access them at any time.

If you encounter any kind of an issue with CircuitBlocks, press the **'Send error report'** at the bottom of the main screen. You'll get an error report number - please reach out to us via **contact@circuitmess.com** and provide your error report number.

Creating a new project (sketch)

Press on the big "New project" button.

You'll get an option to choose the device and sketch type.

For the device, pick **Synthia.**
For the Sketch type, choose **Block**.
Press the Create button.

You'll get a screen that looks like this:

On the top of the screen, there is a **toolbar** with a few buttons.

The **block selection bar** is located on the far left - you can take the blocks from there and drop them into the "drawing" area in the middle of the screen.

In the middle of the screen is where you'll be "drawing" your code with colorful blocks.

On the right side of the screen, you will see **code written in C++** appear magically by itself when you drag and drop the colorful blocks.

C++ is one of the most popular programming languages, but it's fairly complex to understand if you've never coded before.

That's why we've created CircuitBlocks - here, you can drag and drop colorful blocks that represent parts of code and see what your program would look like in C++. When you get skilled enough, you will be able to switch directly to textual coding in C++ without the need for colorful blocks.

Toolbar

Here's a short explanation of what the buttons in the window toolbar do:

1. **Back to the main menu** – returns you to the home screen without saving
2. **Save/Save As** – saves your sketch, make sure to press this button from time to time, and before closing CircuitBlocks
3. **Synthia connection indicator** - the red dot turns green if your Synthia is connected to your computer via a USB cable

4. **Export to binary** - saves a binary file of your code to your computer. This is a more advanced function that you won't need for now
5. **Serial monitor** - this button opens a window that we call the "Serial monitor". "Serial" is a nickname for a type of communication that is happening between Synthia and your computer. In this window, you will later be able to see the messages sent from Synthia to your computer via the USB port.
6. **Close code** - with this button, you can close or re-open the code window on the right of the screen. This is useful if you need more screen space for seeing your colored blocks.
7. **Run** - This button will translate the code you have constructed in CircuitBlocks to *machine code* that Synthia understands (beep boop beep boop 1011100101) and send the code to your Synthia via the USB port

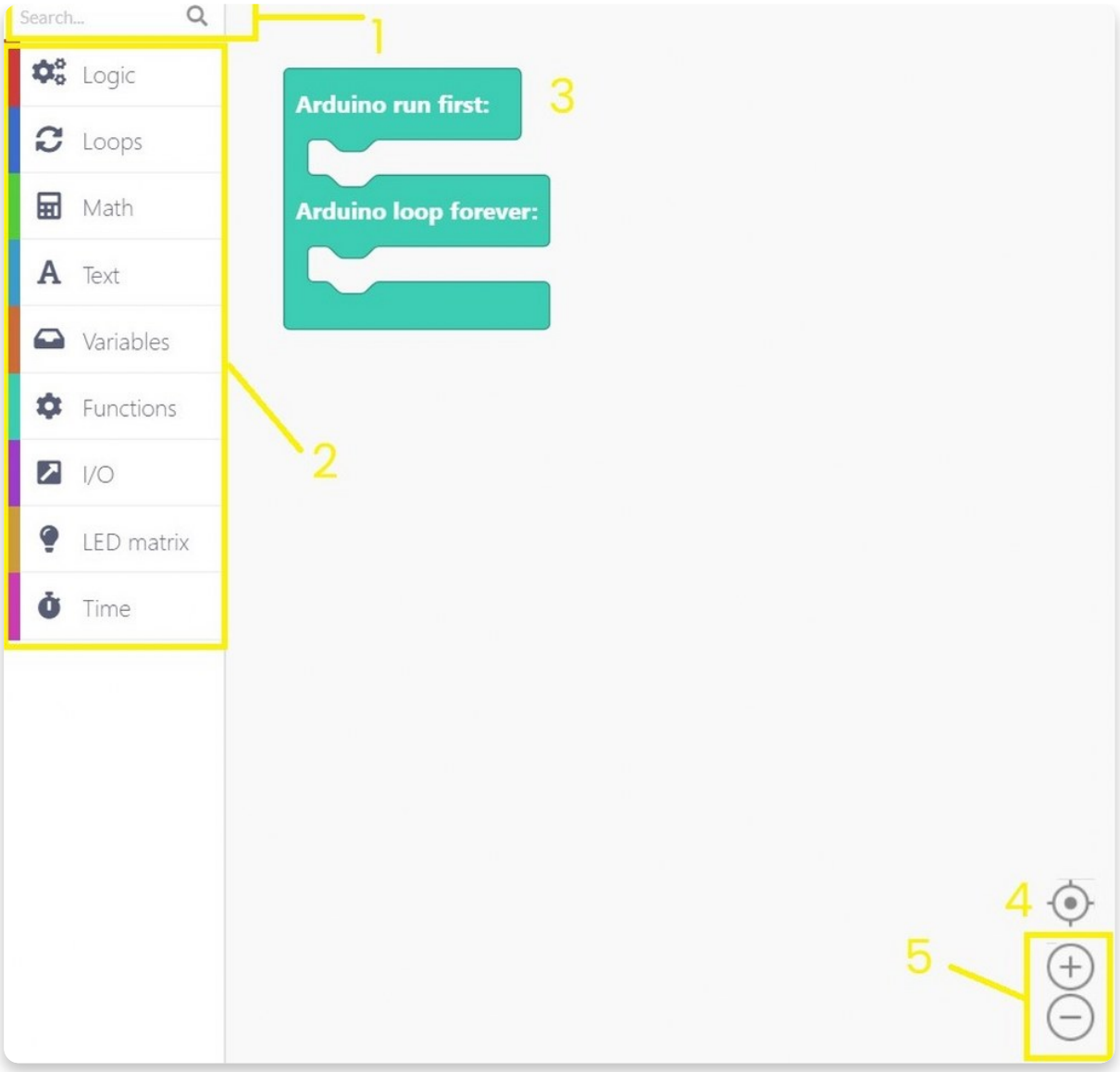
Code window

The so-called "Code window" has the following parts:

1. **Main code screen** - code written in C++ will appear here as you drag and drop colorful blocks on the left side of the screen.
   You'll see that some parts of the code are colored in funny colors.
   Programmers call this *syntax highlighting.* Basically, what is happening is that different categories of code commands are colored differently so that programmers can understand the code more easily.
2. **Light/dark theme switch** - you can toggle the background and text color of the code window with this button.
3. **Expand** - stretches the code window across the entire screen. Press it again to resize it to the half-screen again.
4. **Close** – closes the code window, the same functionality as the toolbar's 'Close Code' button.

Drawing board

The drawing board is where the magic happens.



It has the following parts:

1. **Search bar** – type the component's name you are looking for here.
2. **Component selector** – the blocks are divided into different categories here. Each category has a specific color designated to it.
3. **Drawing area** – you will drag the blocks from the component selector and drop them into the drawing area. This is how code is made. Easy peasy!
4. **Center tool** - if you get lost when scrolling across the drawing area, press this button, and it will center your view on the blocks you have dropped on the drawing area.
5. **Zoom buttons** - to zoom in and out of the drawing area.

Types of blocks

There are a total of **nine** block types in CB. Each of them is represented by their color. Every block translates to code, which is then compiled and uploaded to the phone, just like on every Arduino based platform.

Pressing on every block type will open a section from which you can drag and drop those blocks into the drawing area.

Also, pressing on **'More'** will open even more blocks that are not so commonly used.

There are two main functions of every Arduino code – **void setup()** and **void loop()**.

Everything that goes into the **void setup()** the function will run only once. It is primarily used to start the software, initialize and declare variables, and run functions that only have to run once (ex., Intro screen in a video game).

The **void loop()** is where everything else takes place. It basically runs every bit of code inside it repeatedly (speed depends on the device – just imagine it's ultra-fast!). It should pretty much follow the screen's refresh rate and make the program do things accordingly.

Every block you place automatically goes into the **void loop()** function.

If you wish to put something in the **void setup()**, you have to drag the main block from **Functions** and place your blocks inside as you wish, but more on that a little bit later.

Elliptical blocks

Elliptical blocks represent variables. Whether we're talking about integers, strings, or other variable types (other than Boolean), they can all be recognized by the same shape.

Also, larger blocks with elliptical shapes return either integer or float values.

> Whenever you find circular "holes" inside some blocks, you can insert variables. It's most commonly found in **comparison** or **action** blocks.

Triangular blocks

Triangular blocks represent boolean variables.

Both variables (true and false) and functions that return boolean values have the same shape.

Regardless of color, each of these blocks returns either true or false.

Triangular “holes” require boolean blocks to be inserted.

Building blocks

Everything else is basically a building block. Those are functions that have no return value (they return ***null***). Both elliptical and triangular blocks must first be placed inside the building blocks to act as part of the program.

They have a specific “puzzle” shape and can be stacked inside each other.

The main **building block** is located inside the **‘Functions’** section.

It basically gives you two main building blocks sections.

Everything placed inside Arduino runs **first** goes into **void setup()**, and everything placed inside **Arduino loop forever** goes into a **void loop()**.

Inserting blocks

Now, this is the main part.

The whole point of blocks-like IDE is connecting blocks and placing them inside another.

It is all done by simple **drag-and-drop** action.

Here is an example of a program that will set the variable **Var** to **1 and then increase that variable while it is smaller than 10**.

At the end of the program, **Var will be 10**.

This is just a simple example, and block-building will be further explained in the following chapters.

Block sections

There are a total of nine sections in CircuitBlocks. We've organized them so that you'll be able to find everything in a maximum of two clicks.

The sections themselves are pretty self-explanatory, but we'll go through them all to get a little better understanding of the whole concept.

Some of the sections also have additional blocks (in the **'More'** menu) where you'll be able to find some of the functions that are not used that often but can still be useful.

Logic

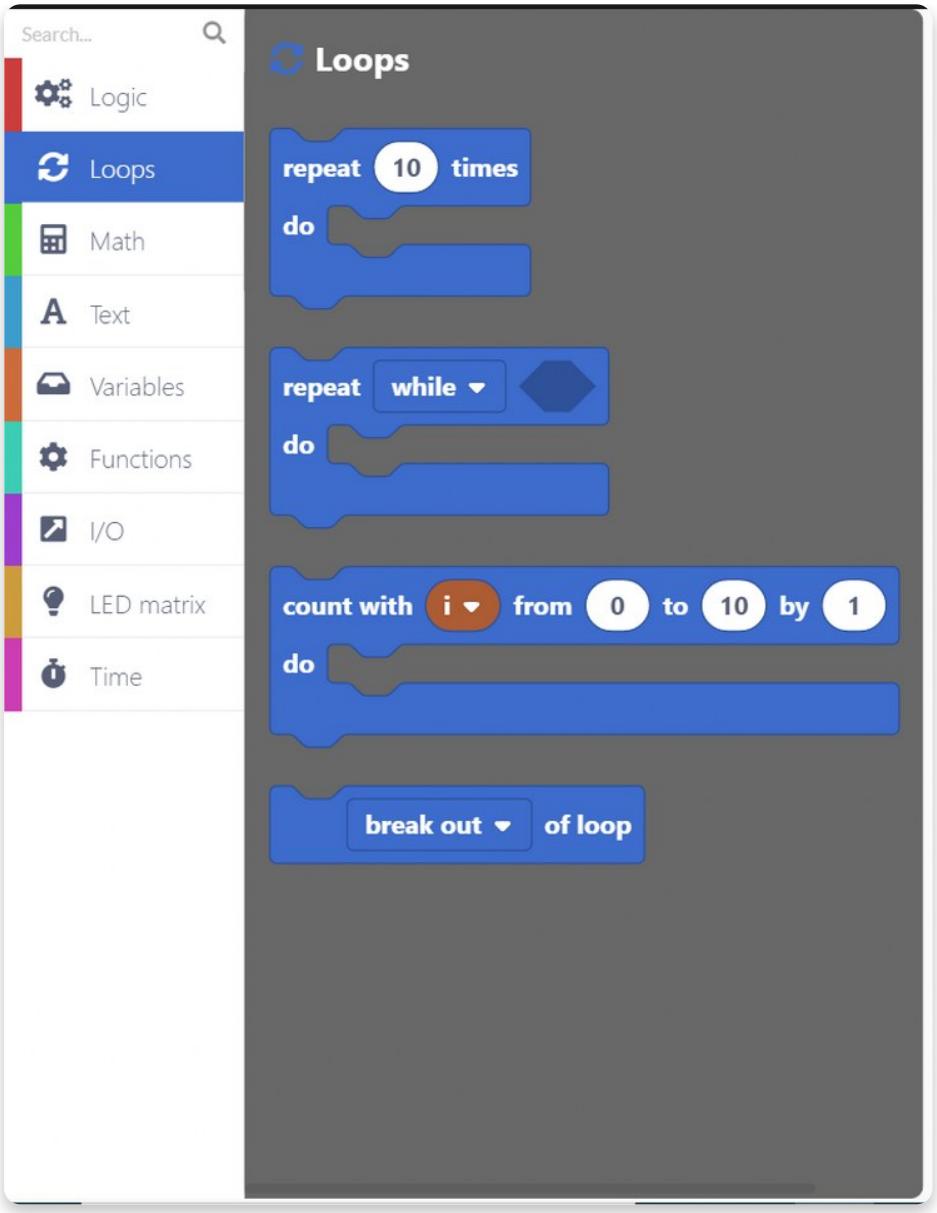
This is where the base of every code is located.

Every **if, if-else, else** function, comparisons, **and/or, not, true/false,** and other logical operators.

Loops

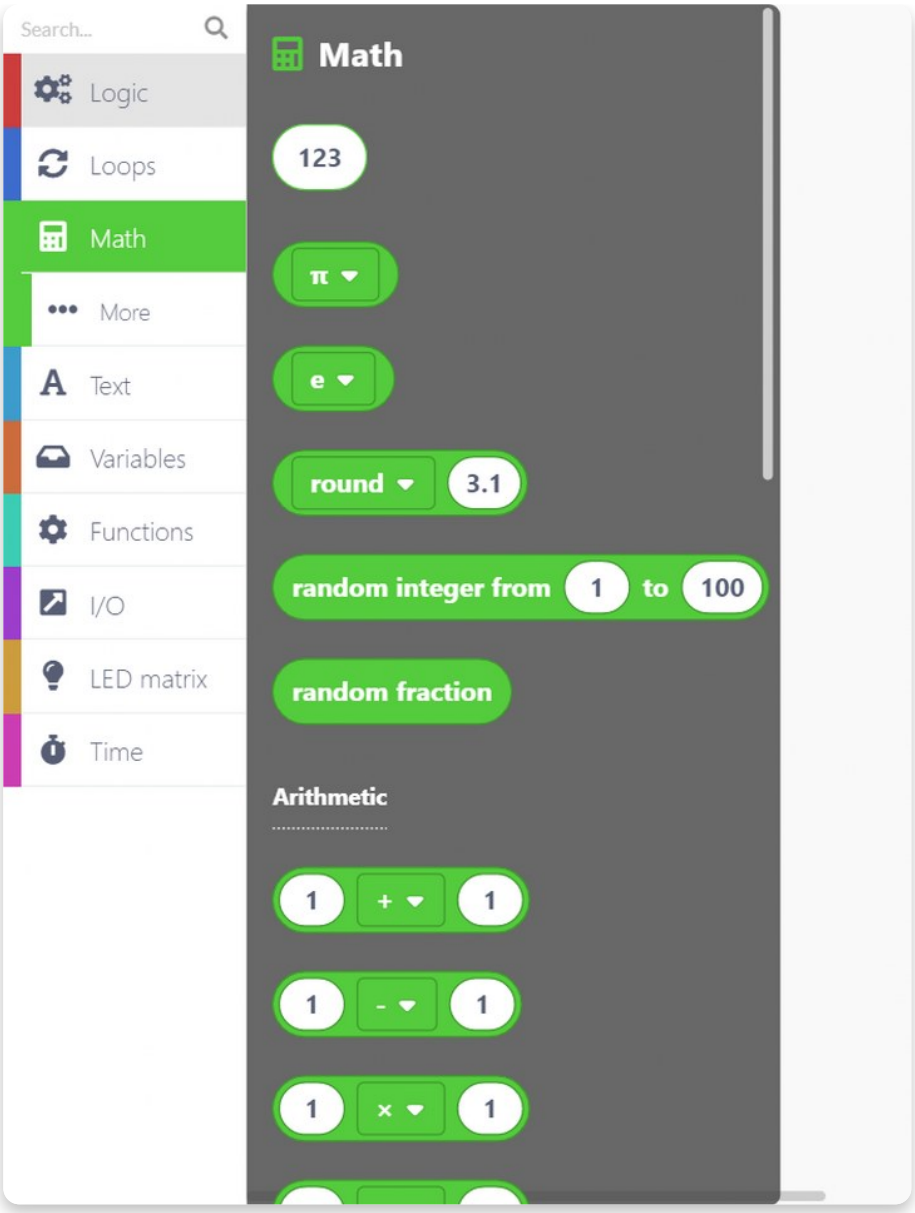
Loops are functions that repeat everything inside for a specific amount of time.

They can have conditional and repeat for as long as that condition is met or have a pre-determined number of repeats.



Math

Pretty much every math function is located here. From basic operations to rounding numbers and working with angles, you will easily trigger your inner Einstein or Pythagoras in a matter of seconds!



Text

Strings, characters, and string manipulation. Great place for creating new text and implementing it in your programs.

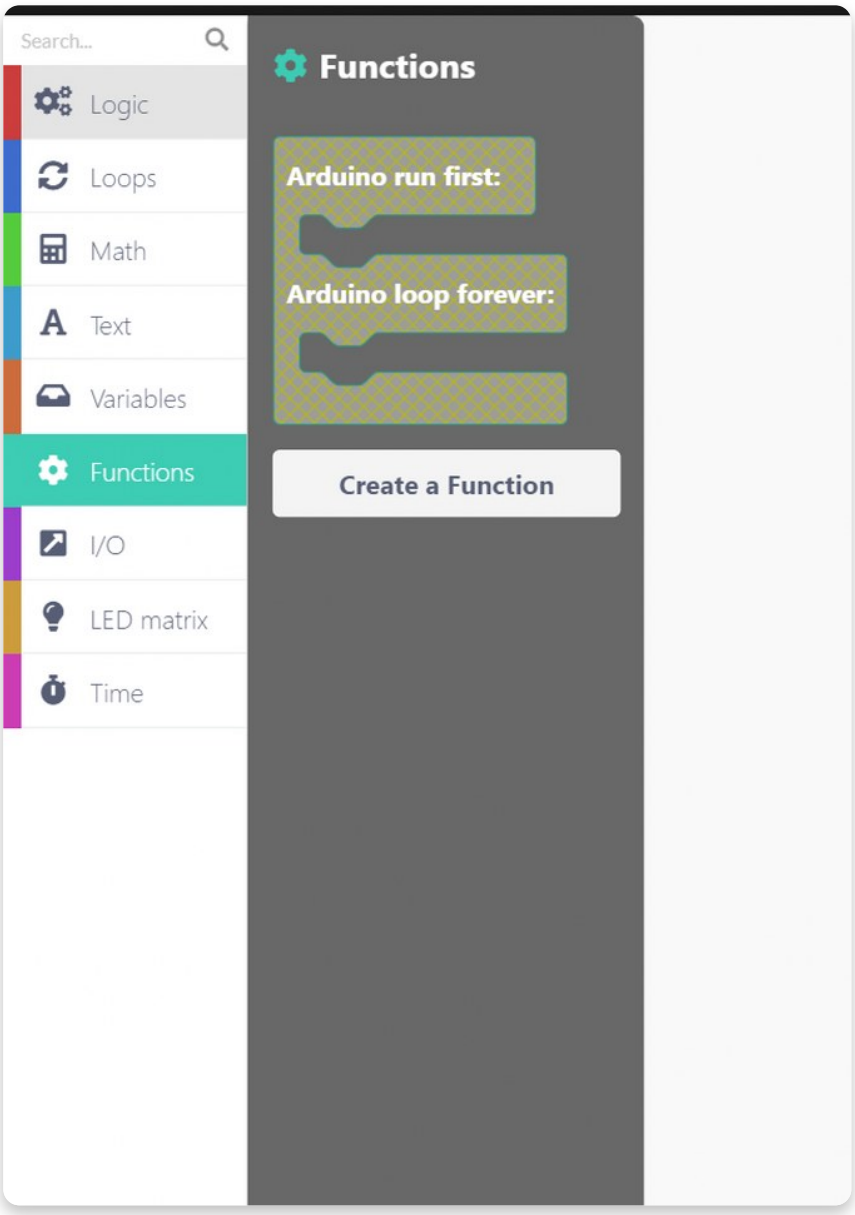
Variables

Create a variable of any type and set its name and desired value. CB will automatically recognize the variable type (int, double, string, boolean), so you don't need to worry about that.

Functions

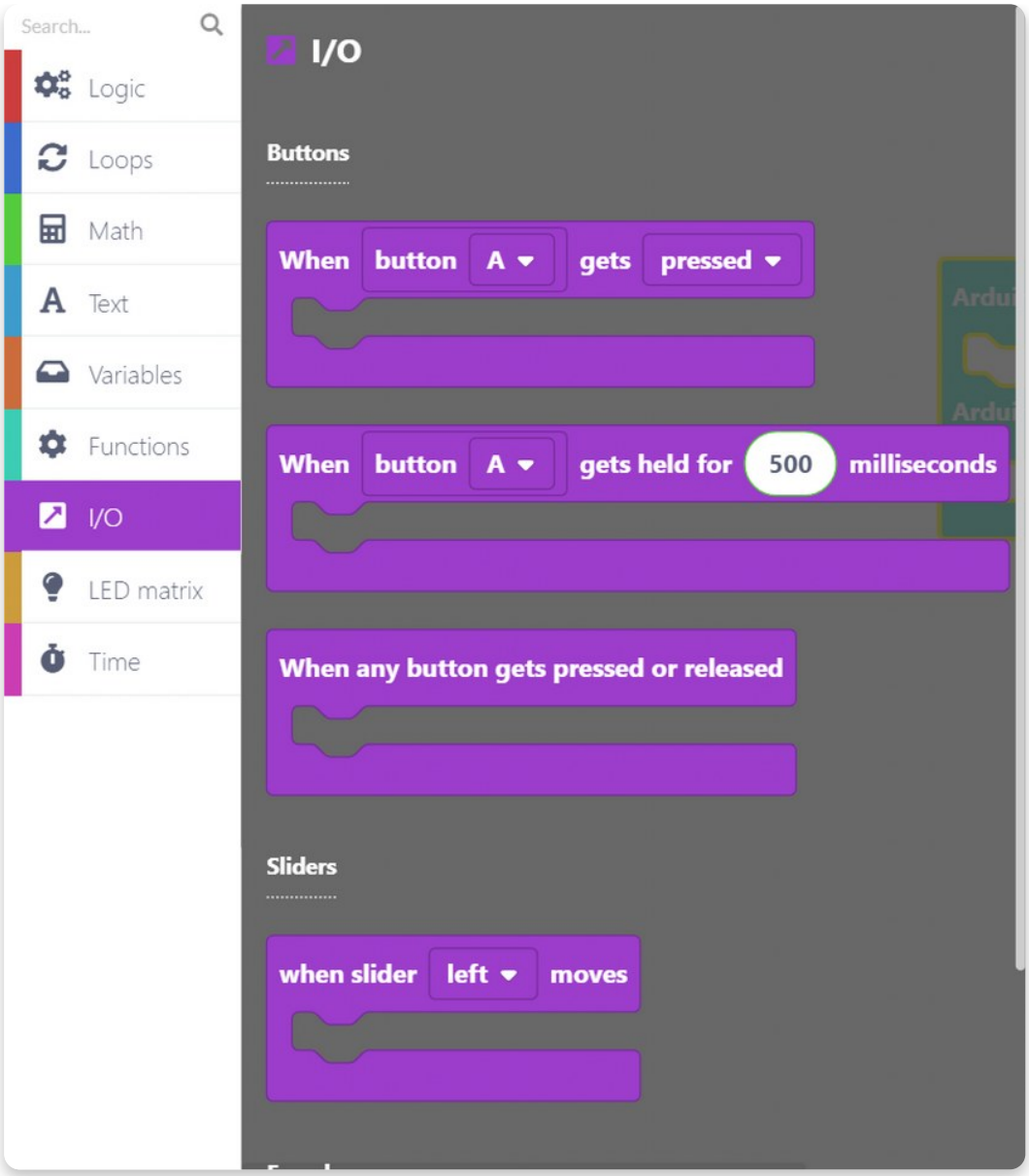
The Default Arduino function (explained on the previous page) is located here.

You can also create your functions which can then be inserted as one of the main parts of your program.



Input/Output

Everything regarding Synthia's components is located here.



LED matrix

Well, since we put that many LEDs on Synthia, it would be cool if you would be able to do something with them!

Here is where all the magic translates to those little LEDs on the front side of the PCB. You can create so much through these blocks.

Time

Delays, timers, and other time-related stuff, are great for creating cool animations and video games.

Search bar

There is also a **search bar** above all function sections to ease the search for that one specific block you just can’t seem to find.

Just type in whatever comes to your mind, and all blocks that have anything to do with the written word will be shown on the right-hand side.

Now, you really can’t say that it’s impossible to find something.

You’ve learned everything about the blocks!

It’s time to move on to the next lesson...

Let's start! Step by step

Play with LED matrix!

In front of you are three sketches that will introduce you to the coding world!

Connect Synthia with your PC, open CircuitBlocks, and follow these steps.

Usually, there are displays on our devices, but we decided to change it a bit and put an LED matrix on Synthia.

First things first!

Let us introduce to you a few important terms:

1. **Track monochrome** -> those are the LEDs in the middle of Synthia (the one with the largest number of LEDs)
2. **Cursor monochrome** -> the white row under the track monochrome
3. **Sliders monochrome** -> next to the sliders
4. **Track RGB** -> on the left and right side of the track monochrome
5. **Slot RGB** -> under the pushbuttons

Click on the **new sketch** in CircuitBlocks and choose **Synthia** since that is the device we'll be coding today.

This is what you'll be seeing once you enter the sketch:

As you can see, we have two sections in this main block - **"Arduino run first"** and **"Arduino loop forever"**. As their name says, the blocks you will put in **Arduino run the first** section will **run as soon as you turn on the device**, and the blocks from the **loop forever section** will run **afterward**.

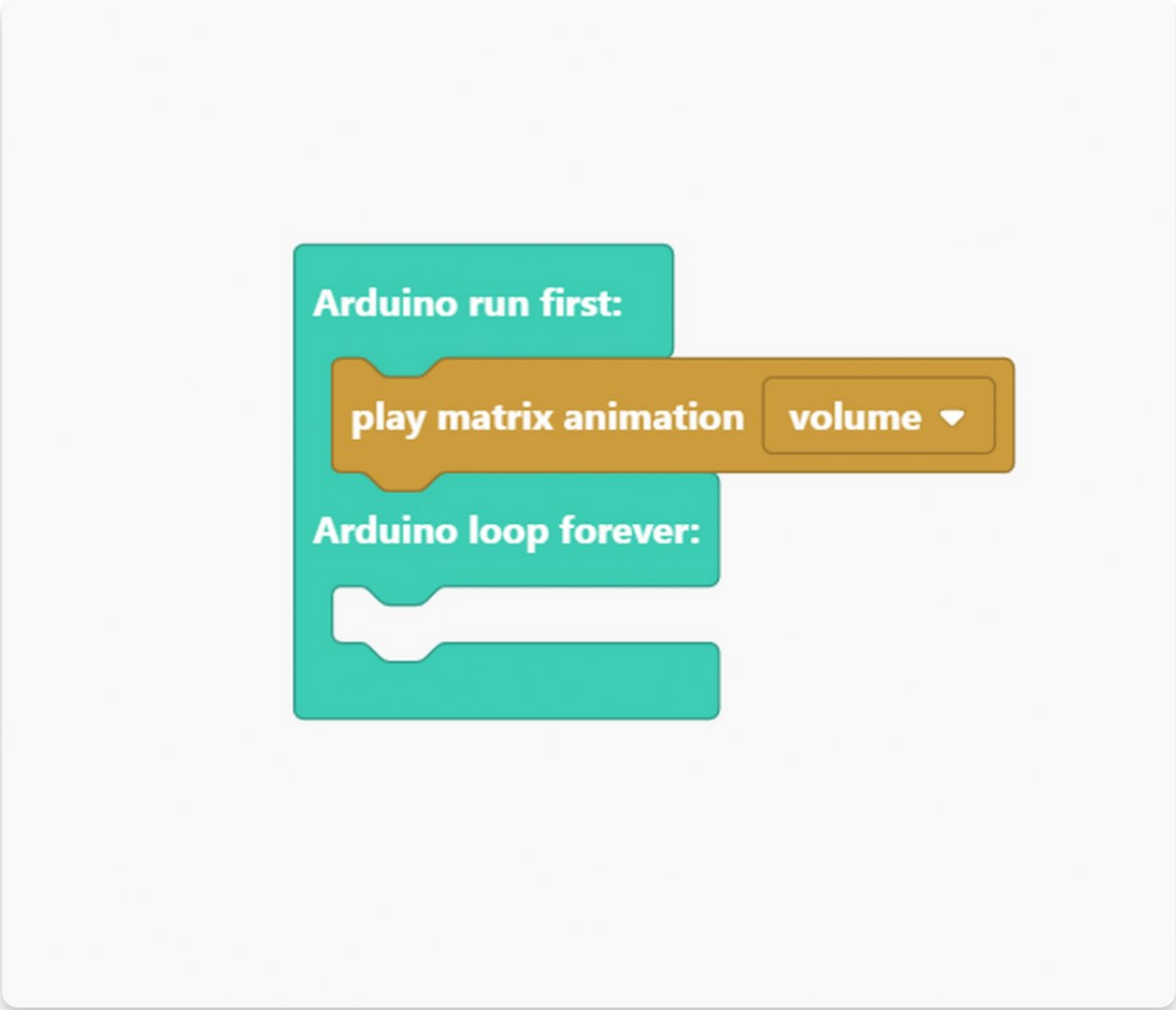
First, we'll check the **LED matrix block section** on the left side and look for the "**play matrix animation**" block.

This section contains a group of blocks that can be used for displaying animations and on Synthia's LED matrix.

Simply, click on it and drag it to the board.

We want to put the **"play matrix animation"** block here so that the animation starts playing only once when the device starts.

Change the pitch into volume!

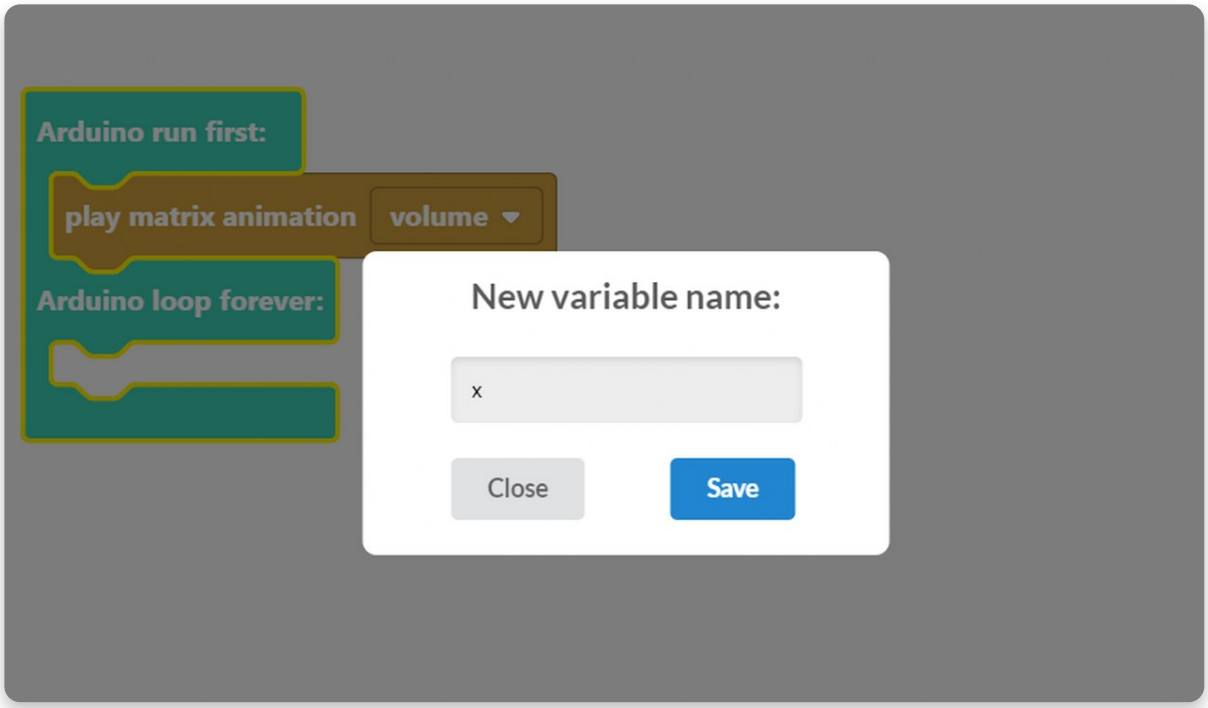


Now we'll have to create our first variable for this sketch. Let's call it "**x**".

In computer programming, a **variable** is a storage location that contains a value. Every variable has a specific name. You can store and change the value of a variable.

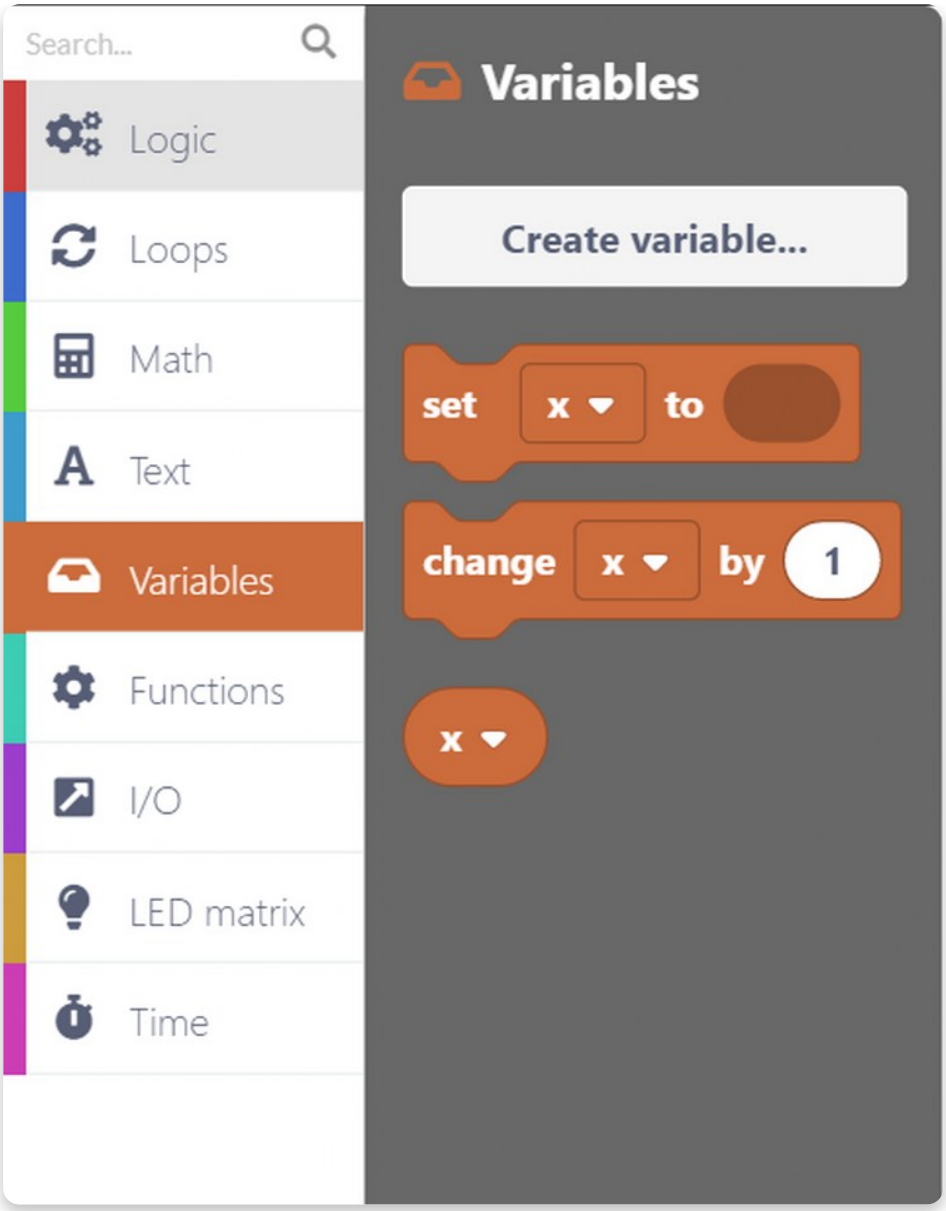
Firstly, let's create a variable. Find the section named "**Variables**" and press the "**Create variable...**" button.

You'll get a pop-up window like this one:



Save it and you're good to go!

Right now, you'll have two new blocks in the variable section called "**Set x to**" and "**Change x by**". For now, we need the "**Set x to**" block.



When we create a variable, it's undefined - **it has no value**. We must set a value for every variable when our computer program starts. That's why you'll need the "**set variable**" block.

There is a block in which you can write any numerical value, and it is located in the **Math block section**.

Set the "**x**" variable to **0**.

Why to 0 you may ask?

Variable "**x**" holds the value of the ordinal number of the LED that is currently lightning up.

Since there are **5 RGB LEDs** under each pushbutton, and the PCs are counting them from 0, we'll have to put **0** for the value of the **first LED**.

Now is the time to make the first function.

Just like with the variable, you can create a function by clicking on the block, saying create a function, and a pop-up window will appear.

You can give any name you want to your function, but we named it "**drawPixel**".

Once you click on the done button, you'll get a new blueish block on the drawing area. Leave that part for later and use "**call drawPixel**" right now.

What this function will do is light up the LED that the variable "**x**" is showing.

Firstly, the "**drawPixel**" function will turn off all the LEDs, change the current pixel to cyan, and push the changes on the matrix.

Let's create another variable and call it "**pixelTime**".

We need that variable in order to track the time that passed since the last LED lightened up.

You already know how to do that, right?

So, click on the **Variables block section**, and then on the create variable option.

Write down a name for it.

Drag and drop "**Set variable to**" block under the "**call drawPixel**" block.
Just like in the photo below:

You can see that once again, we don't have the value of the variable, so we have to fix that.

This time, we won't give any numerical value to the variable.

Go to the **Time block section** and choose the "**current elapsed Time (milliseconds)**".

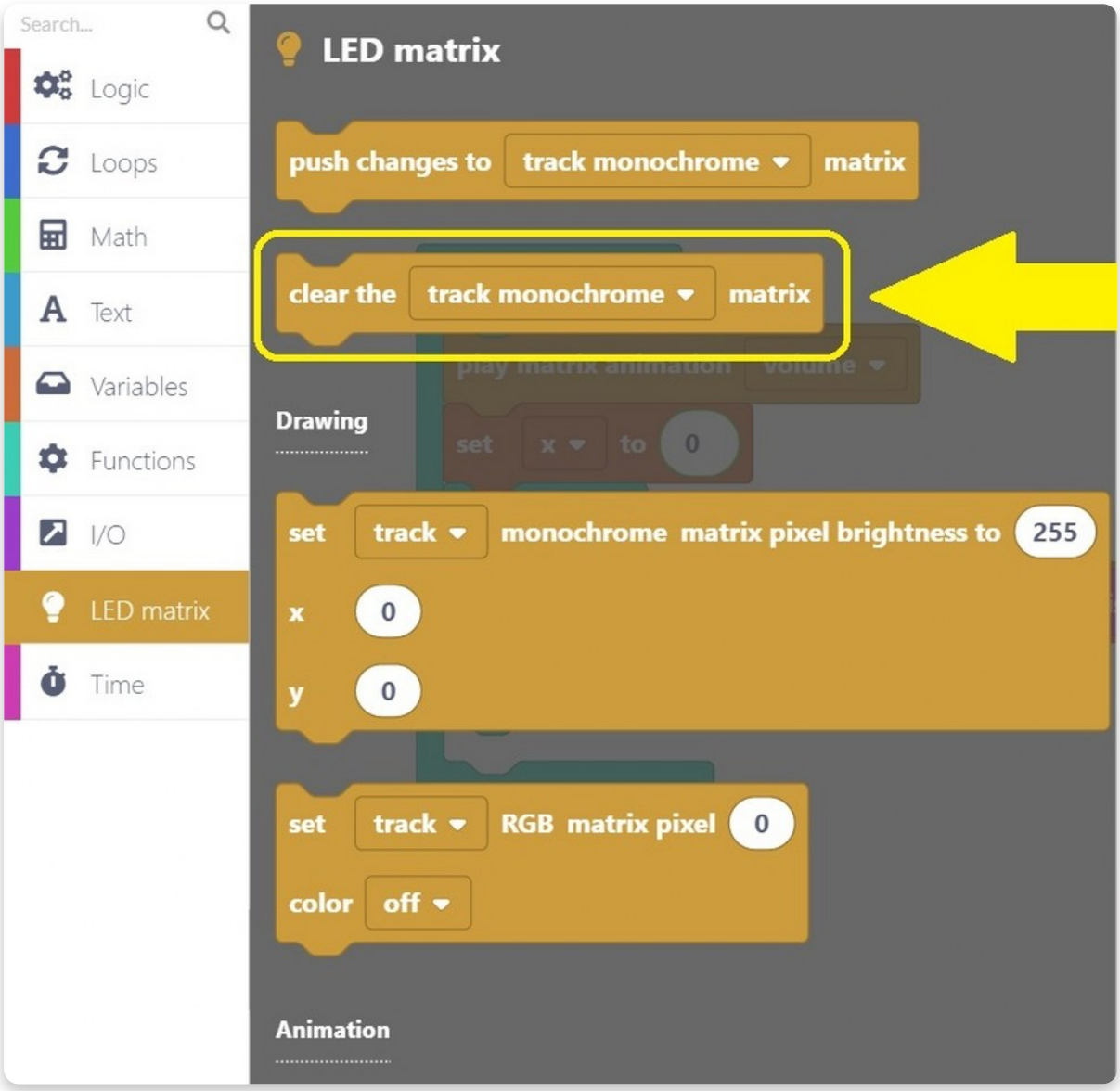
Place that block on the place of a circle in the variable block.

Now is time to use the "**drawPixel**" function block.

This block will determine what will be shown on the LED matrix.

The first thing you want to do is to **clear the entire matrix.**

For that, you'll need this block saying "**clear the track monochrome matrix**".



After dropping it in the "**drawPixel**" block, we'll change the "**track monochrome**" to "**slot RGB**".

The next block we'll need is also from the **LED matrix block** section.
Search for this particular block:

Change the track into a **slot**, since we are working on a slot right now, and put the color **cyan** (or any other that you want to).

You'll also have to change **0** to a variable "**x**".

The final product should look like this:

We are still in the LED matrix block section.

Search for this block saying "**push changes to track monochrome matrix**".

Since we are working on slot RGB, we have to change the "**track monochrome**" to "**slot RGB**".

Once again, we are going to use the "**clear the matrix**" block. You can either make it the same way as we did for the first such block, or you can simply **copy** the one we already have.

For copying, you just click on the right button on your mouse and choose **duplicate**. You can use that button to **delete** some blocks as well.

Now, we will be working on a track RGB.

You can copy the next two blocks called "**set RGB matrix pixel color**" and "**push changes to matrix**" as well.

The only thing you'll change is "**slot RGB**" into "**track RGB**".

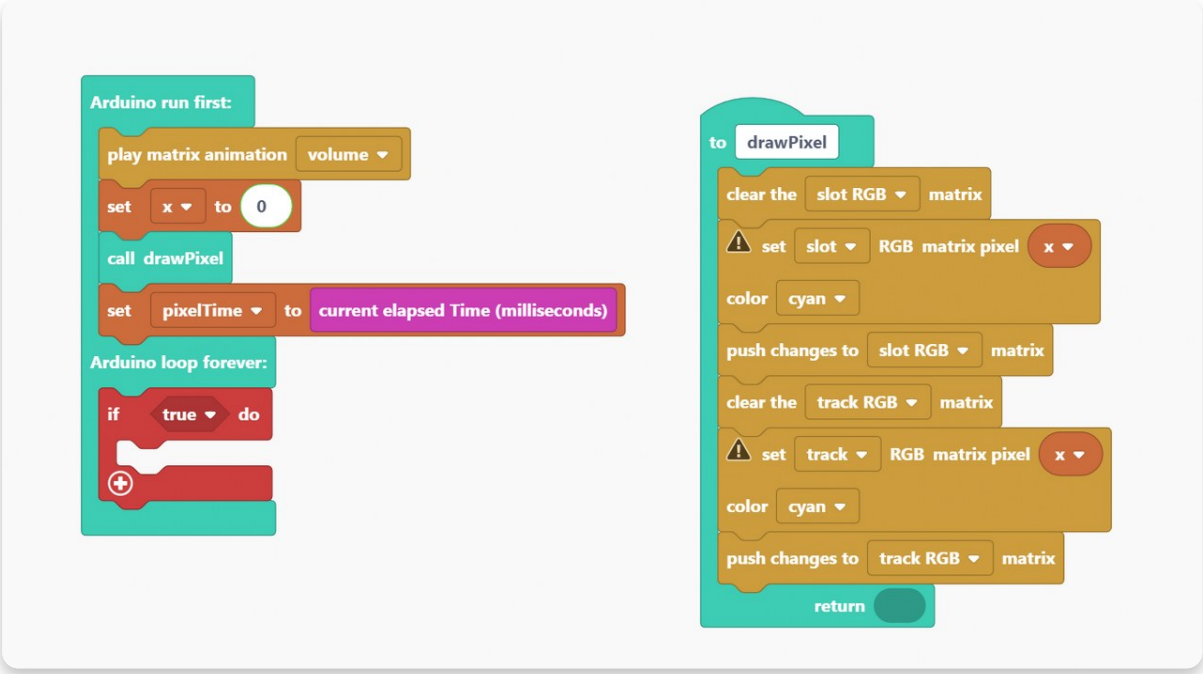
This is what the drawing area should look like by now:

We successfully finished everything in the "**drawPixel**" function.

The only thing missing is the **Arduino loop forever** part.

Let's finish that as well!

The **logic block section** is the next section we'll use.



We'll replace the "**true**" part of the block with this block:

This is a comparison function, so we'll need to use the math block instead of the left number.

After making some changes, this is what your drawing block should look like:

We will change the variable "**x**" by 1.

This change means that every 500 milliseconds "**x**" will change by 1 (up to 4).

Once again, we are in a need of an "**if - do**" block.

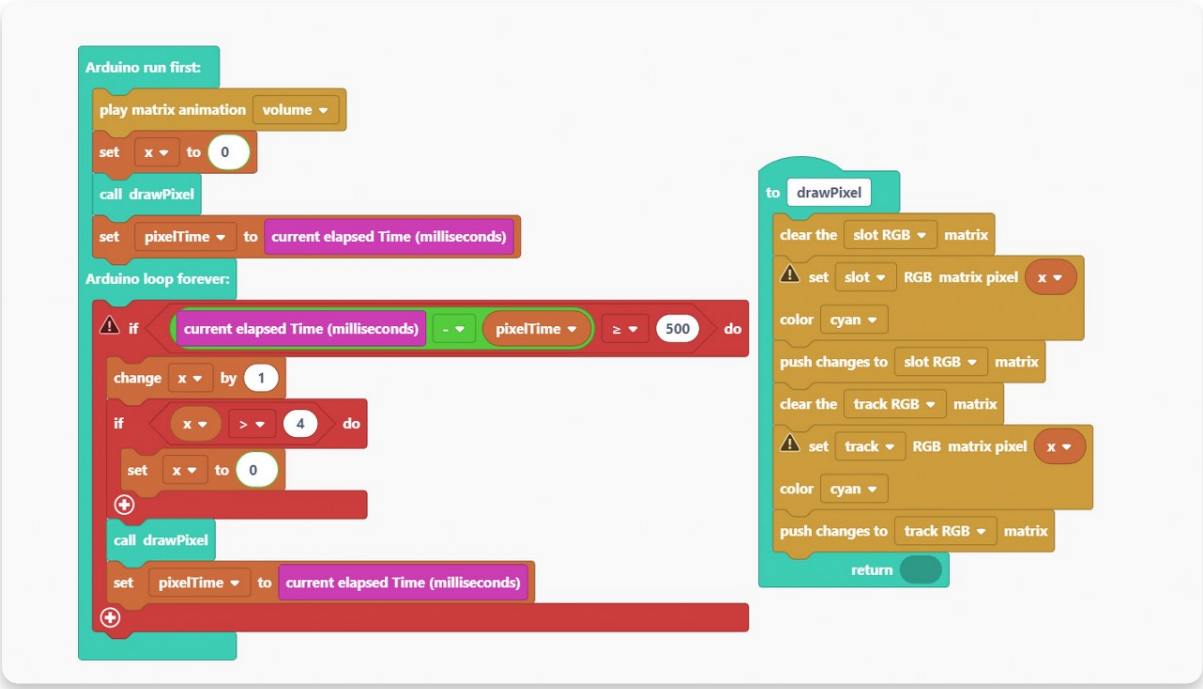
Take another **comparison block** and place it instead of the "true" part.

What we did here is we determined that the **variable "x" is set to 0**.

Every **500 milliseconds the variable "x" will change by 1.** This will go up to 4 since there are 5 RGB LEDs in the row.

Once "x" get to 4, it will go back to 0.

Now we have to call the "**drawPixel**" function to light the LED and save the current time.



Now that you finished your sketch and you understand what is going on, hit the big red **Run** button and wait for the code to **compile!**

If you're doing this for the first time, the code can take up to a minute to compile. But, don't worry, after that, the compiling should be faster.

Here are a few photos showing what it should look like once you compile the code.

So this is how Synthia will look once you turn it on. The first LED under the pushbutton will light up (the one we set as 0).

Every 500 milliseconds, the light will switch to another LED.

Also, the animation will be playing on the LED matrix this whole time.

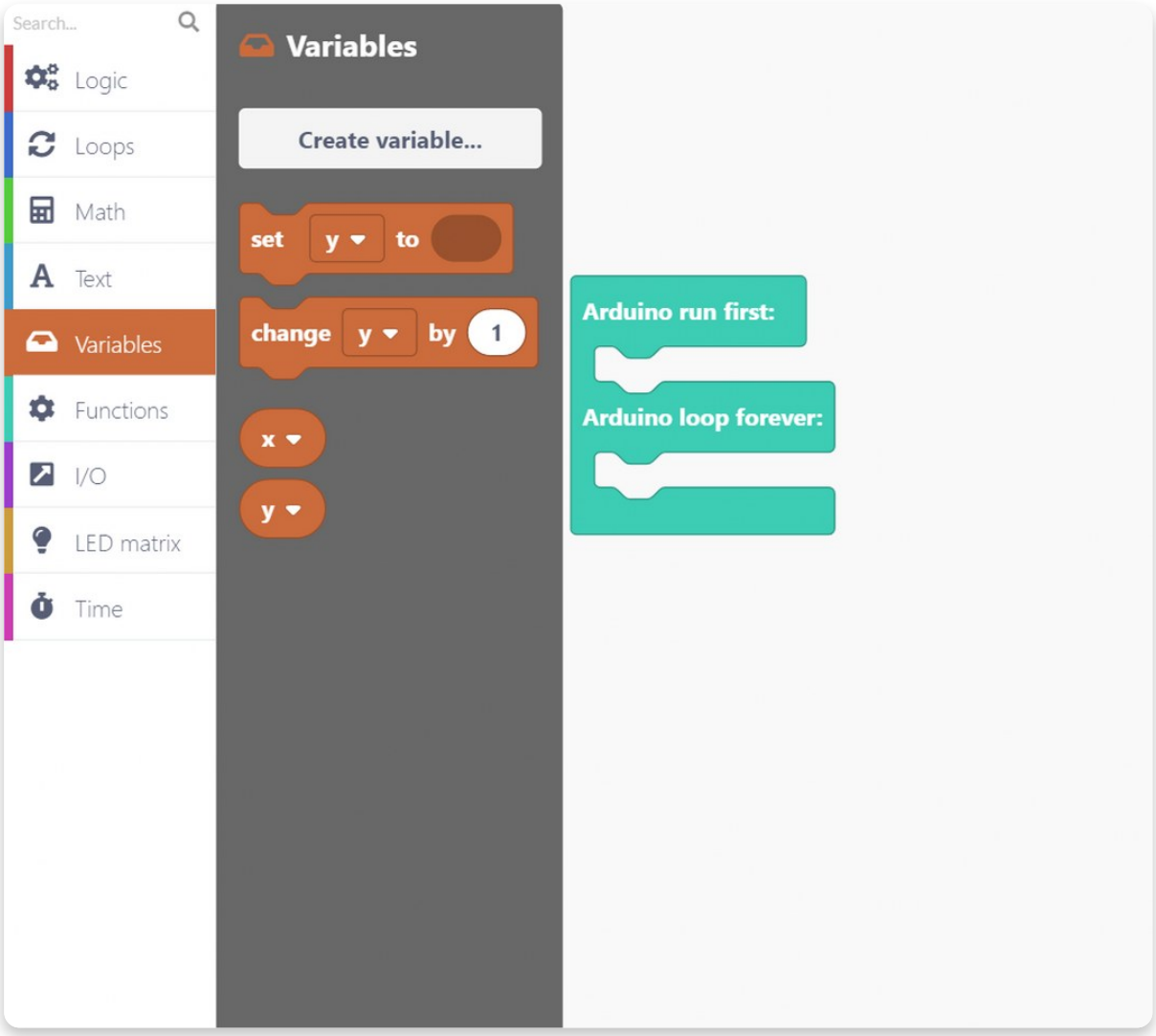
Click, slide, and rotate!

Welcome to the **second example** in Synthia's coding guide.

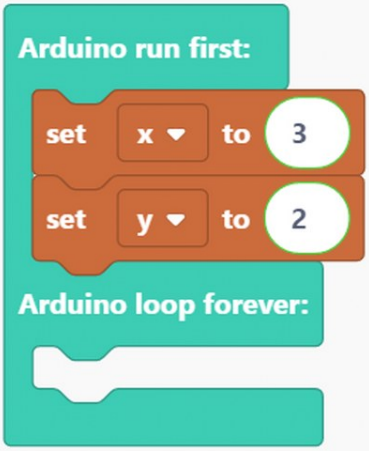
In this one, we'll use all the accessible inputs and play with them for a bit.

Inputs on Synthia are **pushbuttons, sliders, and encoders** - so, the parts you have soldered!

The first thing we'll do in this sketch is **create two variables**, "x", and "y", that will represent the axis.



Take the "**set to**" blocks for both variables and stick them in the **Arduino run first** section.



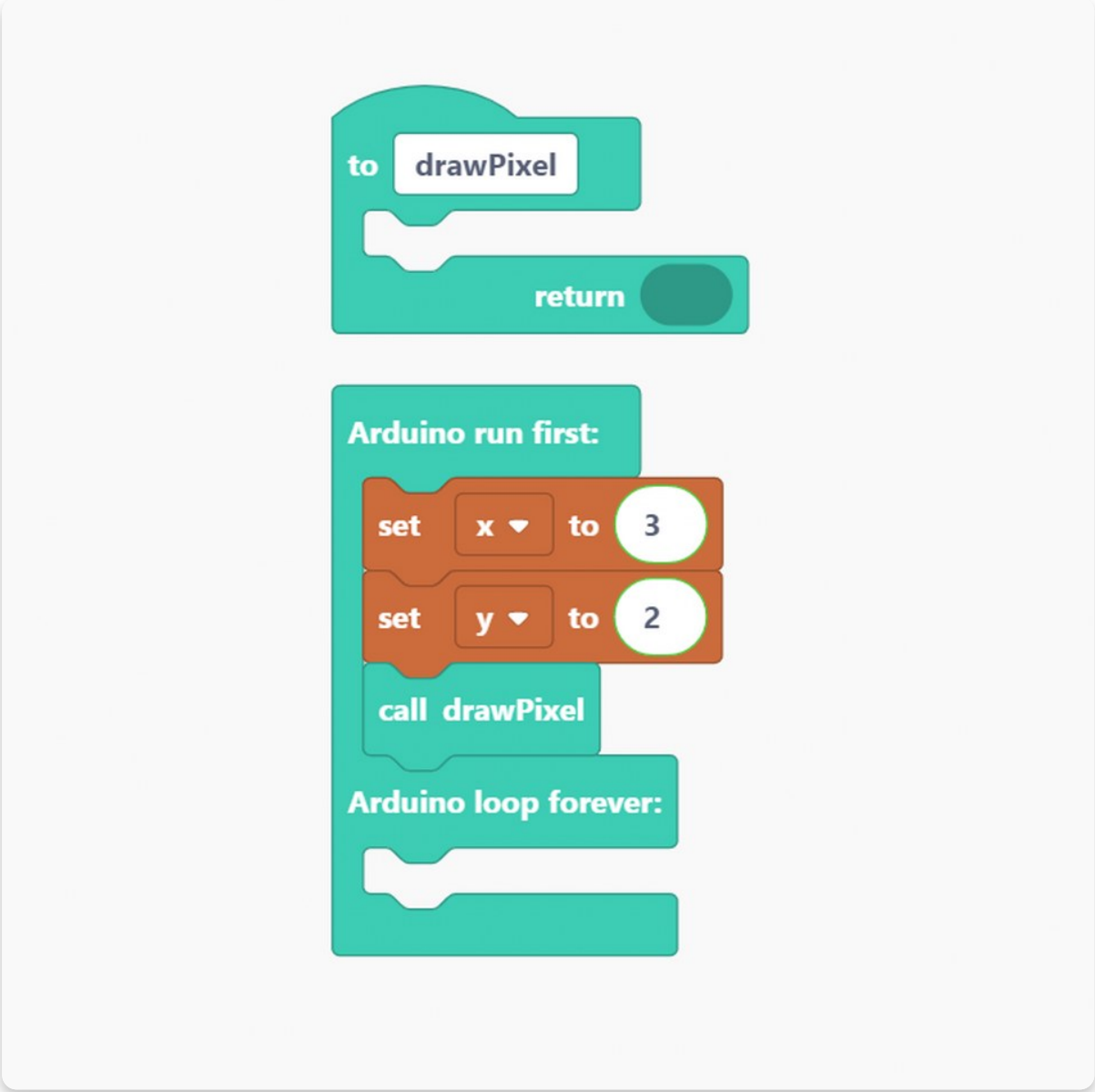
You can put the variables on any number you want, but we choose these numbers so the starting point is not on edge.

So, these numbers signify **the position of the LED that lights up on the main matrix once we turn on the device**.

Now is the time for another thing you know how to do: **create the function**.

We'll call it "**drawPixel,**" as in the last sketch.

Take the "**call drawPixel**" block and place it under the variables.



With this function, we'll light up the LED that is determined by the variables "**x**" and "**y**".

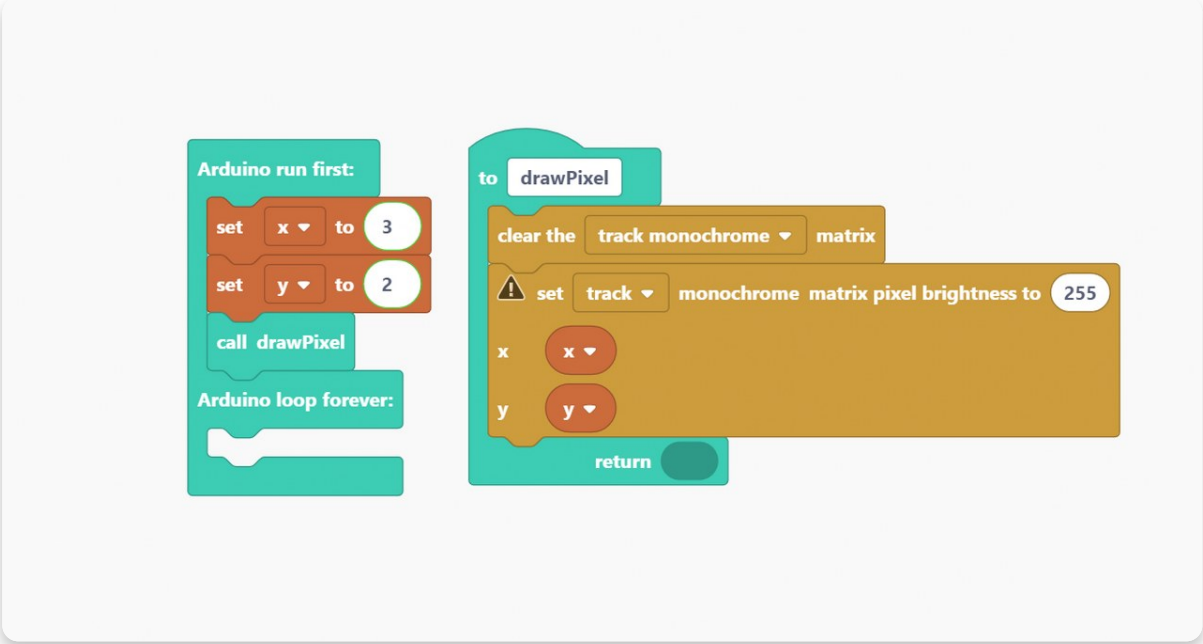
Now, let's ensure the "**drawPixel**" will be able to light up that particular LED.

Step one is to clear the matrix before anything else starts.

You'll use the "**clear the matrix**" block from the **LED matrix block section.**

We'll need the "**set track monochrome matrix pixel brightness to**" block from the LED matrix.

Will set the brightness to **255** (that is the maximum), and "**x**" and "**y**" as the variables of those coordinates.



Finally, you have to **push the changes** you have made, and for that, you have to use the "**push changes to matrix**" block from the **LED matrix block section**.

Amazing!

Now, let's go back to the **Arduino run first section**, create another **variable**, and call it **RGB**.

We'll use the "**set RGB to**" block, just like we did for the "**x**" and "**y**" variables, and place it under the "**call drawPixel**" block.

RGB variable defines the ordinal number of the LED that lights up. Set the numerical value of the **RGB variable** to **0.**

The reason why we choose 0 is the same as in the last sketch. There are five RGB LEDs in the row, and the PC starts counting from 0, so we have to put the numerical value 0 for the first RGB LED to light up.

Let's create a function for this variable.

This function will do exactly the same as the last one, but for the RGB variable.

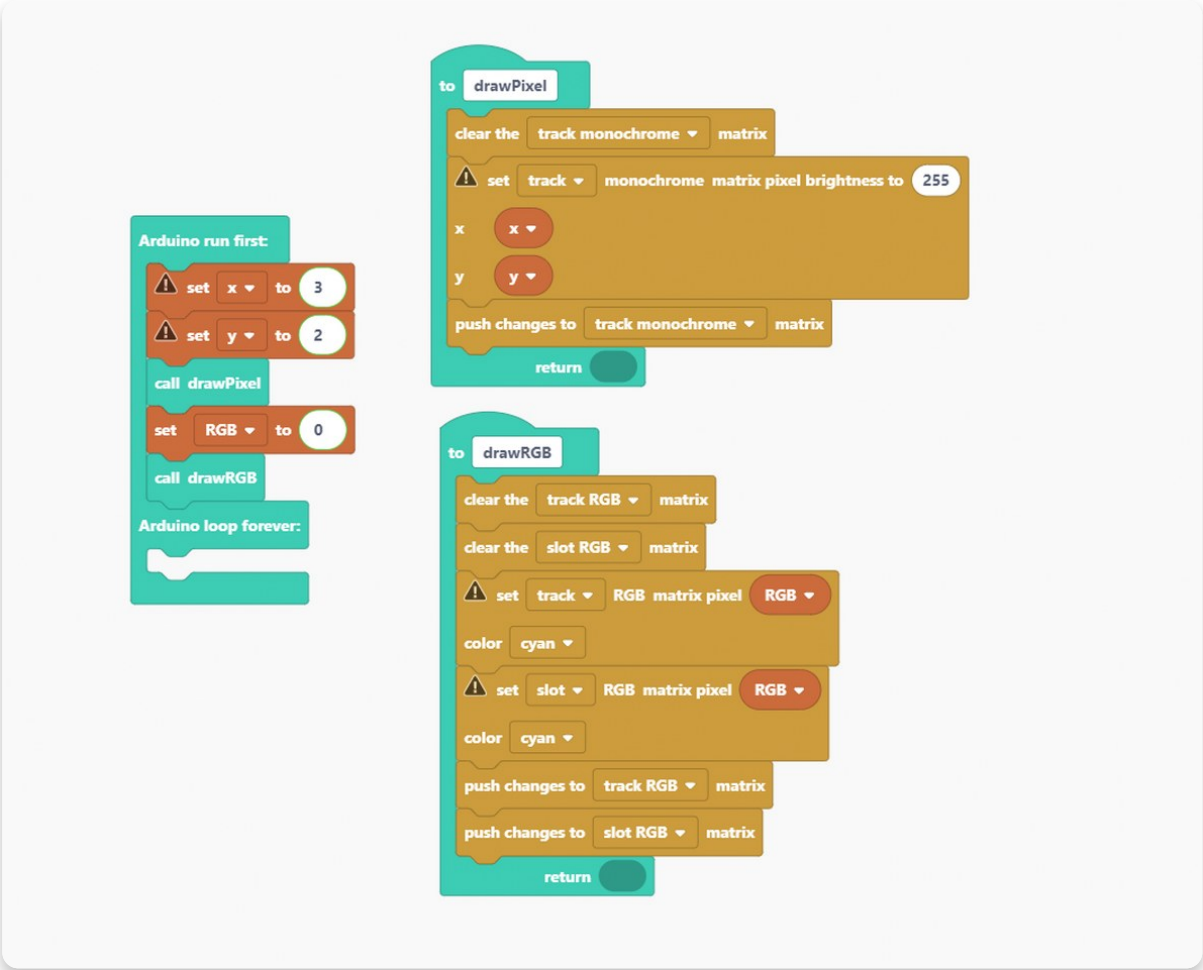
So, the next thing you'll have to do is place the "**call drawRGB**" block under the "**set RGB to**" block in the **Arduino run first section.**

DrawRGB function will light up a particular RGB LED based on the "RGB" variable.

In the "**drawRGB**" function, we'll determine the color of the LED and change the variable to RGB. The difference is that everything will be **done twice** - since we are doing it for the **slot RGB** and the **track RGB matrixes**.

Firstly, you'll have to **clear the slot and track RGB**. Then, **set the color to cyan** for both track and slot RGB, and ensure you **place the value for the RGB variable**.

In the end, you'll have to **push changes** you have made to the matrixes.



This was a setup, and this part is very important for the final code.

Now it's time to use the inputs!

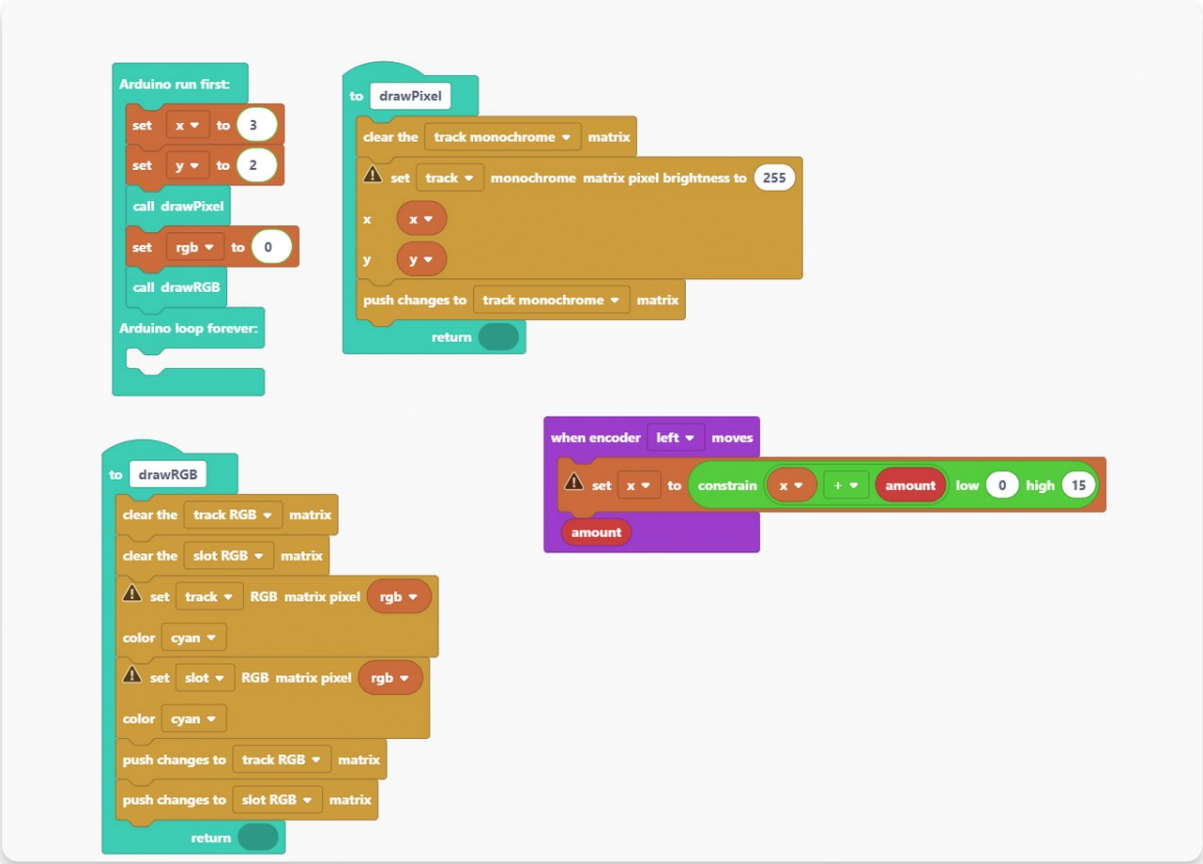
We'll kick off with the encoders.

To work with the encoders, go to the **I/O block section**, and find the purple block saying encoders.

Drag and drop this block into the sketch.

Now, we have to determine what will happen with the "**x**" variable once we **move the left encoder**.

We'll need a "**constrain**" block from the **Math section**.



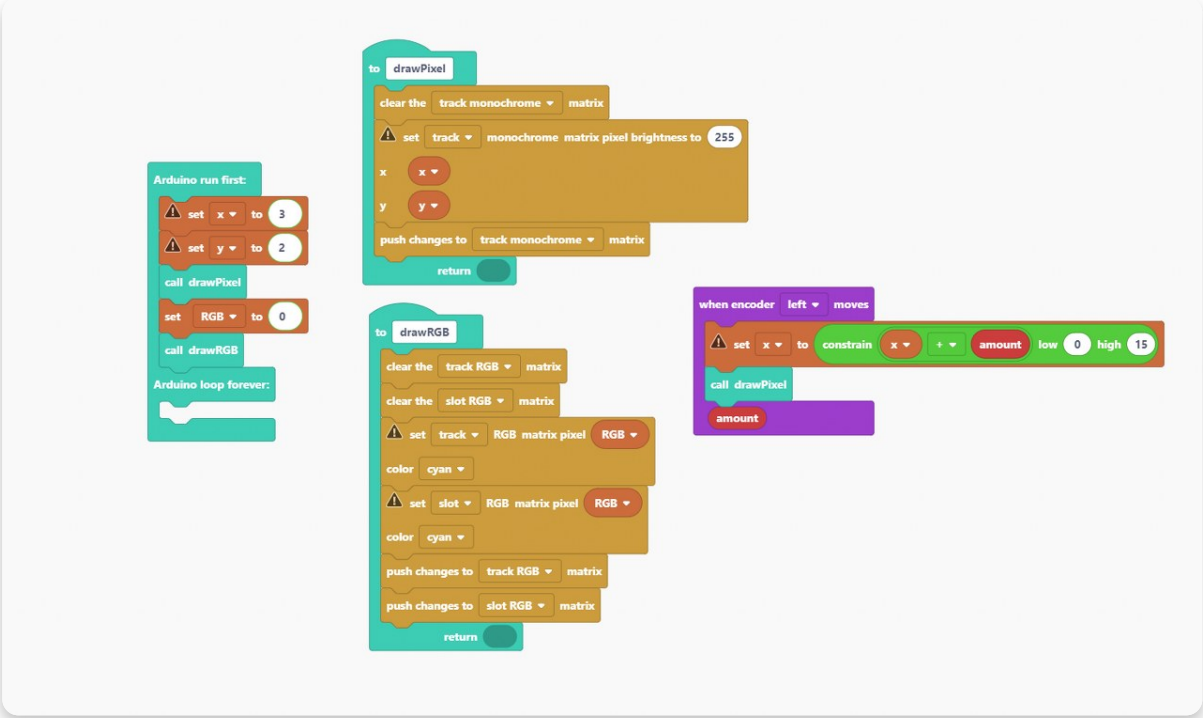
The **amount variable** holds the value depending on which side we rotate the encoder. If we rotate it to the **right**, the value of the amount variable will be **1**, and if we rotate it to the **left**, the value will be **-1**.

With the **constrain block**, we are **limiting the value of the variables between 0** and **15** (because 16 is the maximum number of the LEDs placed horizontally).

So, we took the block from the Math section, get the sum of the "x" and "value", and with the constrain block limit that sum to be between 0 and 15.

If we rotate the encoder to the right, x will increase by 1, and if we rotate it to the left, x will decrease by 1.

After we set everything up, call the "**drawPixel**" function.



Now, let's do the same thing for the right encoder.

Since we'll be using all the same blocks, you can simply duplicate everything we did for the left encoder and change it a bit.

So, instead of left, be sure to put "right encoder" and change "x" into the "y" variable.

Also, the rank for **limiting** will be between **0** and **4** since there are 5 LEDs placed vertically on the matrix.

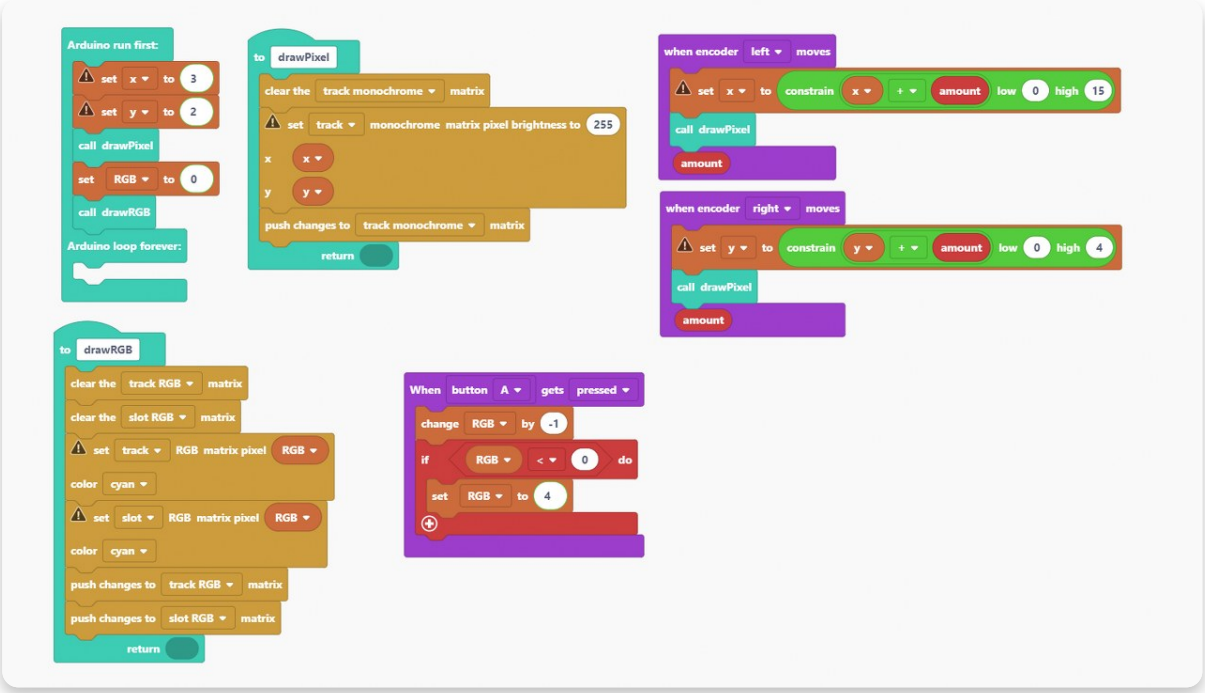
Now is the time for the pushbuttons!

We'll work only on the first two pushbuttons from the left. **The far left one will be named button A, and the right one will be named button B.**

Find this block in the **I/O block section**:

First, we'll set everything up for button A, the far left one.

As you can see, we'll immediately change the value of the RGB variable to -1.



Once we click on the button A, the value will change by 1.

With the **logic block**, we are determining that if the value of the LED falls to 0, the value will go back to 4 (the far right button will light up).

In the end, we must put the "**call drawRGB**" block inside the big purple block.

Let's duplicate everything for button B.

Change it a bit to look like the block on the photo:

Everything will be the opposite for button B.

So, if we click on the button B, the value will change by 1, and once we get to 4, the value will go back to 0 (the far left LED will light up).

And last but not least, the sliders!

First, you have to find this block in the **I/O block section:**

While we are moving sliders, their value changes depending on the position of the pixels on the matrix.

The "**value**" variable is set between **0** and **255**. **Y-axis goes from 0 to 7** (the number of the LEDs on the side of the slider).

So, we have 256 values for the slider and 8 for the LED matrix.

To map the slider to a pixel coordinate, you have to divide its value by 32 (the number we get by dividing the value of the sliders with the value of the LED matrix) and round the number.

The slider's value at the lowest position is 0, and at the highest is 255.

In the end, we have to put the "**push changes to slider monochrome matrix**" block inside the purple block.

We can duplicate everything for the right slider.

The only thing we'll change is the value of the "x" parameter to 1.

Hit the Run button, wait for the code to compile, and enjoy!

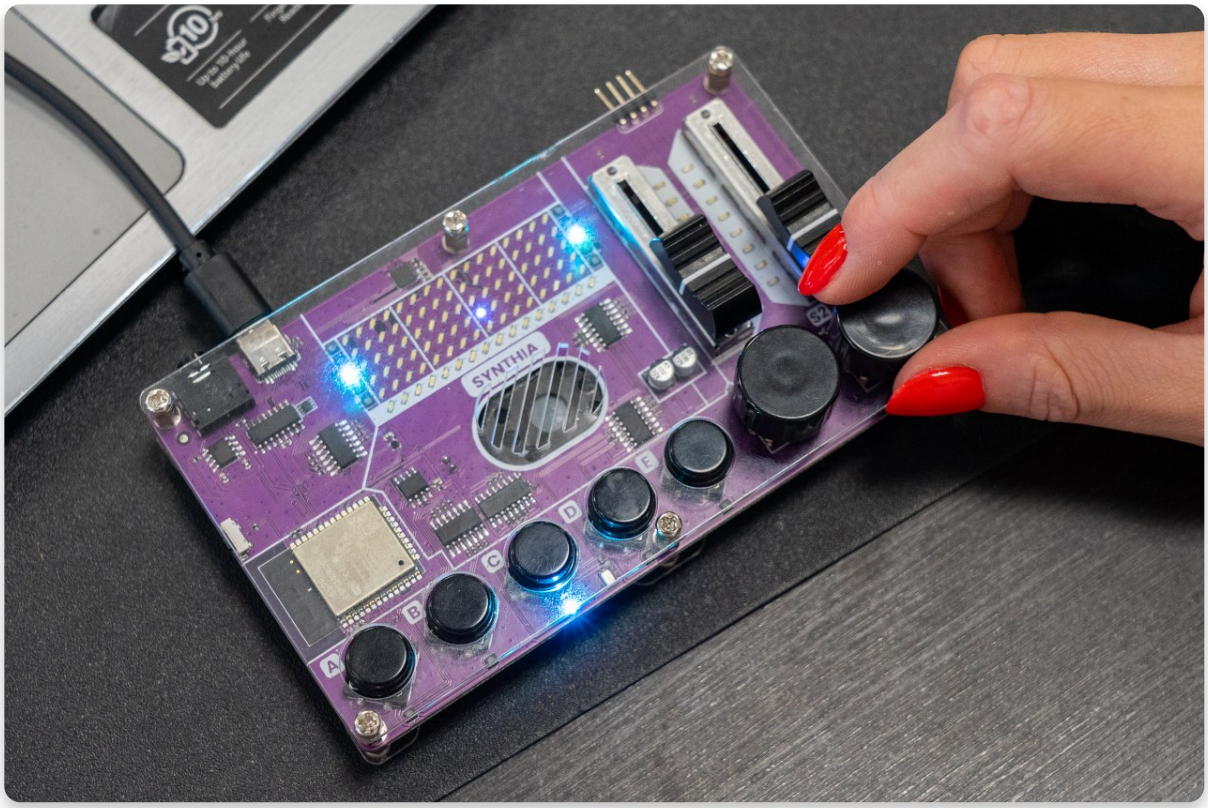
You can test your code by rotating the encoders, sliding the sliders, and clicking on the pushbuttons.

Looks amazing, right?

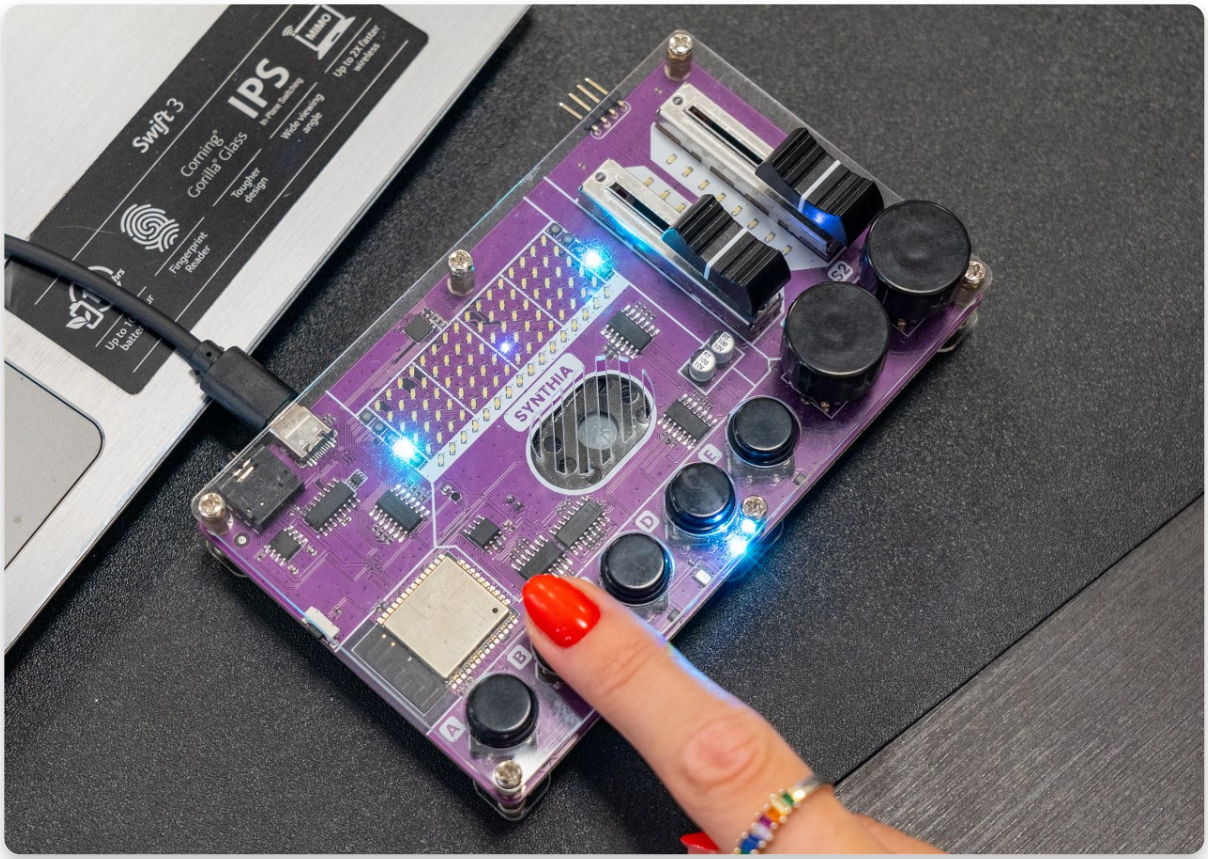
One more sketch to go, and you're ready to code on your own.

Here are a few photos showing what it will look like once you compile the code!

So if you are rotating encoders, you'll be moving the dot's position on the LED matrix.



If you press the far left or the right pushbutton, another LED will light up.



Let's draw!

Welcome to the explanation of the final sketch we have prepared for you.

Once again, click on the new sketch and choose Synthia.

Firstly, we'll make three **variables** and call them "**x**", "**y**", and "**lifted**".

As you already know, "**x**" and "**y**" variables determine the **current position of the LED in the matrix**.

The "**lifted**" variable is a new one; with it, you determine if the **pen is not lifted and ready to draw**. This is the **boolean variable**, meaning it can be **true or false.**

Let's set the values for these variables.

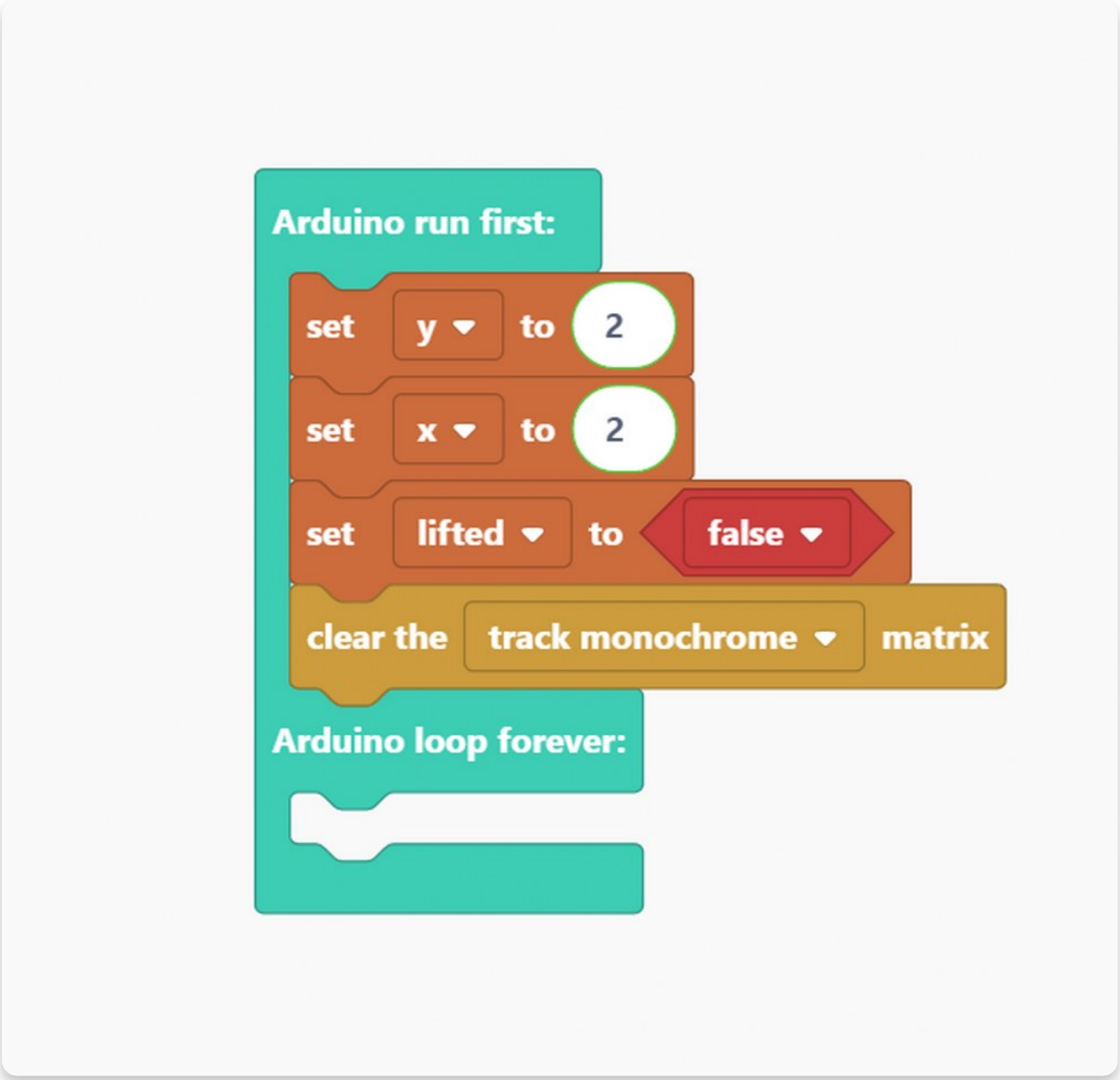
For that, we'll use the "**set to**" block from the **Variables block section**.

"**X**" variable will be set on 2, just like the "**y**" one, and the "**lifted**" variable will be set to false.

If the "lifted" variable is set on false, that means the pen was not lifted (ready to draw), and if it's set on true, the pen is lifted and it is not ready to draw.

Another important thing is to **clear the matrix** every time while turning on the device so you can draw on it.

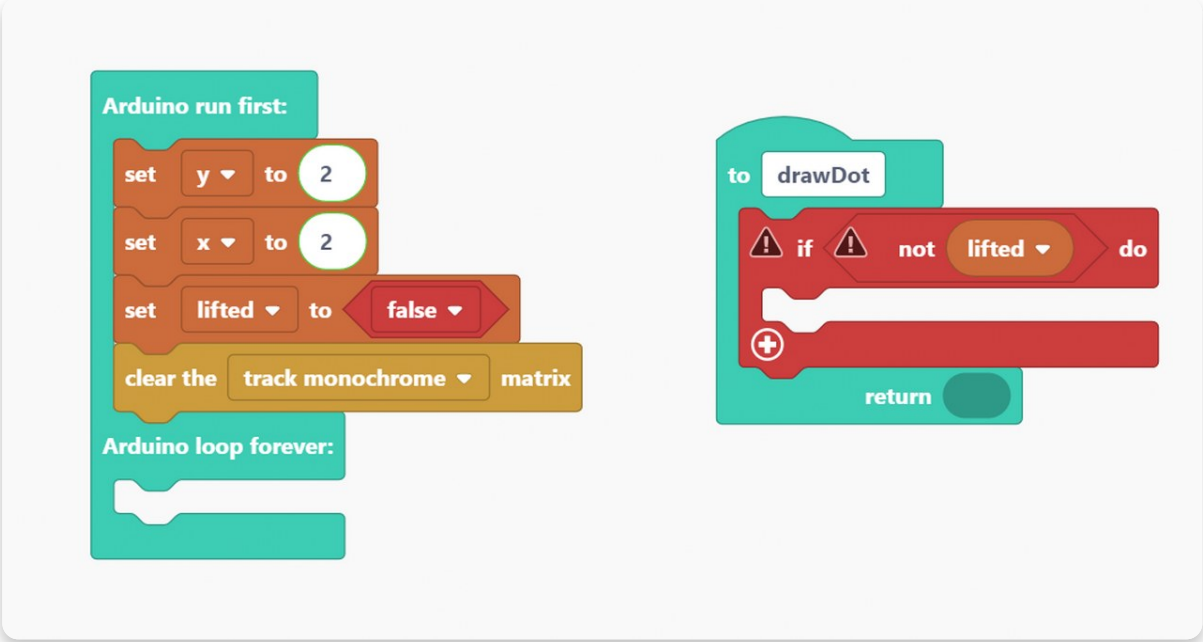
We need the **"clear the track monochrome matrix**" to clear the matrix.



Now that we set the main variables let's create a **new function** and call it "**drawDot**".

This function **draws the current position of the dot** on the matrix.

Take the "**if-not**" block from the **Logic section** of the blocks, and drop it in the "**drawDot**" function.



Another thing to do is to set matrix pixel **brightness** to the maximum - **255**!

In the end, drag, and drop the "**push changes to matrix**" block to ensure your changes will be saved.

You'll use encoders for drawing on the matrix. Because of that, now is the time to use those purple blocks from the **I/O block section.**

With the **left encoder (x variable)**, we'll change the **horizontal position of the dot** on the matrix. With the **right encoder (y variable)**, we'll change the **vertical position of the dot.**

Remember the previous sketch?

We'll do the same steps here.

So, we'll make sure to add the **amount variable** (-1 if we rotate the encoder to the left, or 1 if we rotate the encoder to the right) **to "x" or "y"**, and to **limit** this from 0 to 15.

Let's duplicate this block for the right encoder!

Don't forget to put the "x" variable on the left encoder but the "y" variable on the right encoder.

Also, keep in mind that you're changing the matrix's **width with the left encoder**, so the limit is from 0 to 15. On the other hand, you're changing the **height** of the matrix with the **right encoder**, so the limit is set from 0 to 4.

Once again, we are repeating the same values and limits from the previous sketch.

Now, we'll have to "**call drawDot**" variable under all blocks to make sure the **dot will move** in the matrix.

Your sketch should look like this by now:

The last thing we will do is set what will happen if one of the **encoders** gets **pressed**.

E1 symbolizes the left encoder, and E2 symbolizes the right one.

For that, you'll need this block from the **I/O block section**:

While pressing the encoders, we want to change the "lifted" variable.

So, as the blocks say - if you press the left encoder, the variable will go from lifted to not lifted.

Let's duplicate the blocks for the right encoder.

As you can see, pressing the right encoder will do the same.

So, if the variable was lifted, it will change to not lifted. Of course, this goes both ways, so if the variable is not lifted, pressing on the encoders will change it into lifted.

Congratulations!

You successfully finished all the sketches and are on the way to becoming a real programmer!

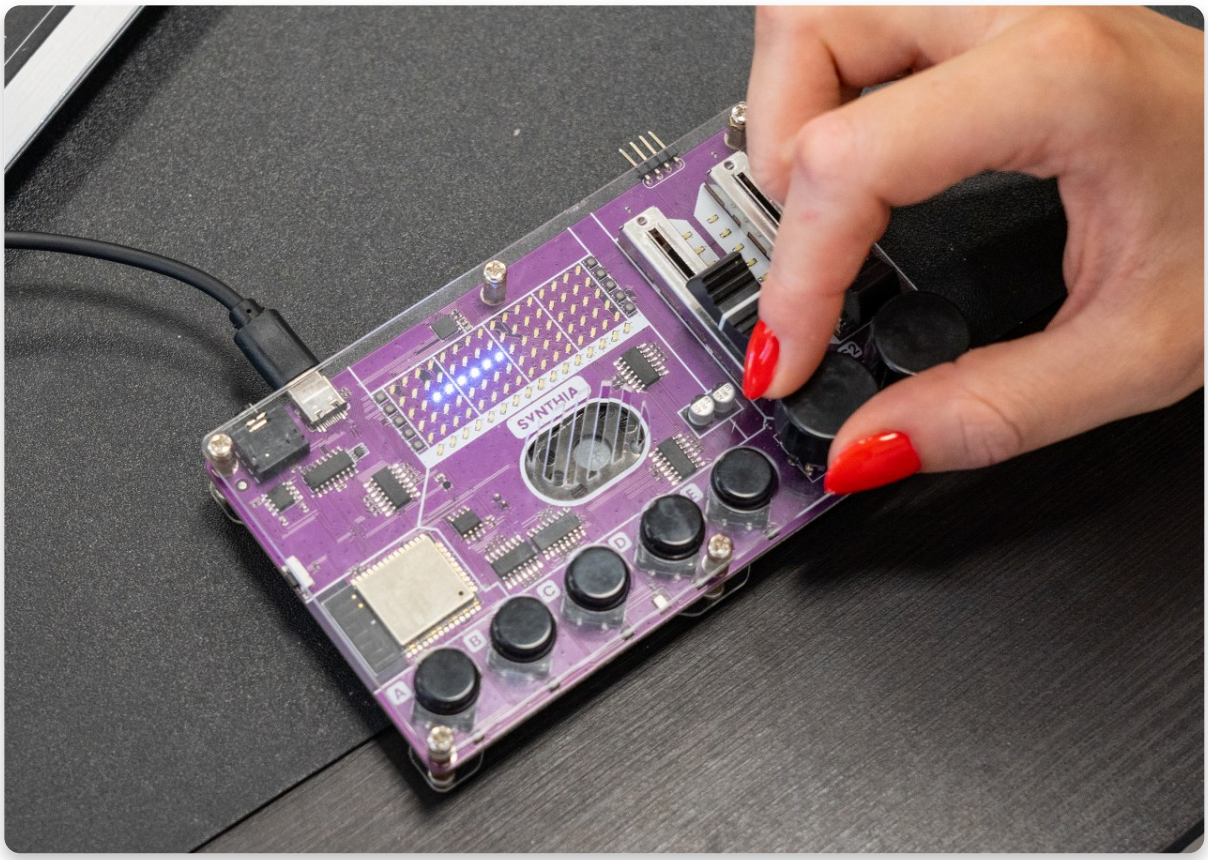
We are almost done with this guide, but make sure to check the next chapter because we'll show you how to delete all the changes you have made and how to

restart your device.

Here are a few photos showing what it should look like once you compile the code.

Once you run the code and the device turns on, you'll see this on Synthia.
So, a little dot (LED on the LED matrix) that is lightning up is the one we set at the beginning of the sketch. That is your starting point while drawing on the matrix.

If you rotate the left encoder, you'll be able to draw horizontally, and if you rotate the right one, you'll be able to draw vertically.



However, if you click on any encoder, the pen will stop working, and if you click on it once again, it will start working.

Restore Synthia's base firmware

Restore Synthia's firmware

Once you're done coding and just want your Synthia to be "normal" again, you need to restore its base firmware.

This is quite simple, just connect your Synthia to the USB port of your computer and press the "Restore firmware" button on the top right.

You will be prompted with a window to choose the device you are restoring the firmware for.

Choose Synthia, of course.

Wait for a few seconds, and your Synthia will be back and running like usual.

You need to do this whenever you're done coding your Synthia if you want it to revert to its initial out-of-the-box functionality.

What's next?

You've reached the end of our first Synthia coding tutorial, congratulations!

We hope you're as excited as we are about Synthia's future since there are so many cool things we want to do with it in the future firmware and CircuitBlocks updates.

In the meantime, continue exploring on your own and show us what you've done with your Synthia by sharing it on the **CircuitMess community forum** or via our **Discord channel.**

If you need any help with your device, as always, reach out to us via contact@circuitmess.com**,** and we'll help as soon as we can.
Thank you, and keep making!