

ByteBoi coding – first steps

Introduction

Installation

Welcome to the ByteBoi coding tutorial

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

We'll use CircuitBlocks for coding your newly-assembled gaming console.

CircuitBlocks is a custom-made coding app that we've designed.

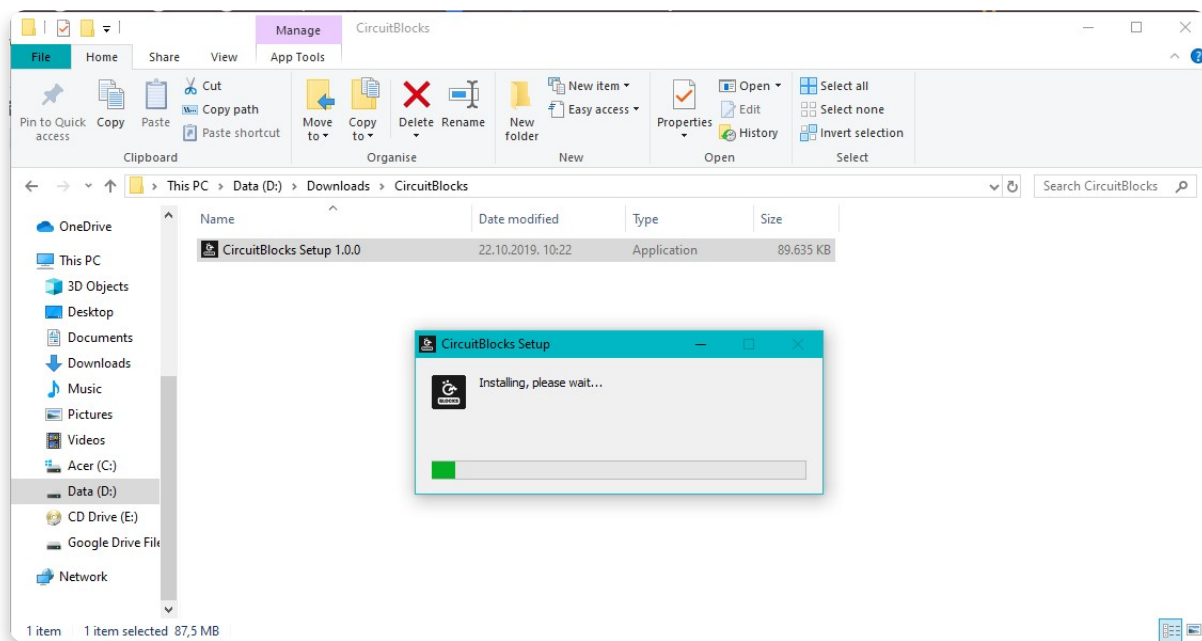
You will code your ByteBoi in CircuitBlocks' graphical block-based coding interface that will help you make your first steps in the world of physical computing.

Installation

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

If you have a Windows computer

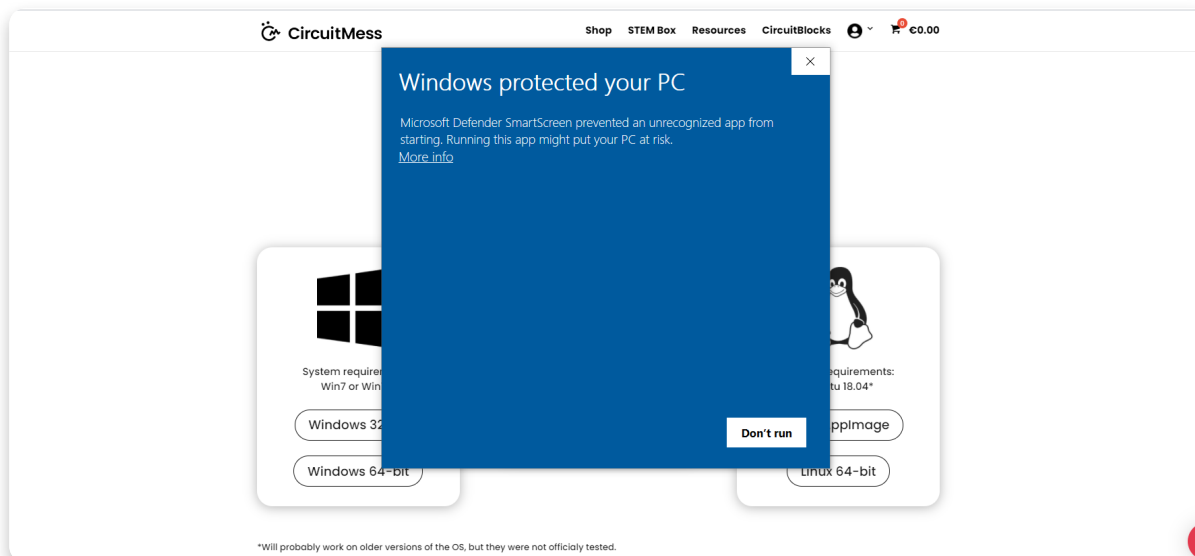
1. Go to the [CircuitBlocks download page](#)
2. **Download the latest version for Windows** – Check if you have a 32 or 64 version. Go to Settings on your PC, click on the System option and find the About section where you'll see the system type.
3. Double-click the downloaded file named "CircuitBlocks"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



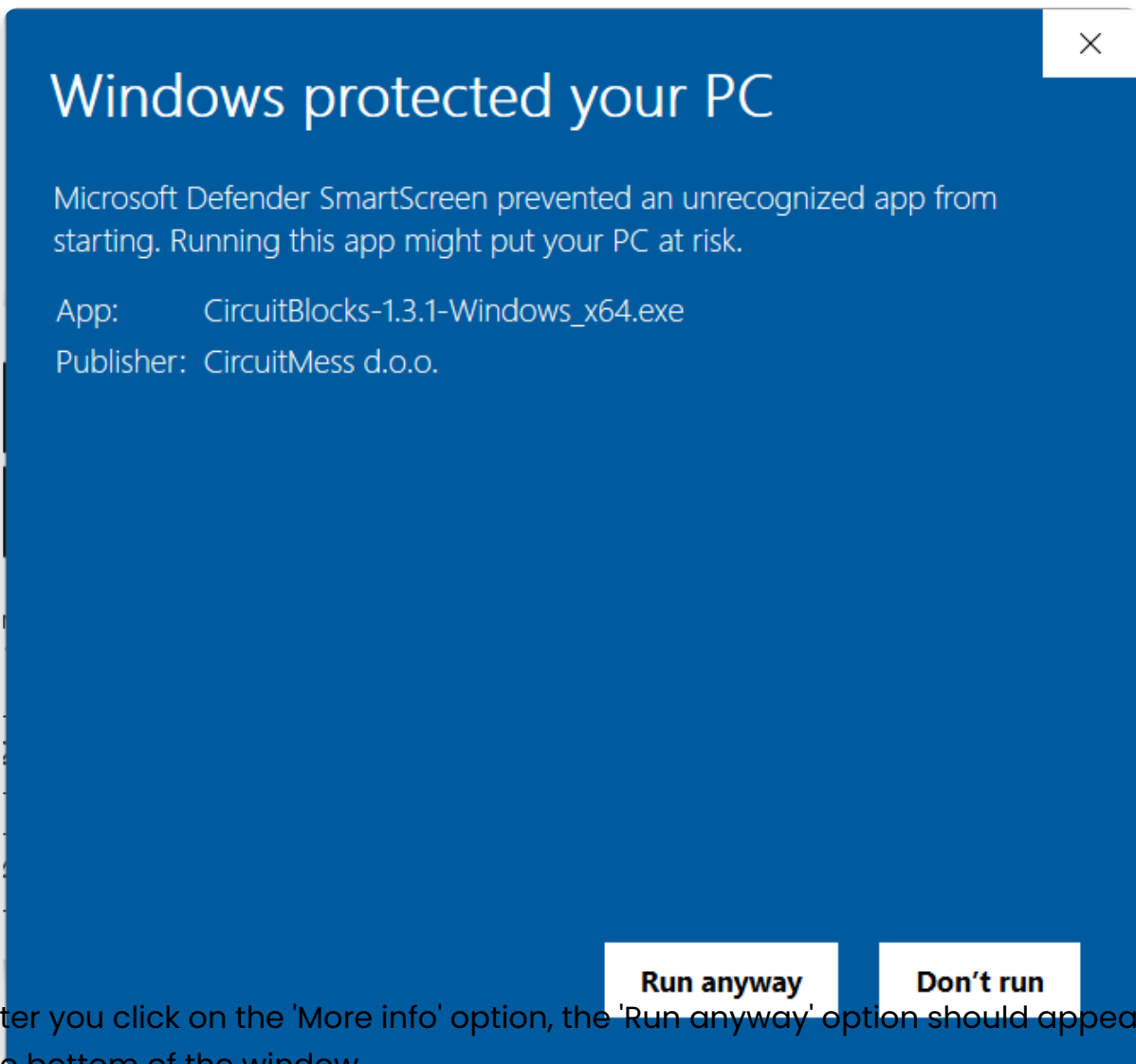
Your PC is not at risk!



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.

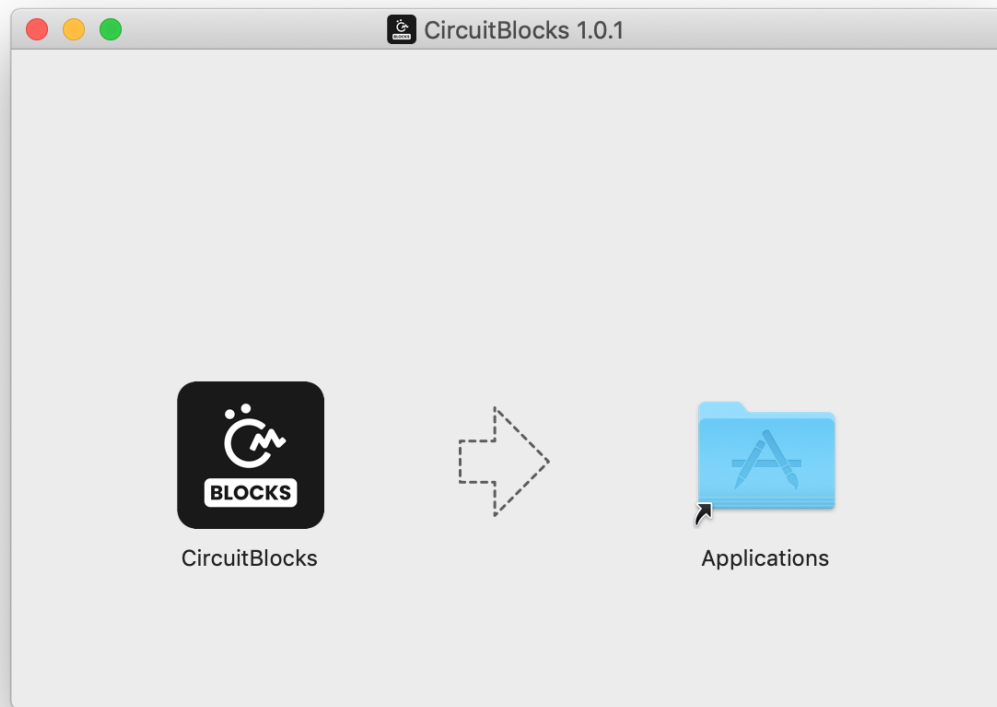


After you click on the 'More info' option, the 'Run anyway' option should appear at the bottom of the window.

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



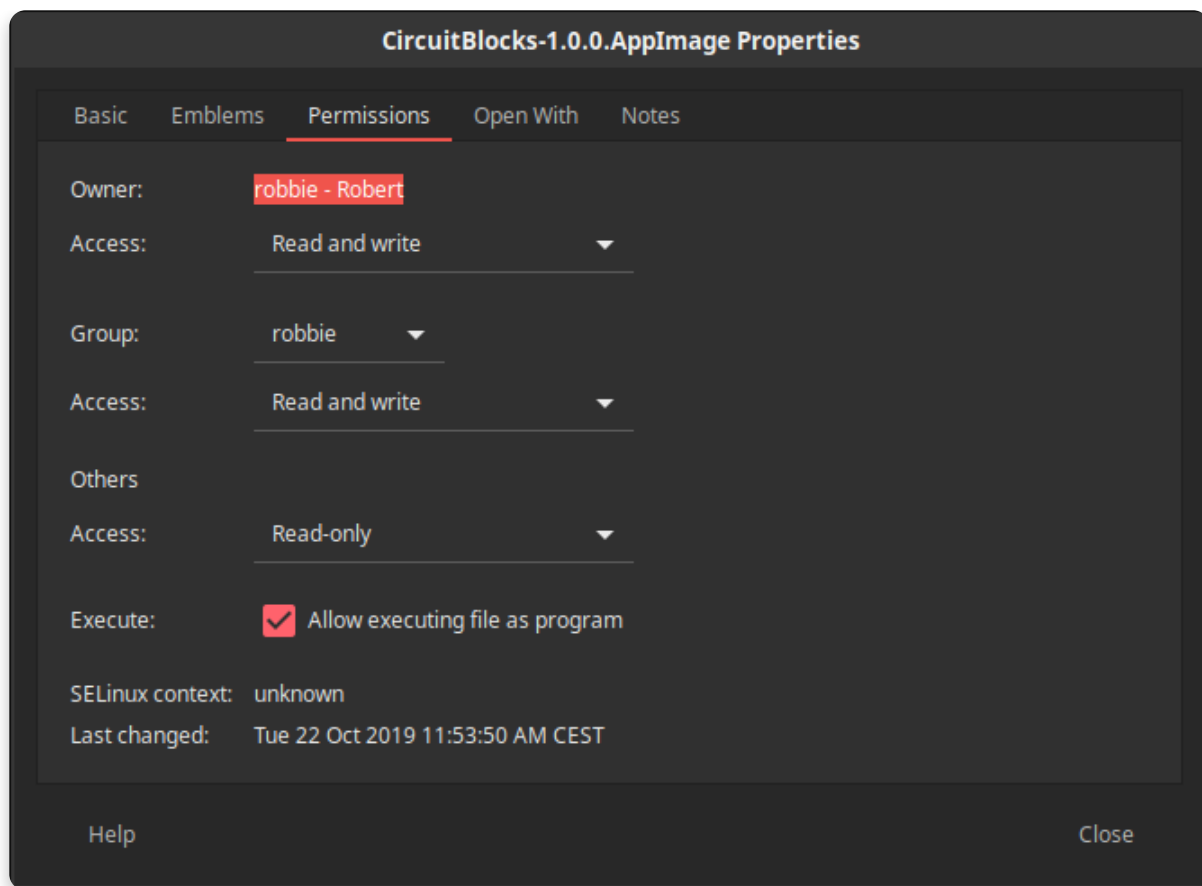
If you have a Linux computer

There are two ways of installing CircuitBlocks on Linux

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):

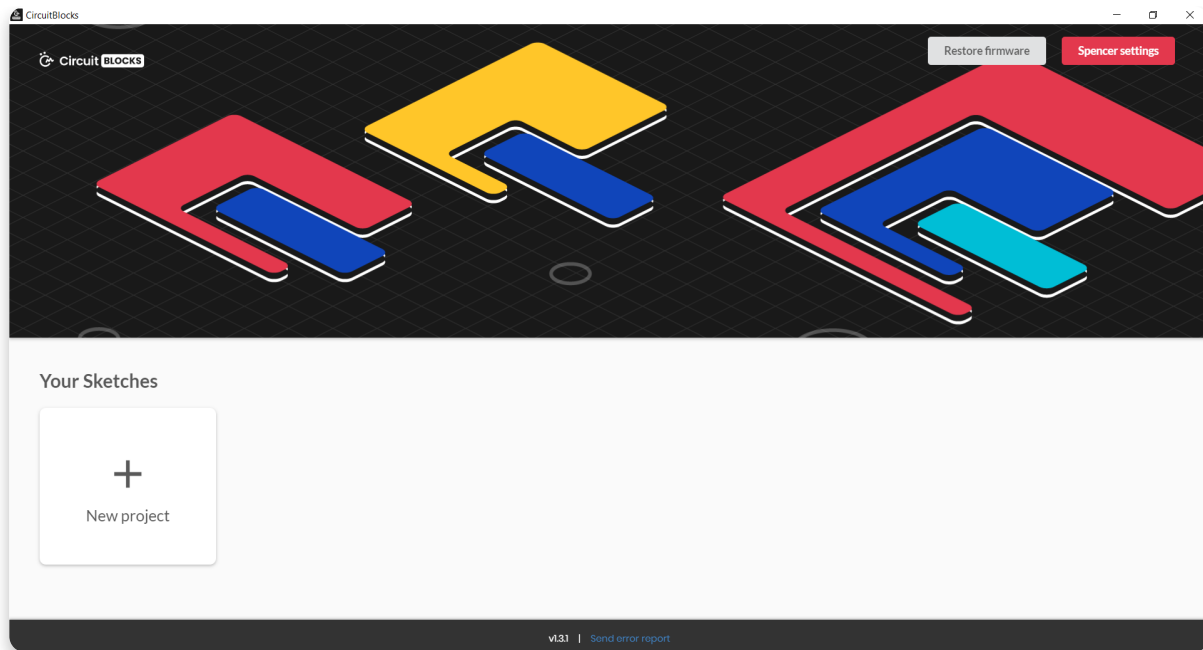
1. Go to the [CircuitBlocks download page](#)
2. Press the "Linux AppImage" download button
3. Right-click on the file and select 'Properties'
4. Go to 'Permissions' and tick 'Allow executing file as program'
5. Double-click the file, and the installation will complete automatically



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)

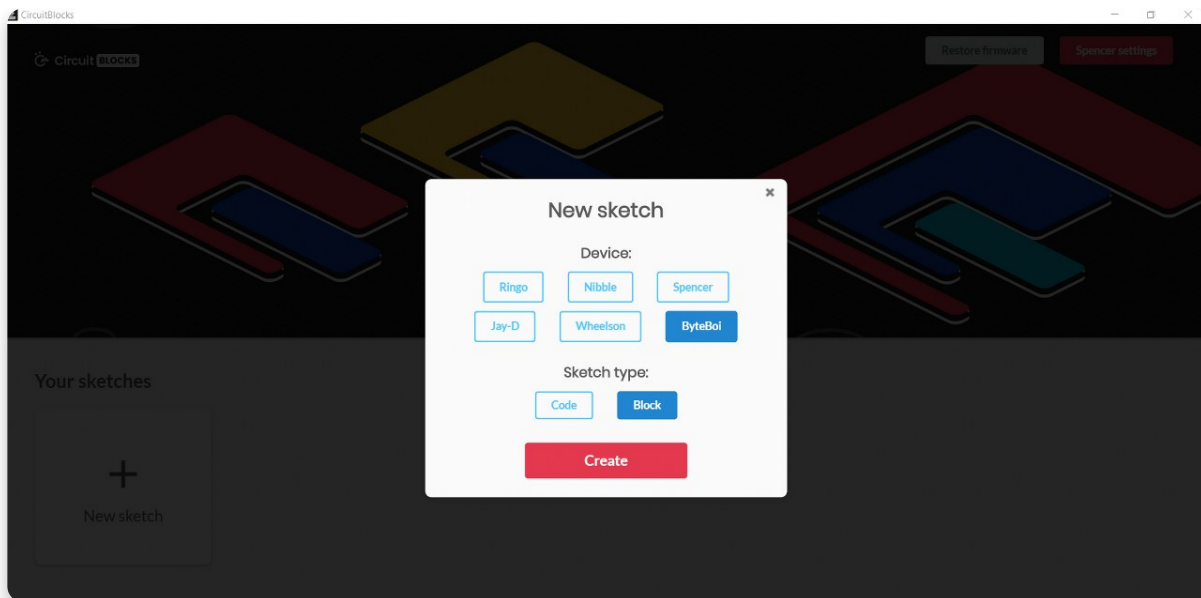
Press on the big "New project" button.

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

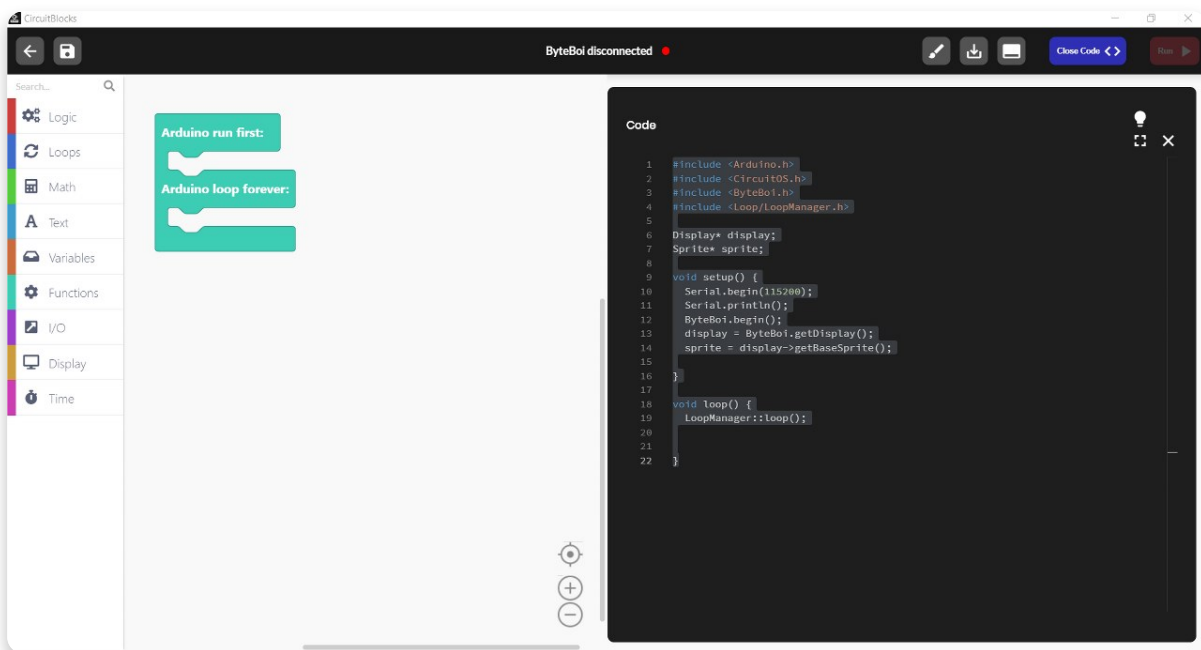
For the device, pick **ByteBoi**.

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.

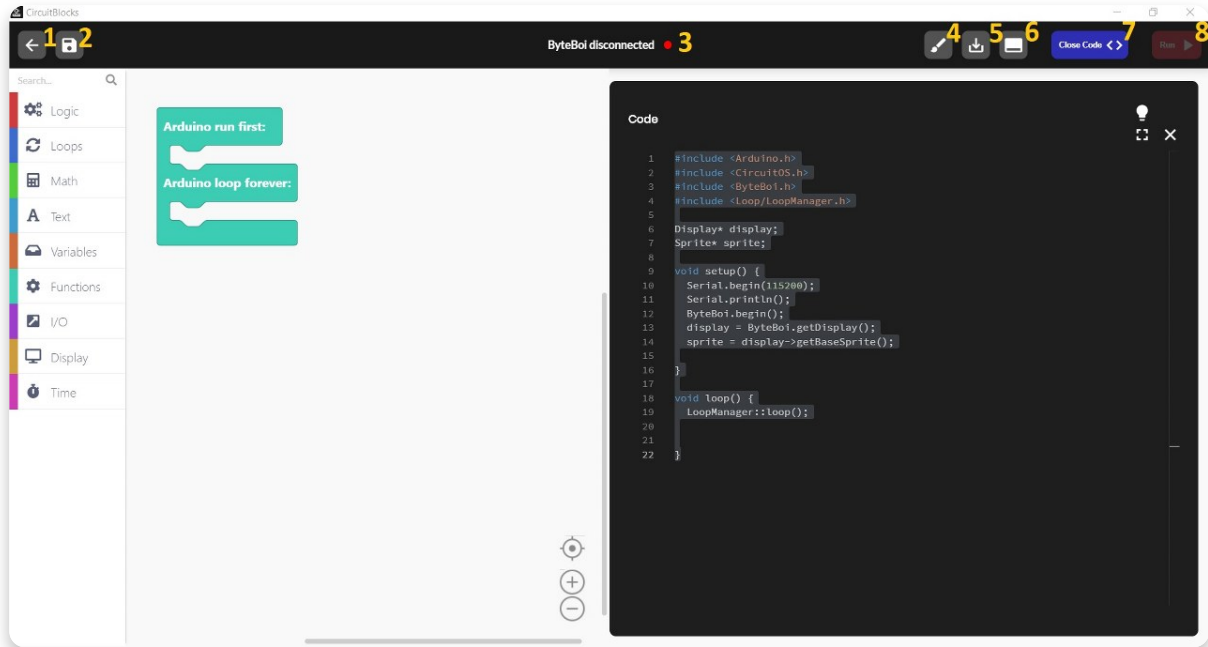
In the middle of the screen is where you'll be "drawing" your code with colorful blocks.

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

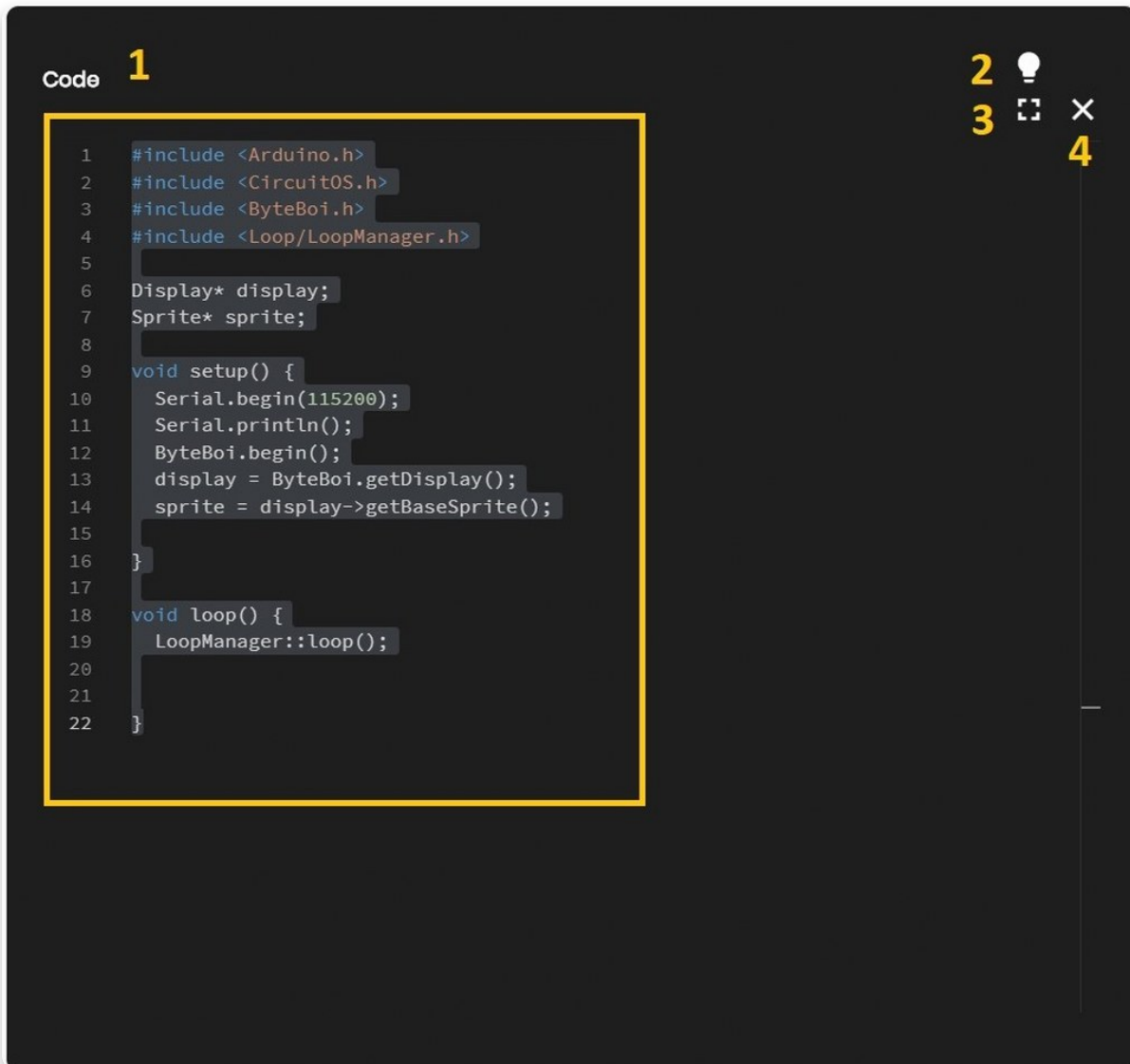


Here's a short explanation of what the buttons in the window toolbar do:

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. **ByteBoi connection indicator** – the red dot turns green if your ByteBoi is connected to your computer via a USB cable
4. **Sprite editor** – for drawing characters you want to have on your ByteBoi
5. **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
6. **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 ByteBoi and your computer. In this window, you will later be able to see the messages sent from ByteBoi to your computer via the USB port.
7. **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.

8. **Run** – This button will translate the code you have constructed in CircuitBlocks to *machine code* that ByteBoi understands (beep boop beep boop 1011100101) and send the code to your ByteBoi via the USB port

Code window



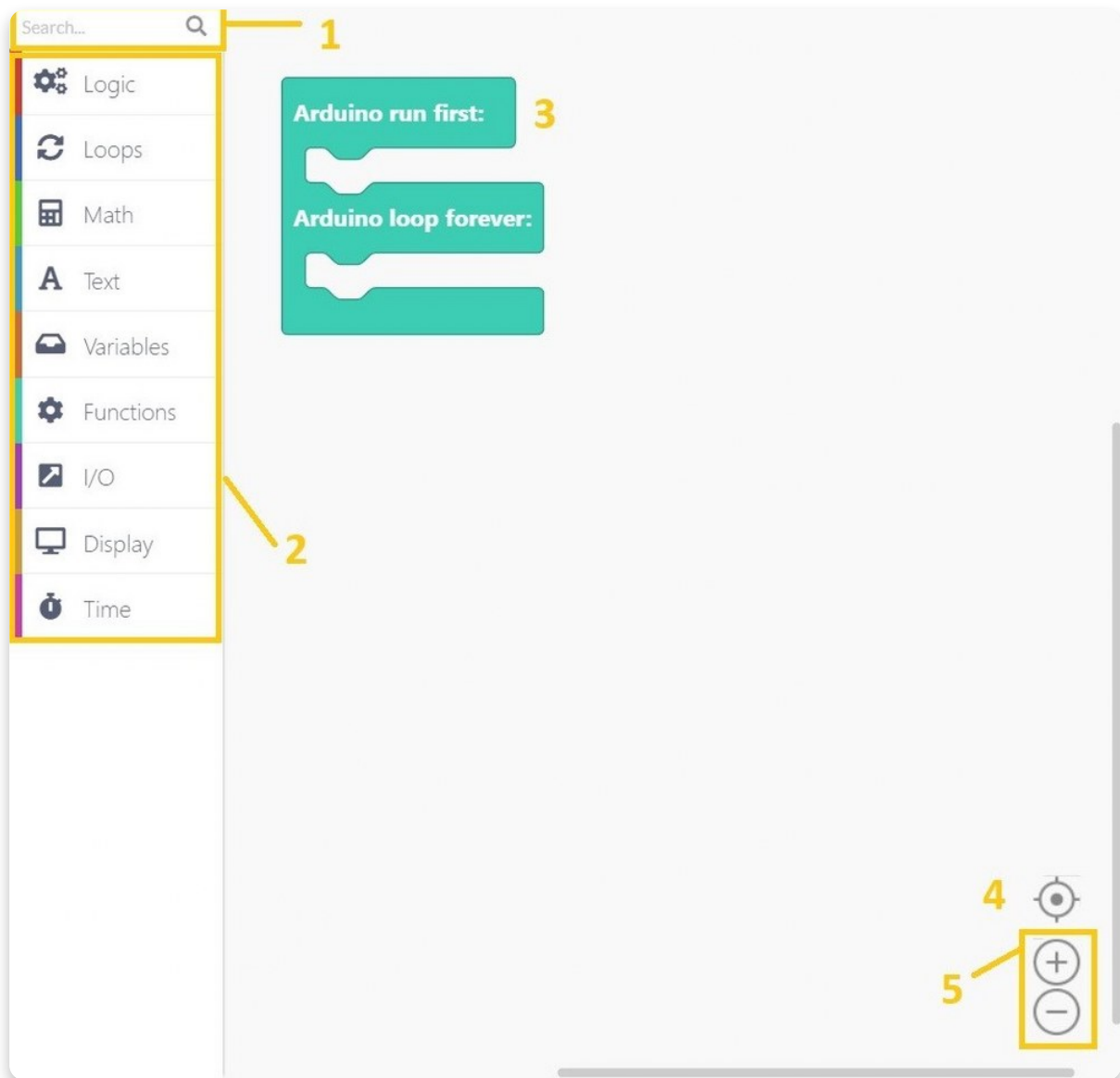
```
1  #include <Arduino.h>
2  #include <CircuitOS.h>
3  #include <ByteBoi.h>
4  #include <Loop/LoopManager.h>
5
6  Display* display;
7  Sprite* sprite;
8
9  void setup() {
10     Serial.begin(115200);
11     Serial.println();
12     ByteBoi.begin();
13     display = ByteBoi.getDisplay();
14     sprite = display->getBaseSprite();
15
16 }
17
18 void loop() {
19     LoopManager::loop();
20
21 }
22 }
```

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.

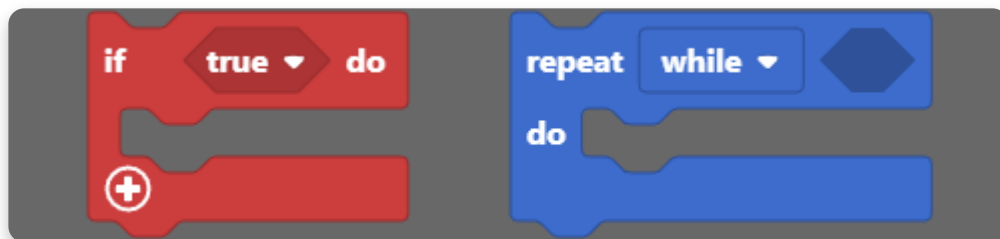
Regardless of color, each of these blocks returns either true or false.

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.

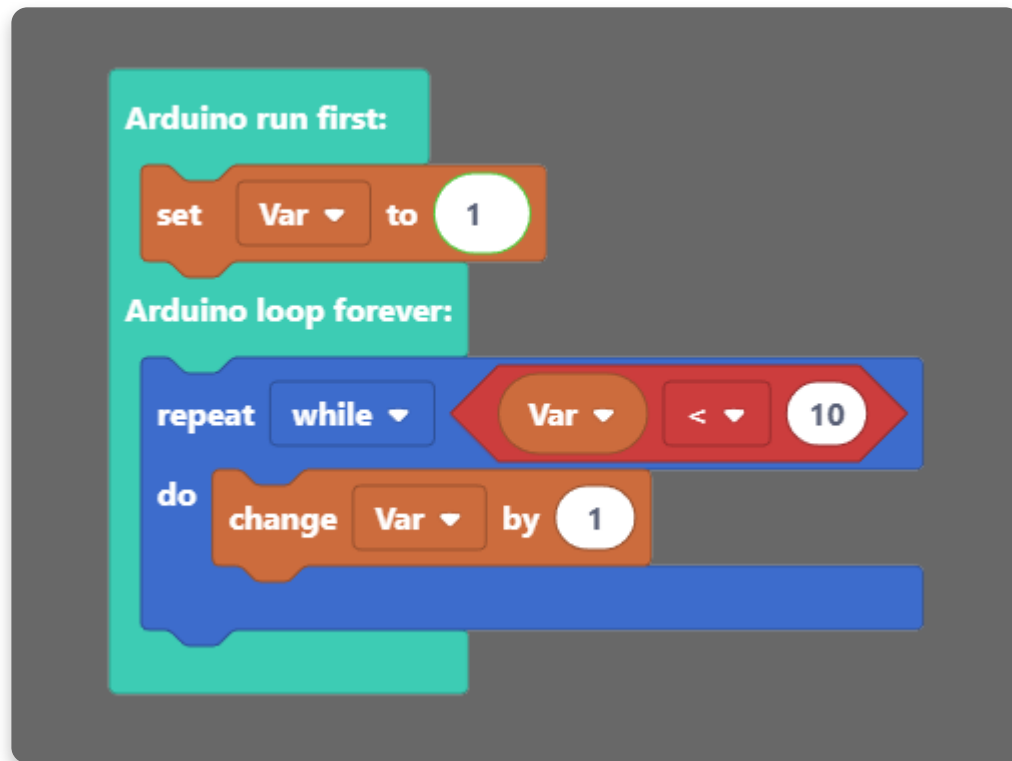
The whole point of blocks-like IDE is connecting blocks and placing them inside another.

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.

Sections themselves are pretty 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

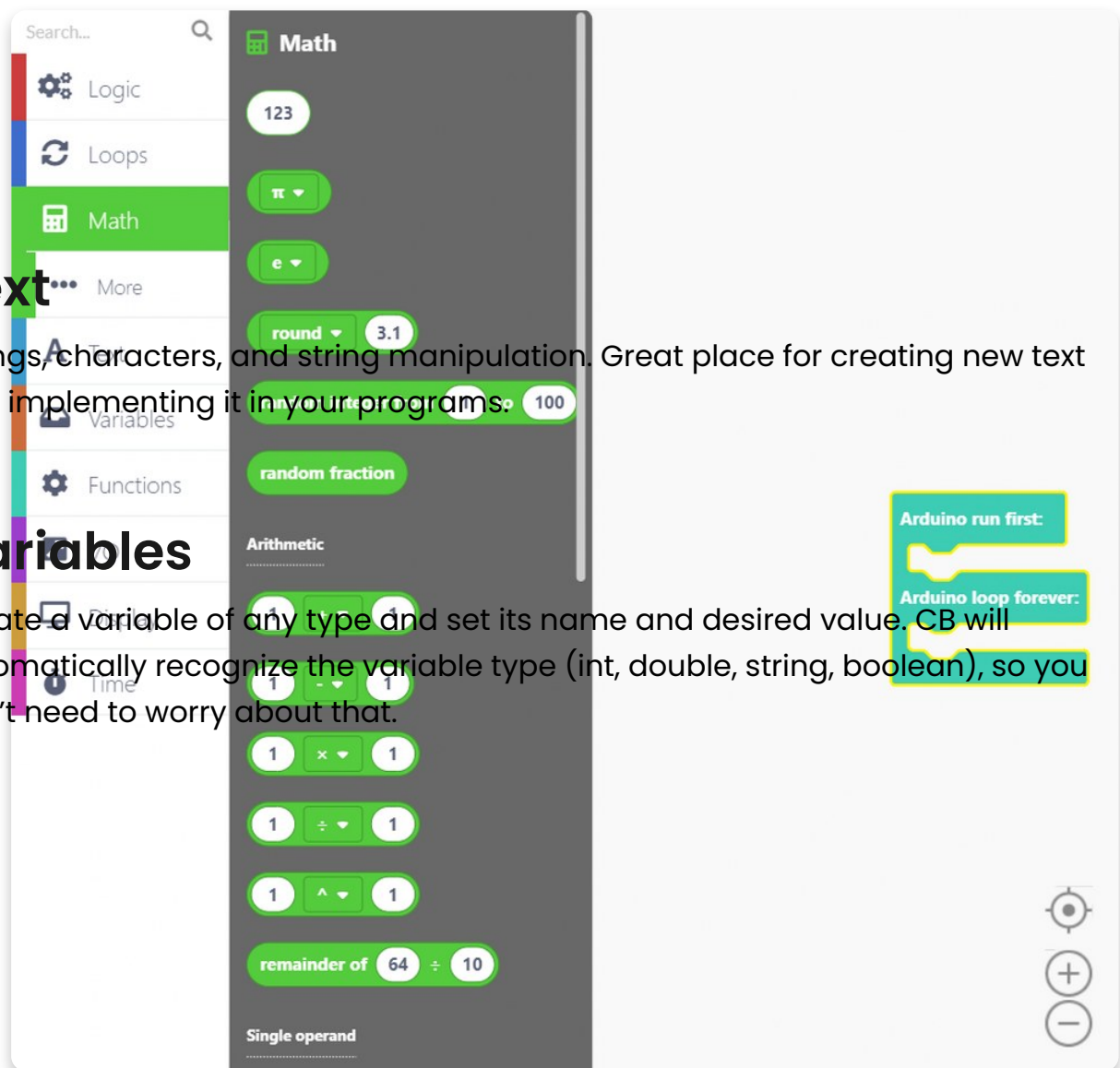
Pretty much every math function is located here. From basic operations to rounding numbers and working with angles, you will easily trigger your inner Einstein or Pythagoras in a matter of seconds!

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

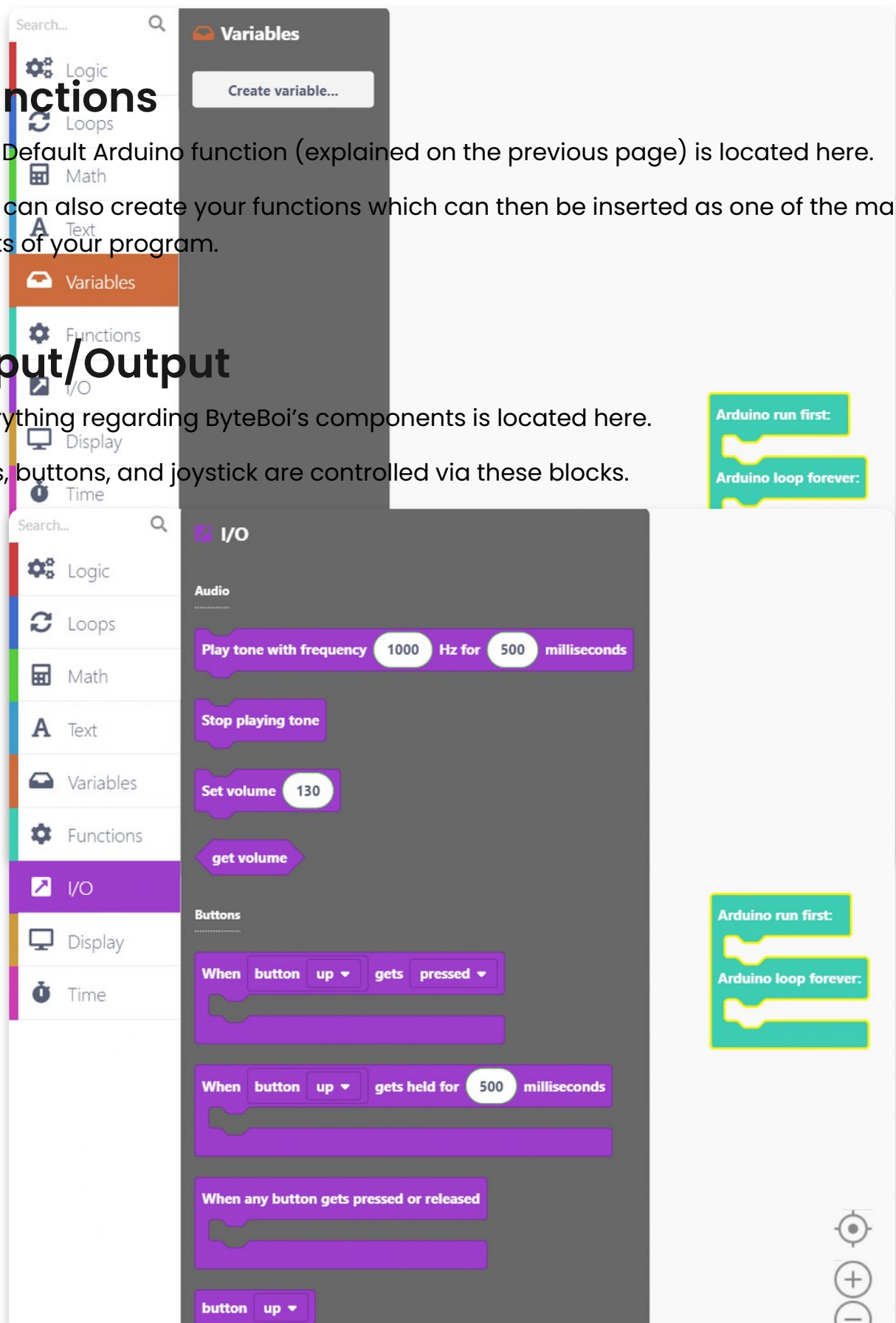
The Default Arduino function (explained on the previous page) is located here.

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

Input/Output

Everything regarding ByteBoi's components is located here.

LEDs, buttons, and joystick are controlled via these blocks.





Display

Well, all these blocks are really not important if you don't see anything on the screen!

Here is where all the magic translates to those colored pixels. You can create so much through these blocks.

Time

Delays, timers, and other time-related stuff, 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!

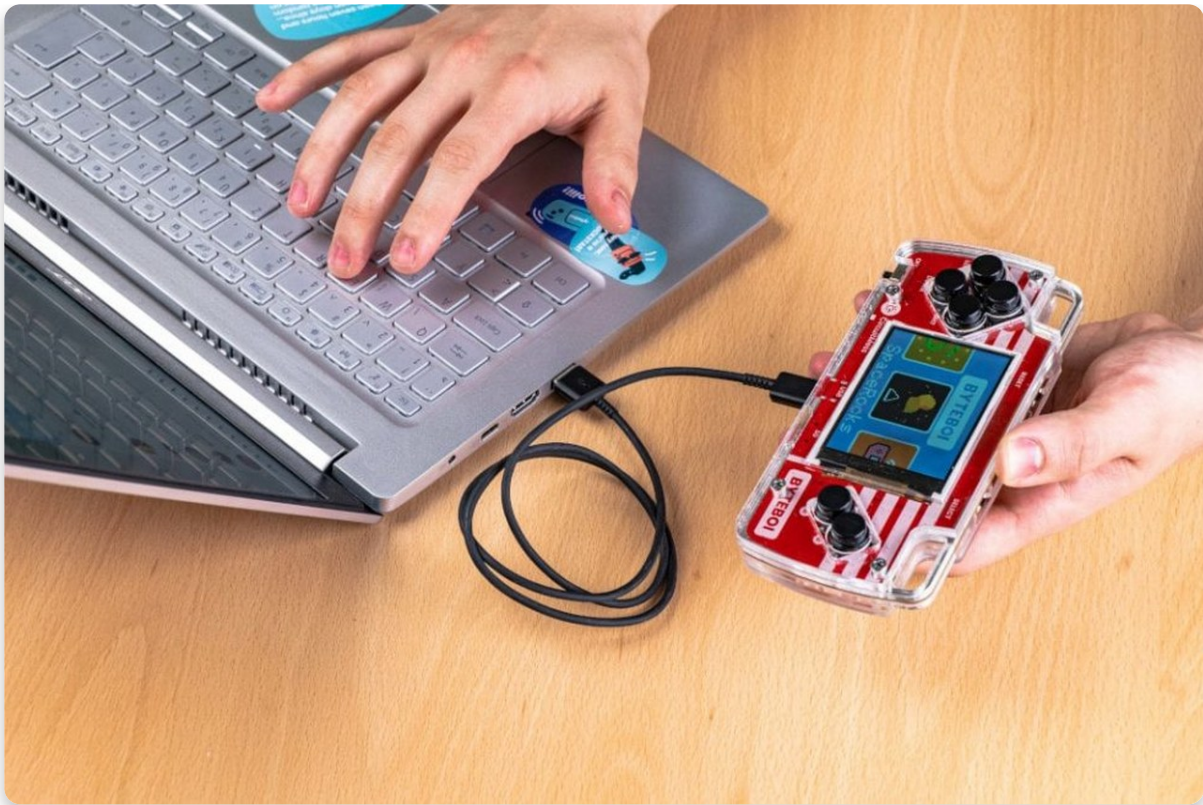
It's time to move on to the next lesson...

Let's start! Step by step

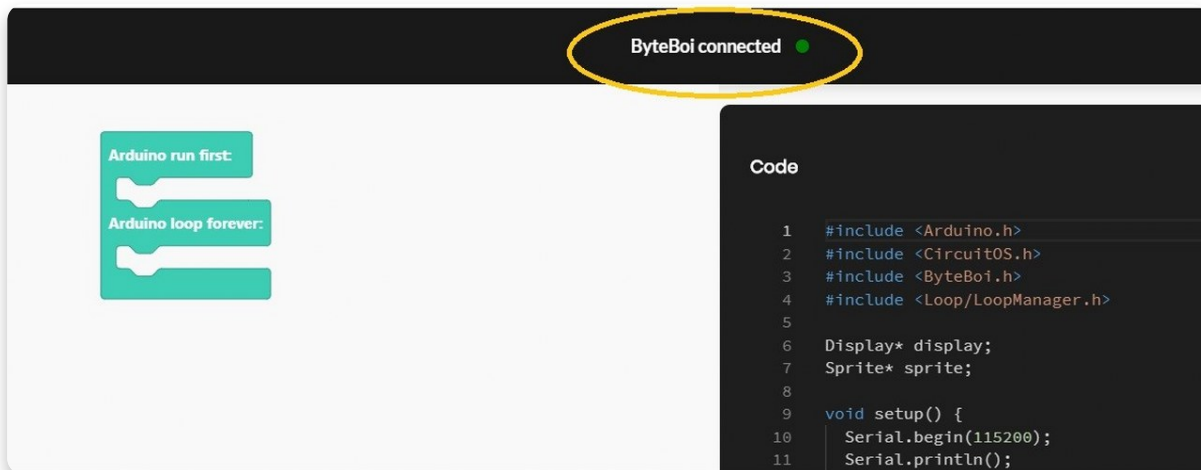
Draw images

Let's get down to business!

First, you need to connect your ByteBoi to your computer's USB port and turn it on.



CircuitBlocks should now say "ByteBoi connected".



If CircuitBlocks didn't recognize your ByteBoi, please check if the USB cable is plugged in properly and if you are using a working USB port on your computer.

If you still cannot get CircuitBlocks to recognize your ByteBoi, something possibly went wrong with the driver installation on your computer. Drivers are these little programs that help your computer communicate with ByteBoi, and they sometimes act funny. Reach out to us via email at contact@circuitmess.com if you cannot get your computer to recognize your ByteBoi.

Run the sketch/restore the firmware, so the error occurs again, then go back to the home page in CircuitBlocks, scroll to the bottom, and in the footer, you will find a "Send error report" button. It will open a window with the information that

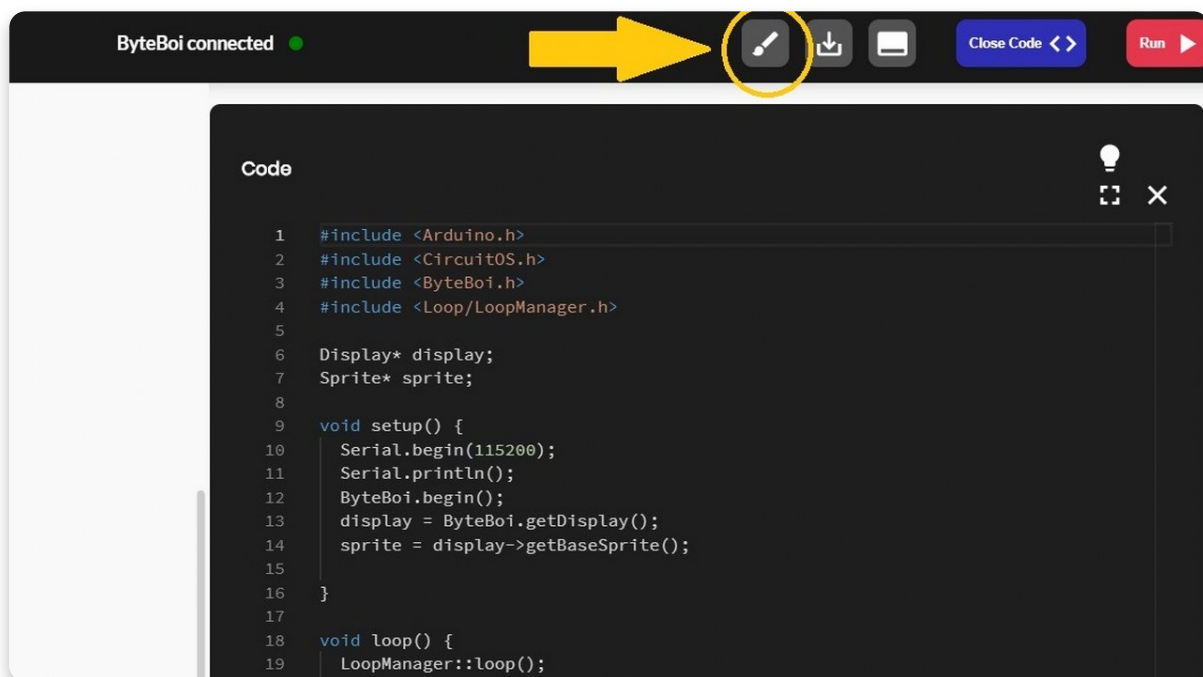
includes the error that happened and some other info about your system that might help us resolve your problem. When you click "Send report", the report will be sent to our servers, and you will get a report ID. Please get back to us with that ID, and we'll be able to assist further.

Let's draw something!

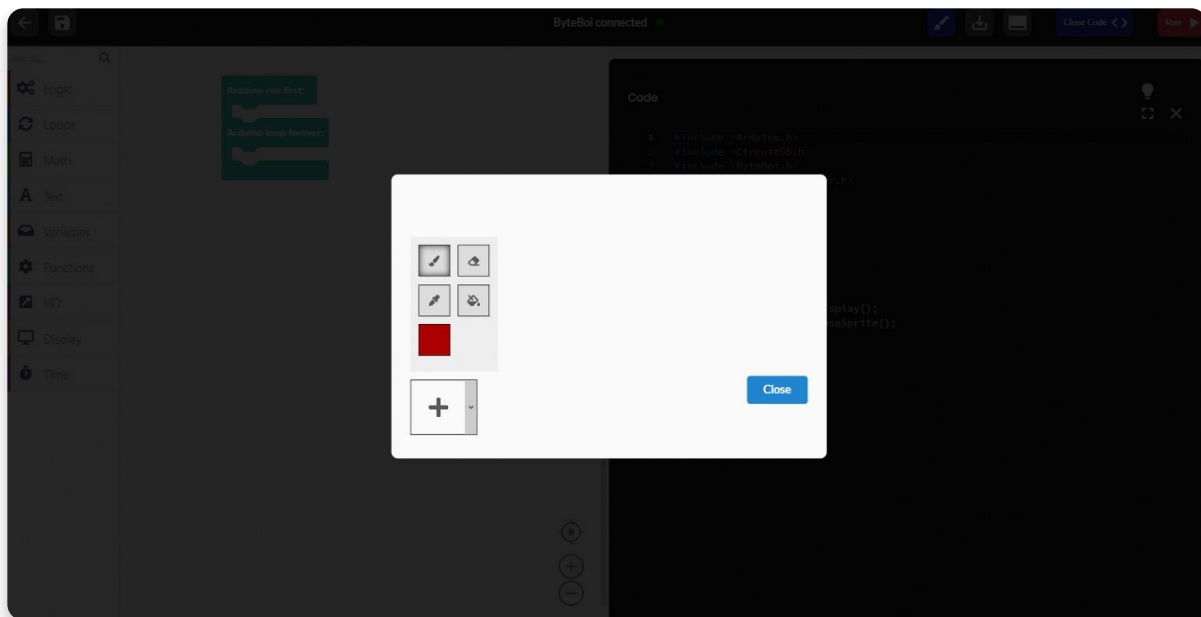
The first thing we're going to learn is how to use our new feature called **Draw sprite**!

In computer graphics, a **sprite** is a two-dimensional image or animation that is integrated into a larger scene.

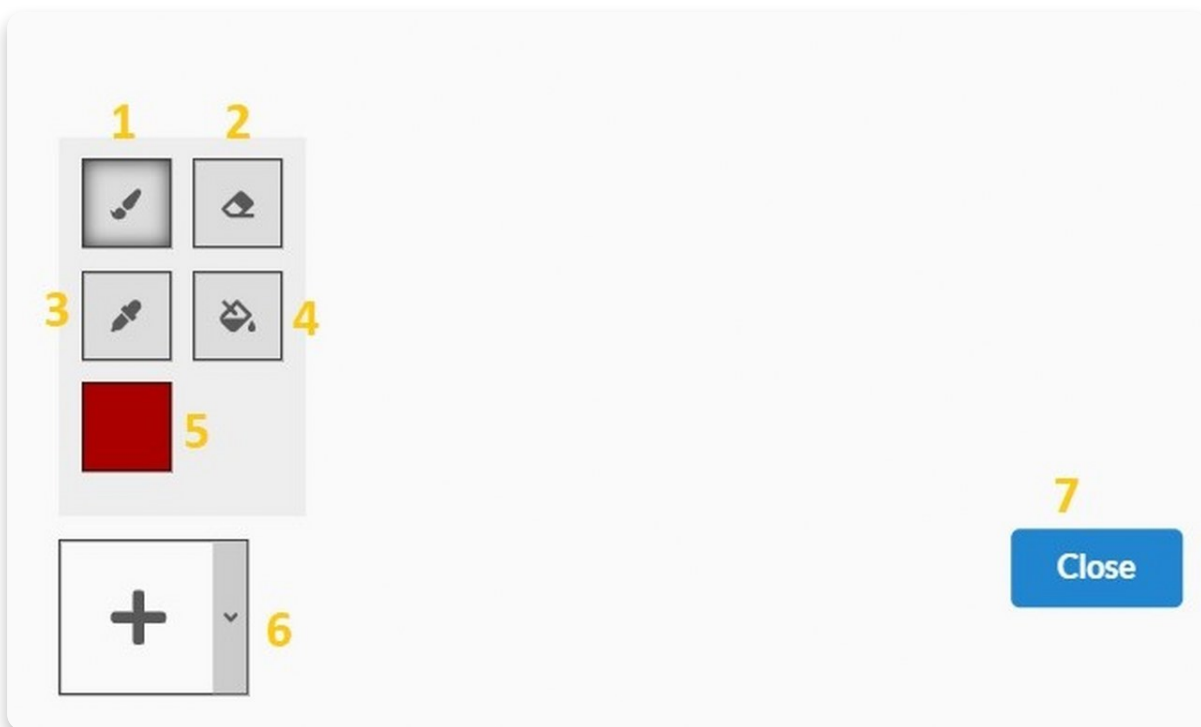
Find a **brush icon** on your **Toolbar** first.



A window will open, as shown in the photo below:



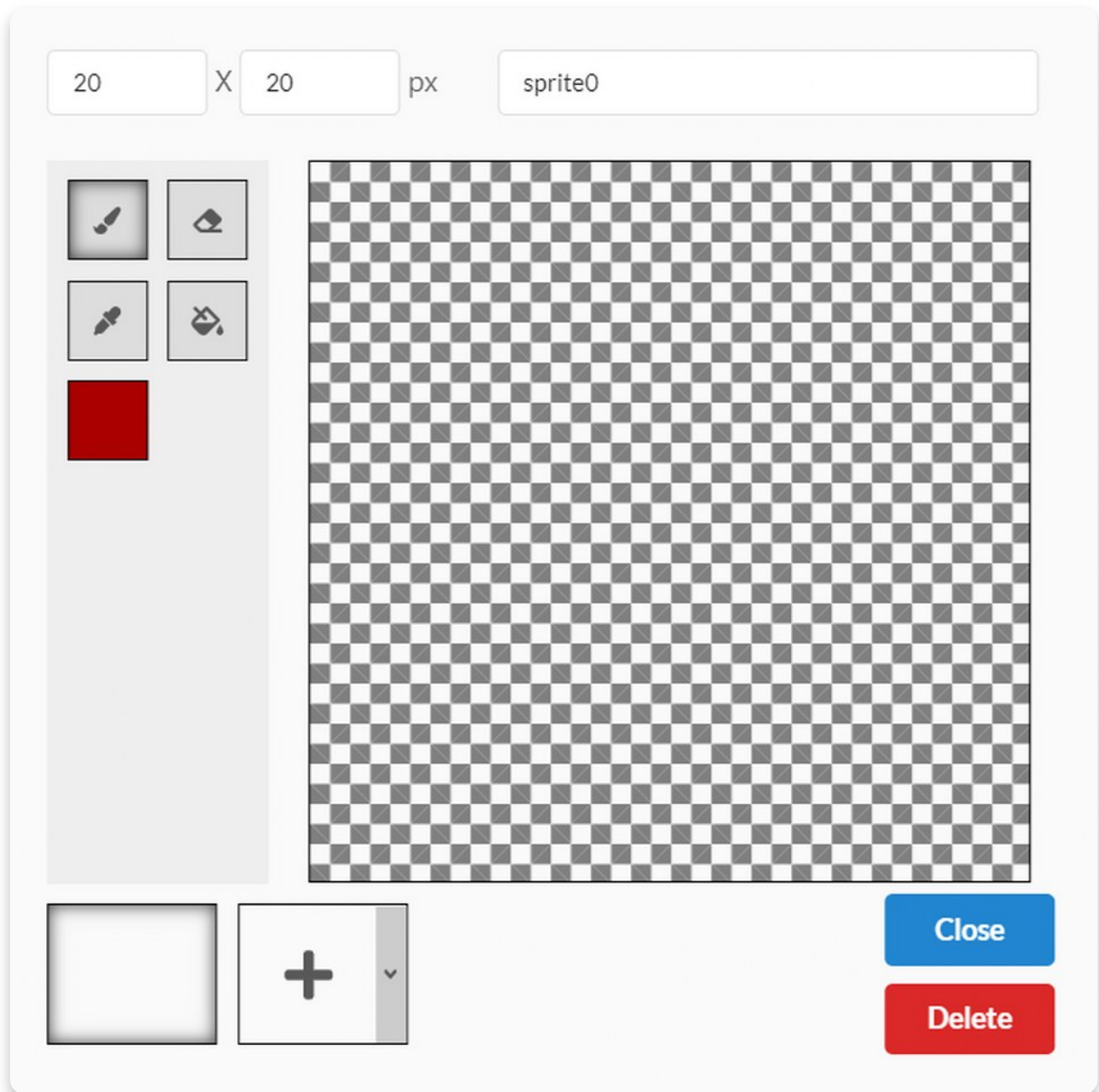
Let's see what each icon means.



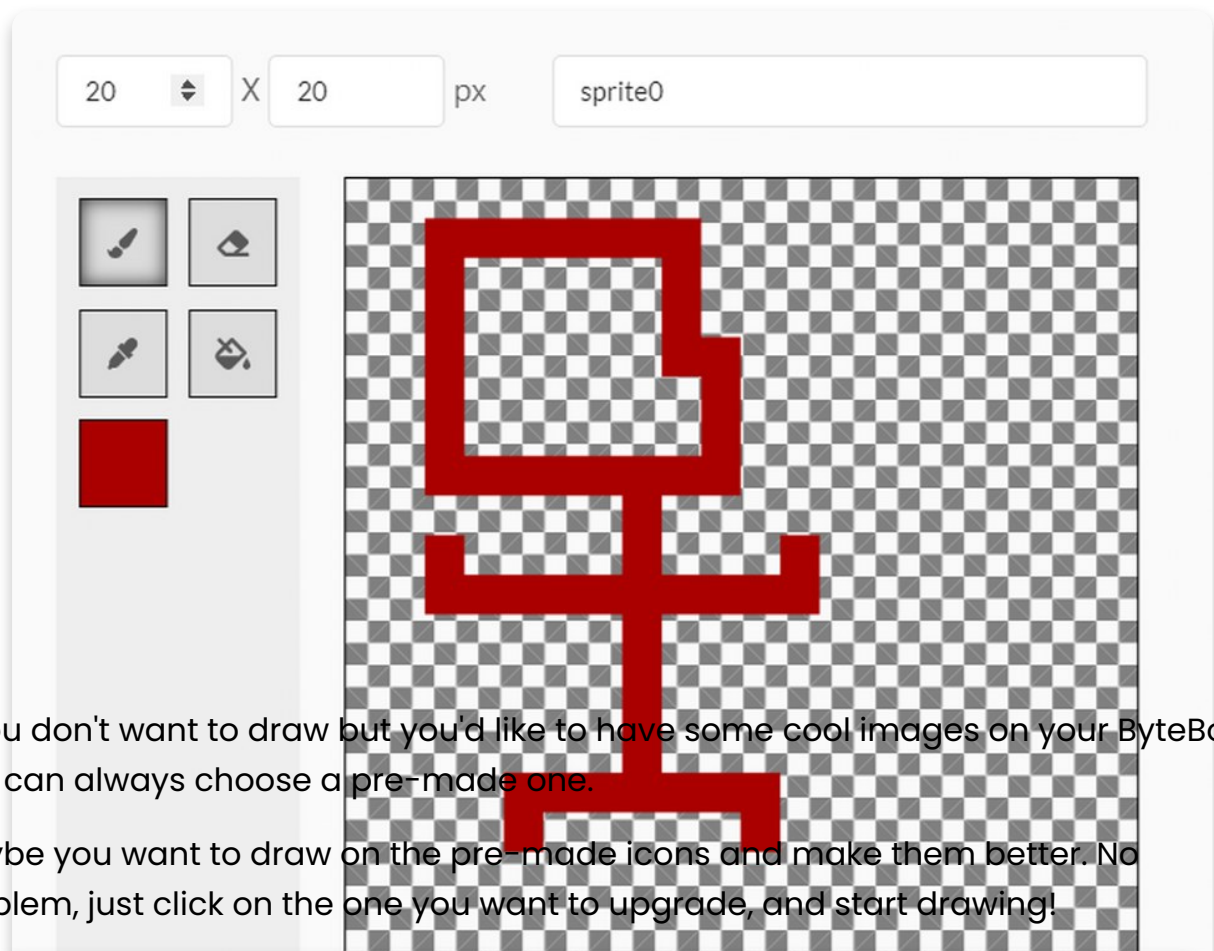
1. **Paintbrush** – you'll use it for painting whatever you want.
2. **Eraser** – you'll use it for erasing mistakes you make.
3. **Color picker** – pick a color from the picture and draw.
4. **Fill in with color** – fill the image or part of an image with a different color.
5. **Color palette** – here, you can choose which color you will use for drawing.
6. **New picture** – if you click on a plus (+) sign, you will get a blank window for your new picture. But if you press an arrow pointing down, you will see all the pre-made images.

7. **Close** - once you're done with the drawing, press the big blue button saying "Close".

This is the blank window you'll see at the beginning of the drawing.



You can paint whatever you want, and later on, you can put it on your ByteBoi's display.



If you don't want to draw but you'd like to have some cool images on your ByteBoi, you can always choose a pre-made one.

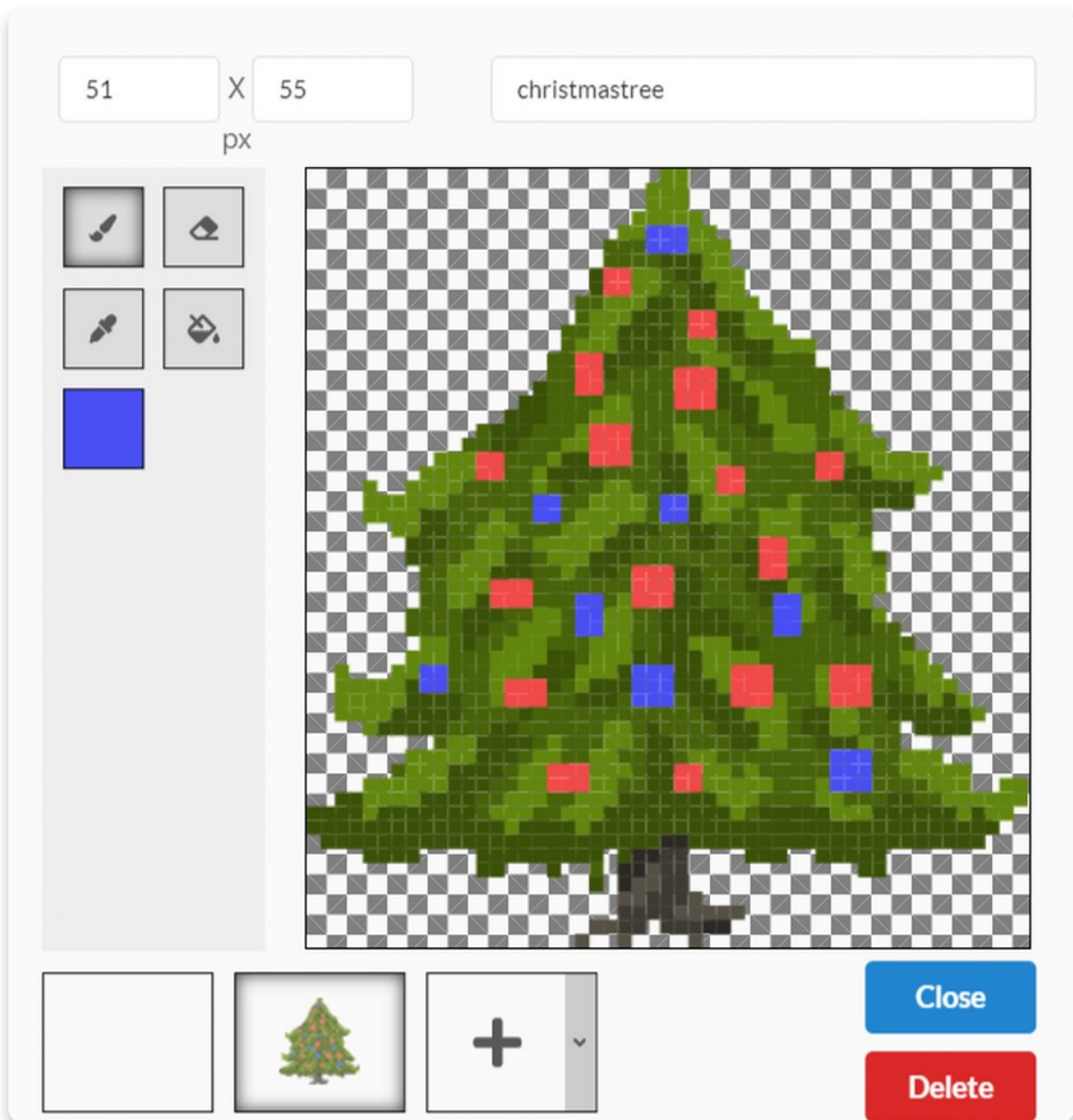
Maybe you want to draw on the pre-made icons and make them better. No problem, just click on the one you want to upgrade, and start drawing!



For example, I chose to draw a Christmas tree but didn't want to draw a tree from scratch. So, I took a premade tree and drew Christmas ornaments.

Also, I didn't want my tree to have a generic name, so I renamed it to christmastree.

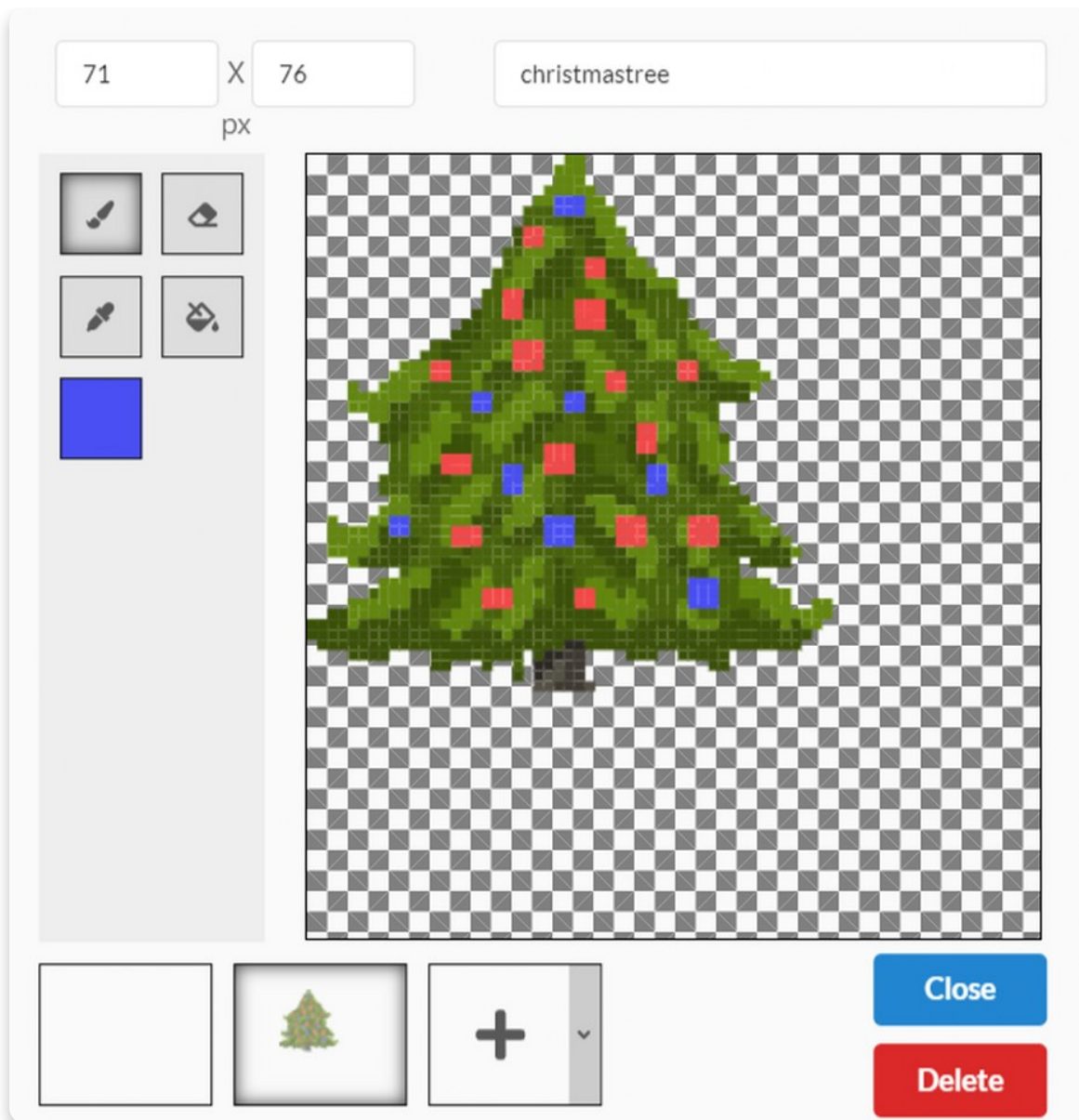
If you want to rename your drawing, you should know that its name cannot begin with a number, and there can't be any spaces between words.



One more tip is you can change the resolution of your sprite in the upper left corner.

The resolution is the size of a sprite, and it's measured in pixels.

A pixel is the smallest controllable element of a picture represented on the screen.



You can play with this some more, but now we're going to code for real!

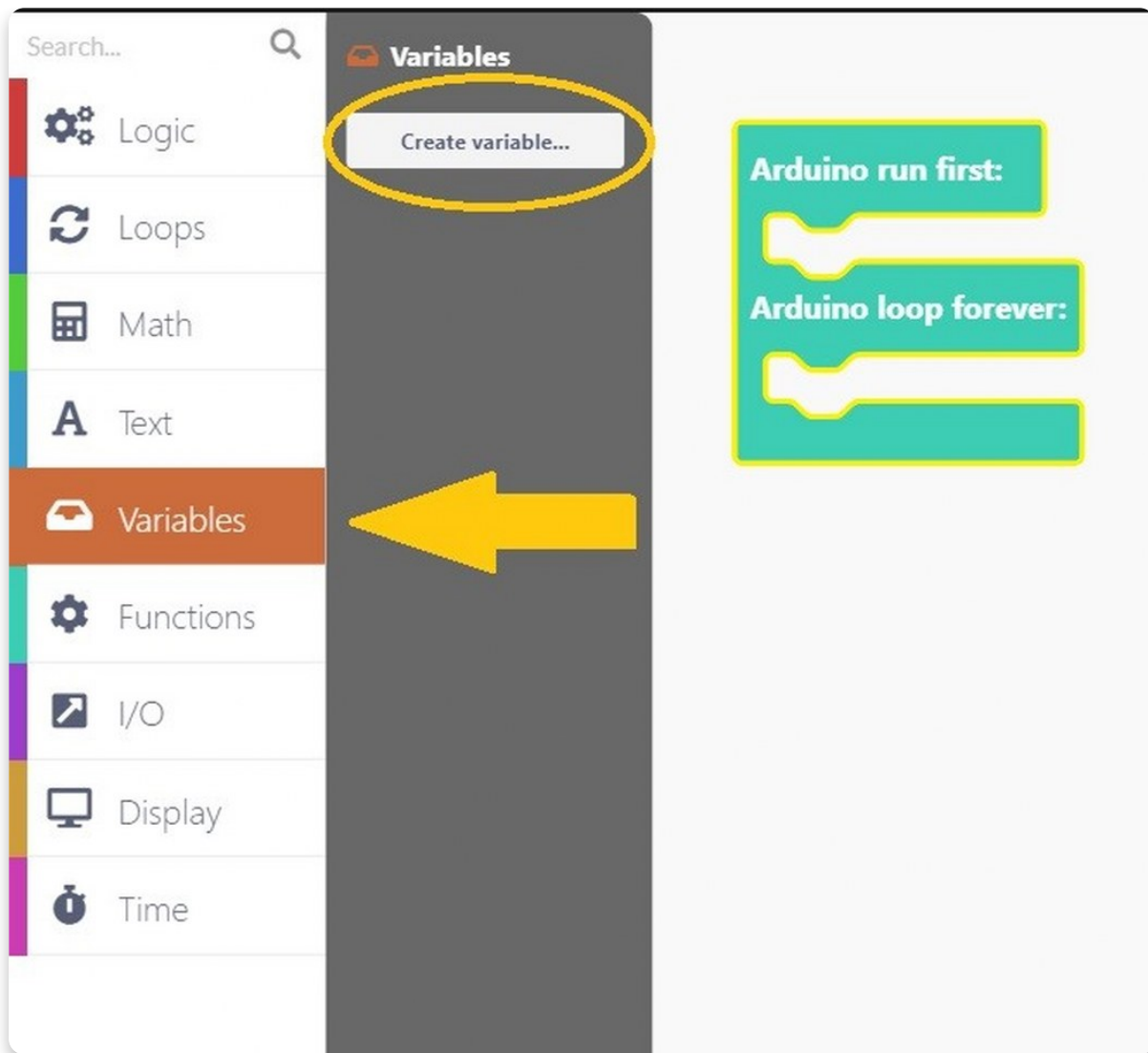
Let's play with the display!

For the next step, let's go a bit further and do something fun on display.

The first thing we are going to use are **variables**.

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 values of a variable.

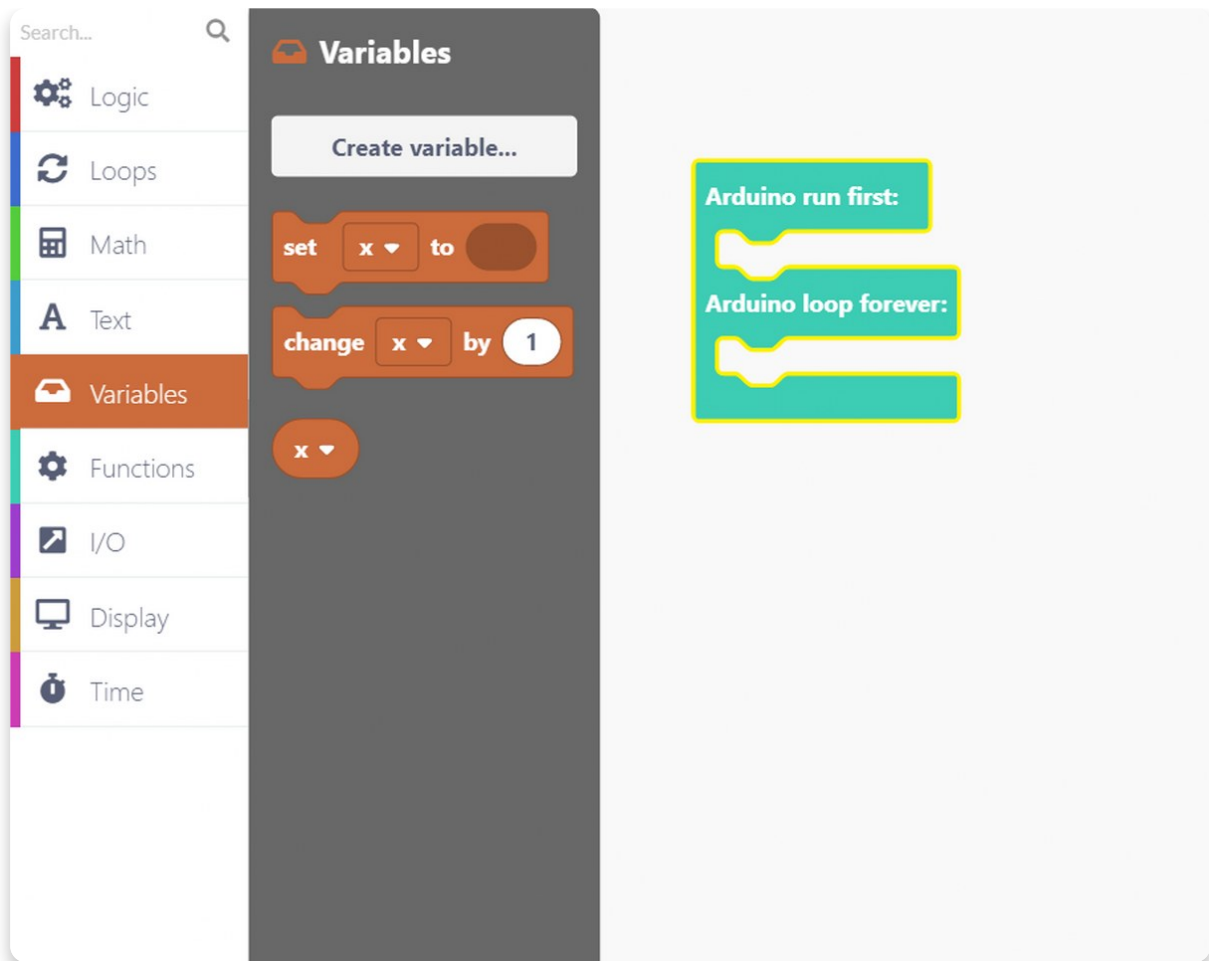
Firstly, let's create a variable. Find the section named "Variables" and press the "**Create variable...**" button.



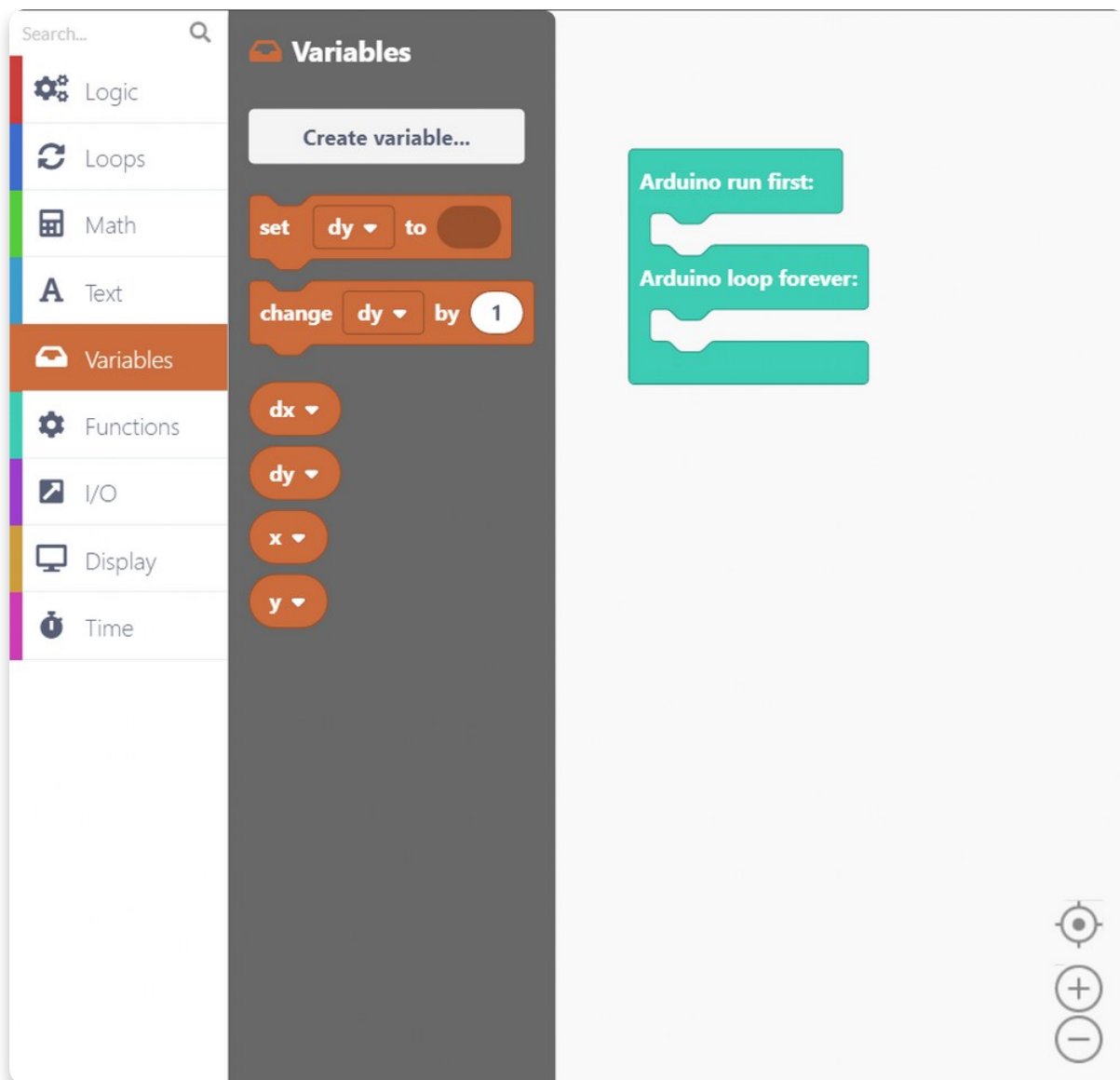
You need to give your variable a name.

I am not that creative, so I will name my variable "x" (just a single letter x).

We have a variable now. Great!

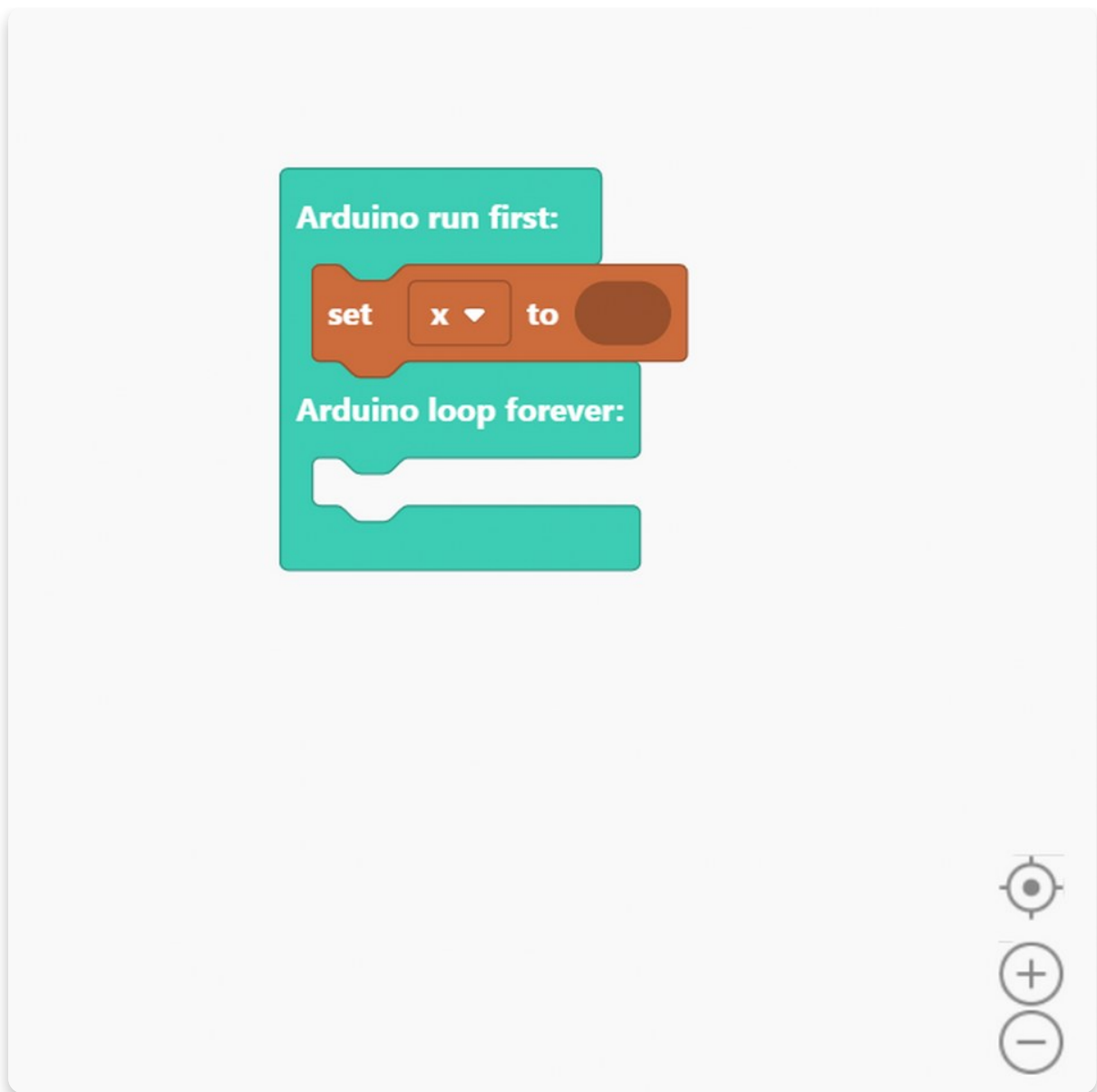


We can make all the variables we will need for this sketch immediately.
Just repeat the same process you did while making the variable "x".



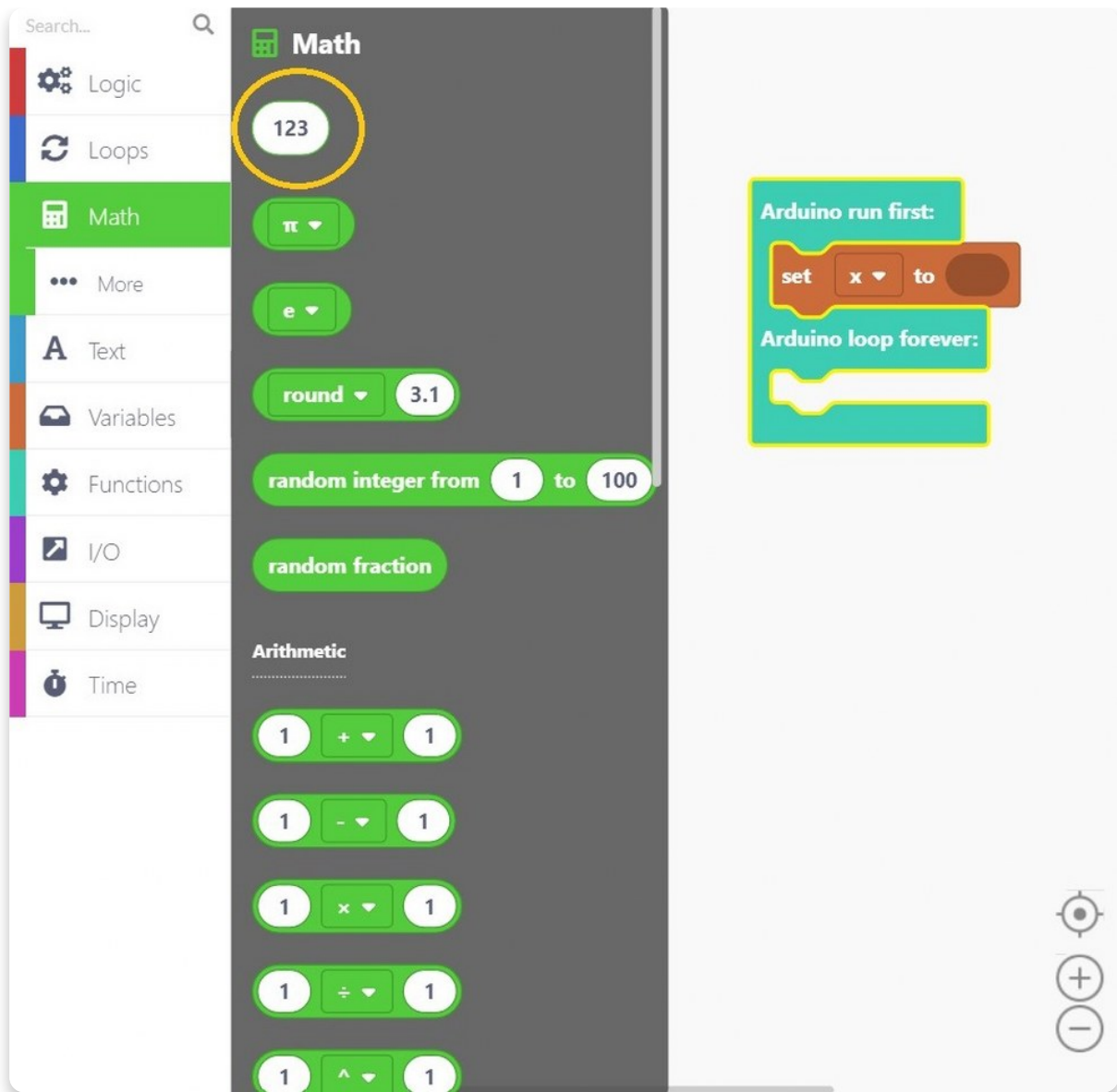
When we create a variable, it's undefined – it has no value. We must set a value for every variable when our computer program starts. That's why you'll need the "**set variable**" block.

Put the "**x**" block in the "**Arduino run first:**" branch.



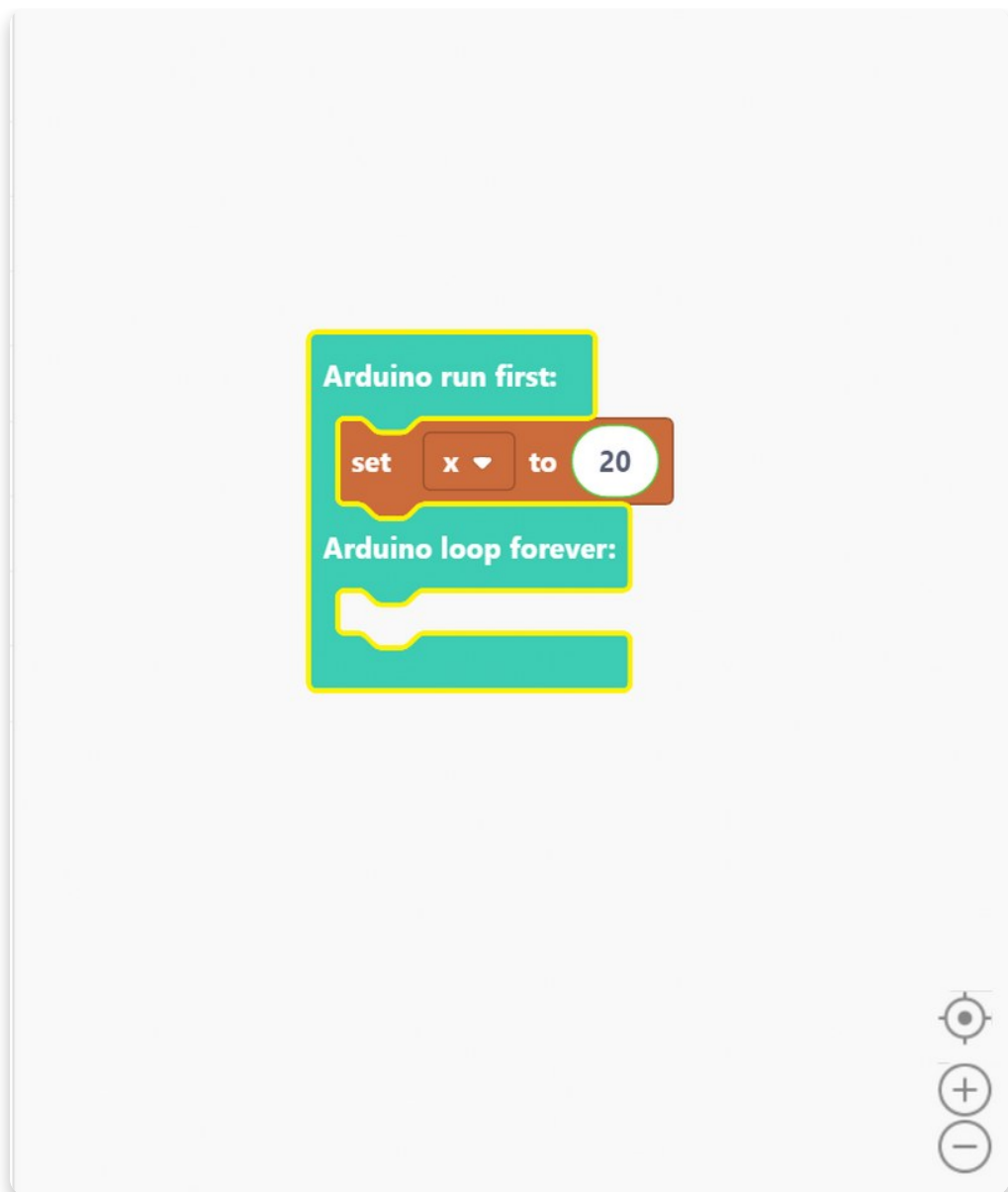
You need to define the value to which you want to set the variable.

Find this block named **"123"** in the **"Math"** section. This block is a numerical value block, and you can type in the numerical value you want once you drop it onto the drawing area.



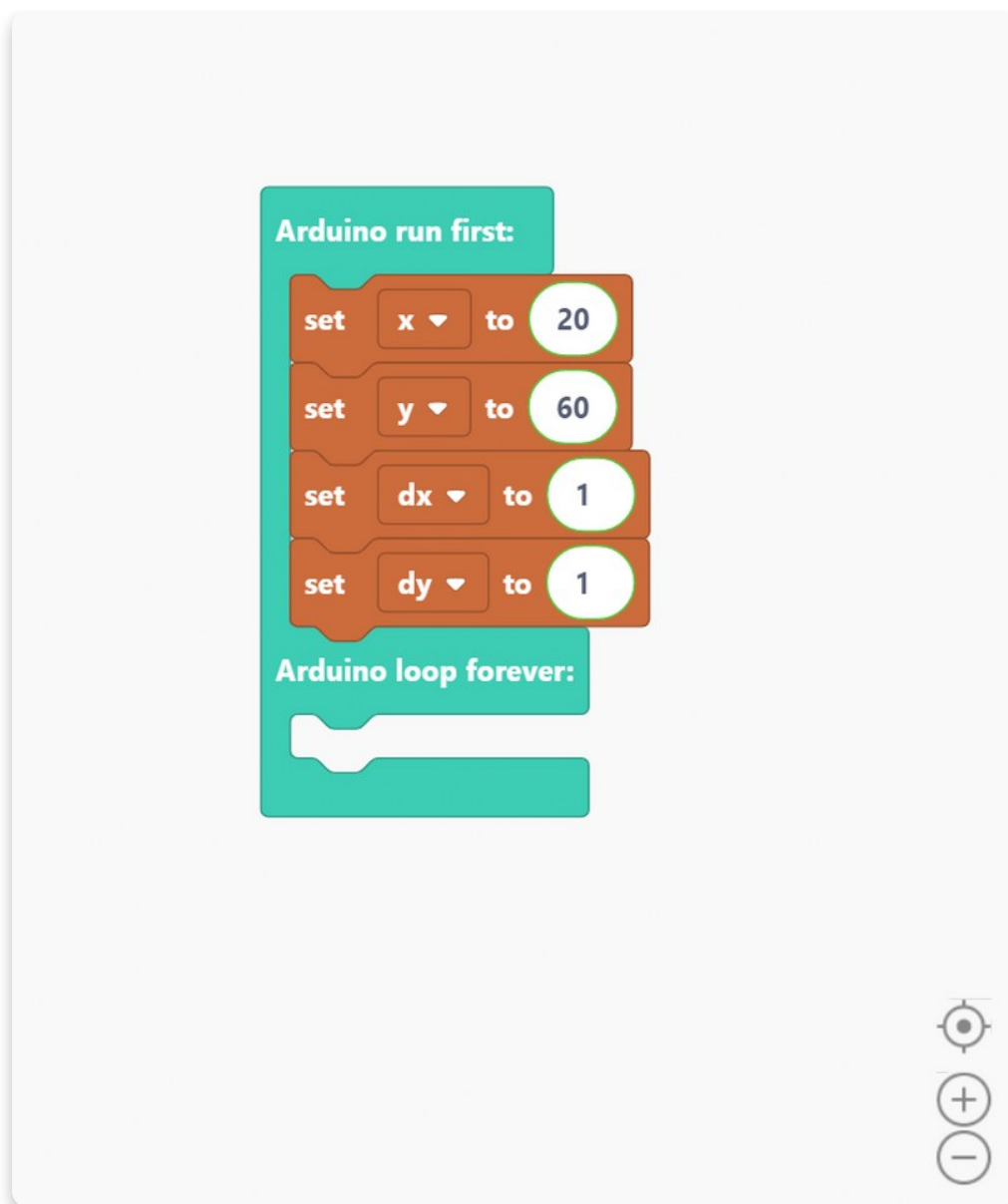
Place the block as shown in the photo below.

Now click on the block and type in the value. Set the **numerical value** of the block to 20. You can do this by simply typing the number 20 on your keyboard.



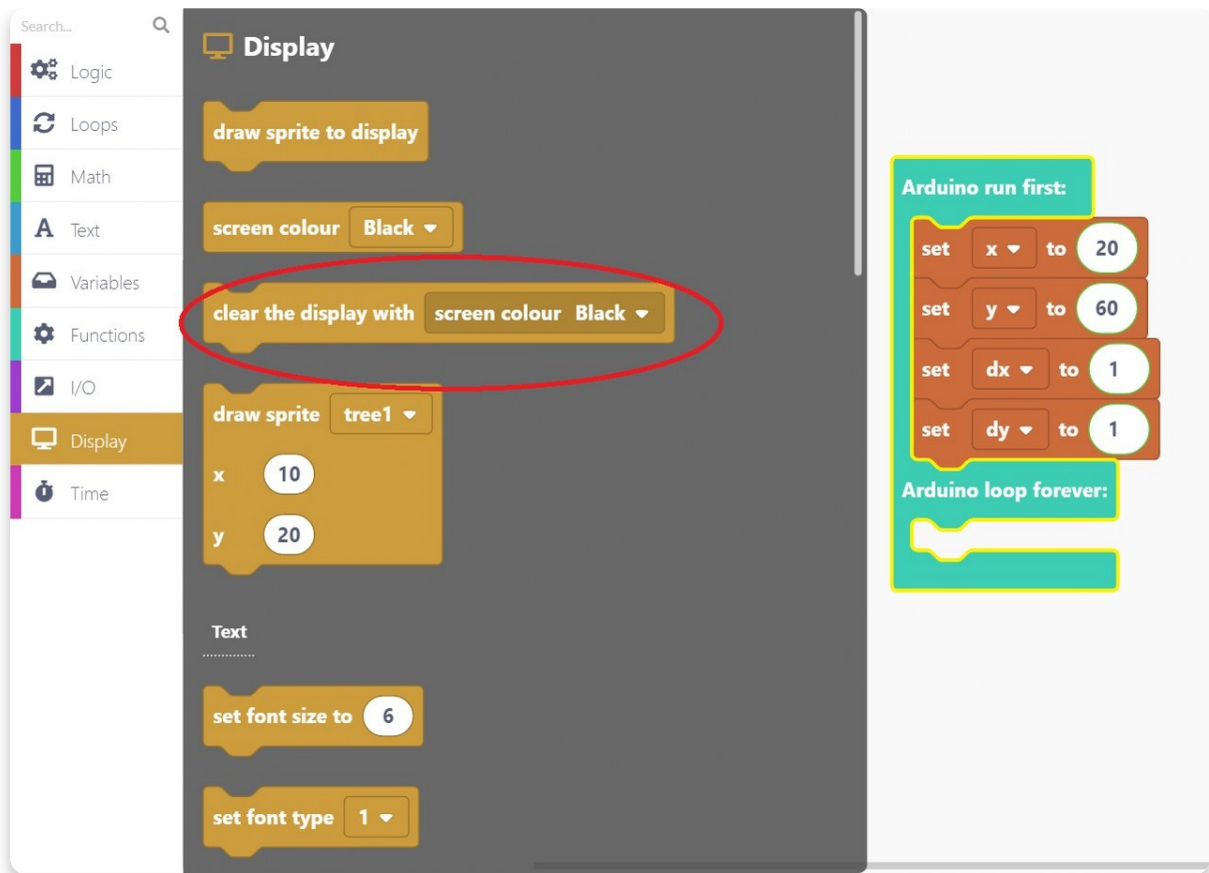
Now, repeat this process for every other variable you made. But, make sure to add the correct numerical value for each variable.

By doing this, we defined the position of the ball on the screen.



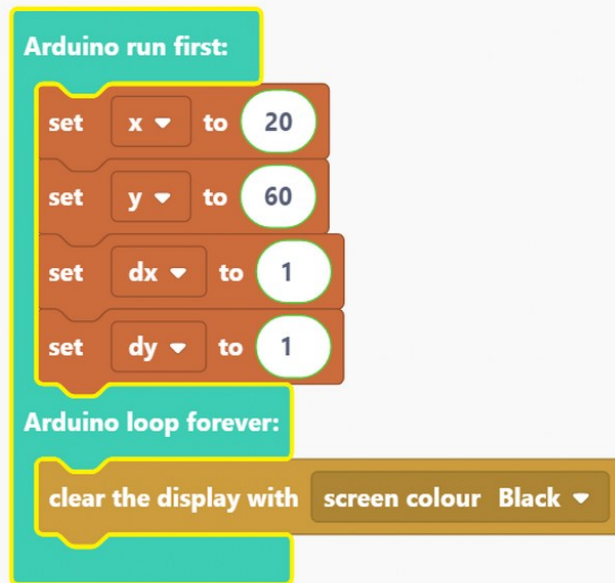
Now that we have variables created and set to correct numerical values let's make a drawing we want to see once we run this code.

You need to find this yellow block named "**Clear the display with screen color Black**".



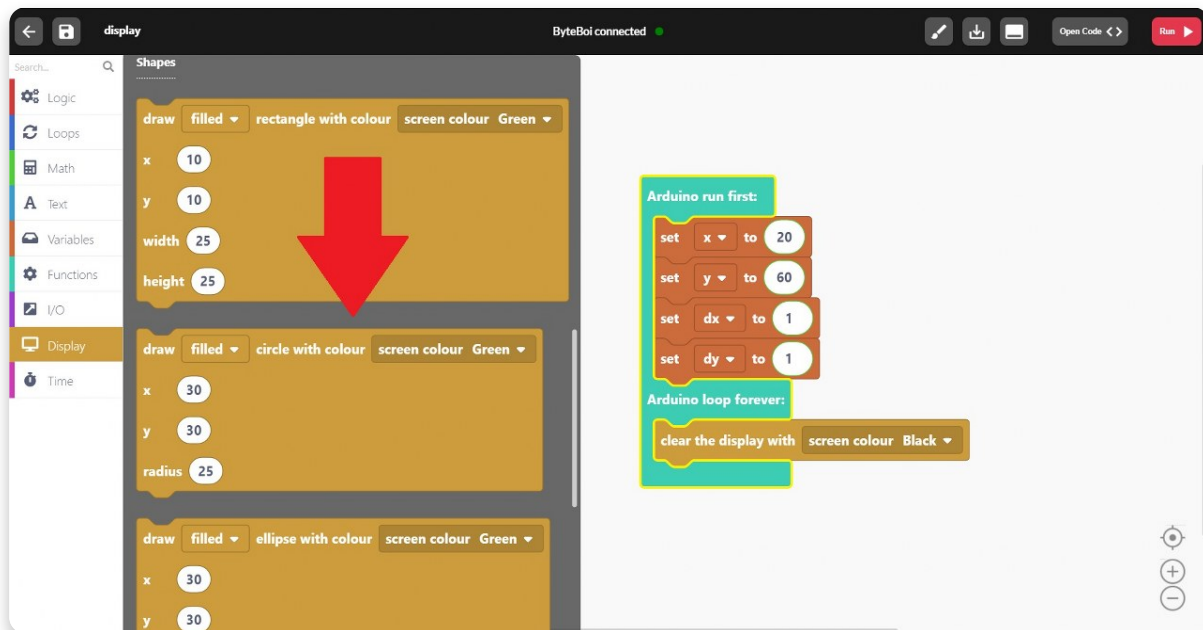
We're going to use the **loop()** part of the code.

Put that block in the "**Arduino loop forever**" branch.



Using this block, we make sure the background of our screen is black once we run the sketch.

Now you need to find another yellow block named **"Draw filled circle with colour screen colour Green"**.



Place this block under the "**Clear the display**" block in the "**Arduino loop forever:**" branch.

Set the radius value to 5 by simply writing the number 5 on your keyboard. As you probably guessed, the radius value defines the size of your ball. Number five in a radius represents a number of pixels of radius.

After that, open the Variables and choose the blocks that only say "**x**" and "**y**".

You will drag them in the circles displaying "**30**". By that, you defined the ball's position using previously made variables.

Your sketch should look like this:

Once again, open the Display section, and find the block called "**Draw sprite to display**".

Put it under the block you used for drawing the ball.

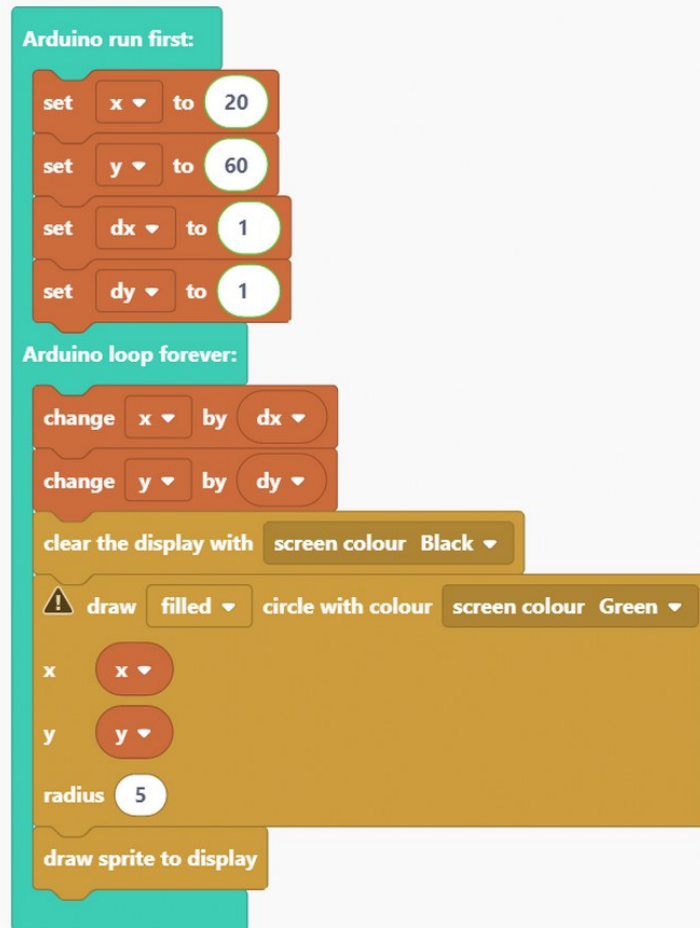
We have to put the "**Draw sprite do display**" block to ensure that our ball appears on display.

Now that we drew the ball, we should control its **motion**.

Firstly, go to Variables, and choose a block called "**change x by 1**". You'll make two of those for the "**x**" and "**y**" variables.

Instead of number 1 in the circle, we will drag variables "**dx**" and "**dy**".

By doing that, you'll increase the value of "**x**" and "**y**" variables by the value of "**dx**" and "**dy**".



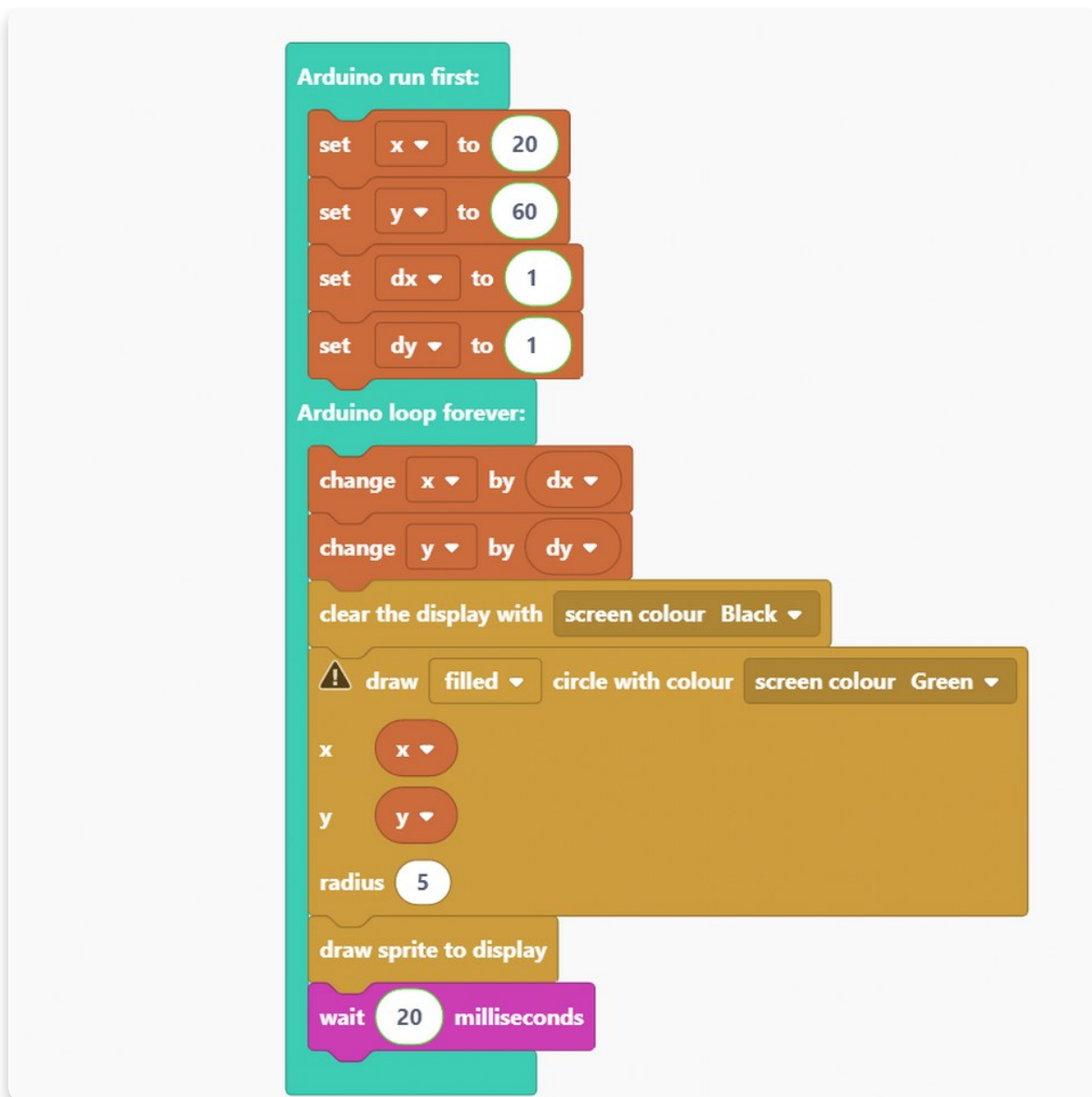
Let's use a time-related block.

Find the block saying "**wait 1000 milliseconds**", drag it to the end of your sketch, and write 20 instead of 1000. You can choose whatever time you want.

With this block, you make the program wait for a certain number of milliseconds. A **millisecond** is a thousandth of a second.

One thousand milliseconds is actually one second, and 20 milliseconds are 0.02 seconds.

If we put 20 instead of 1000, the ball will move for one pixel every 0.02 seconds.



Let's move on to the "**Logic**" section!

Drag the first one you see between the "**change**" and "**clear**" blocks.

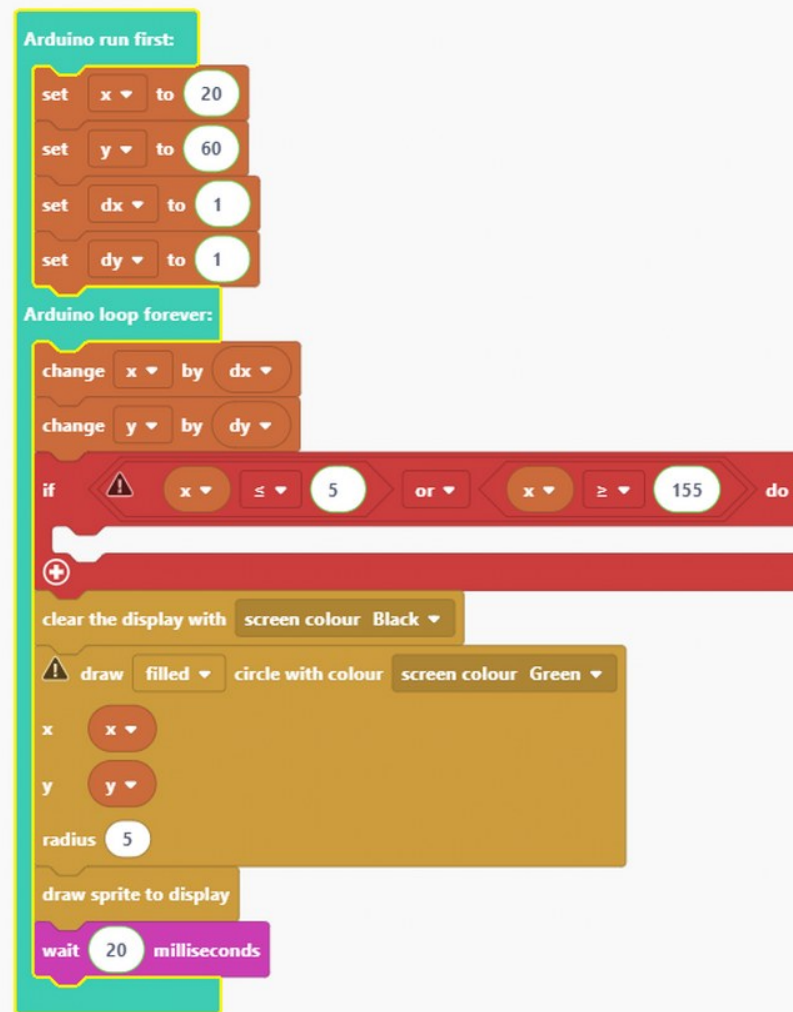
Up next, we'll need a **Boolean block** – the one that says "**or**".

Boolean is a binary variable that can have one of two possible values, 0 (false) or 1 (true).

Now we'll need a **comparison block**. This block is usually used for comparing the value of a variable to a fixed value (i.e., let's see if the variable x **is less than or equal to number 5**).

Place the variable we've created on the left side of the comparison block.

Take the numerical value block from the math section and place it on the right side of the comparison block.

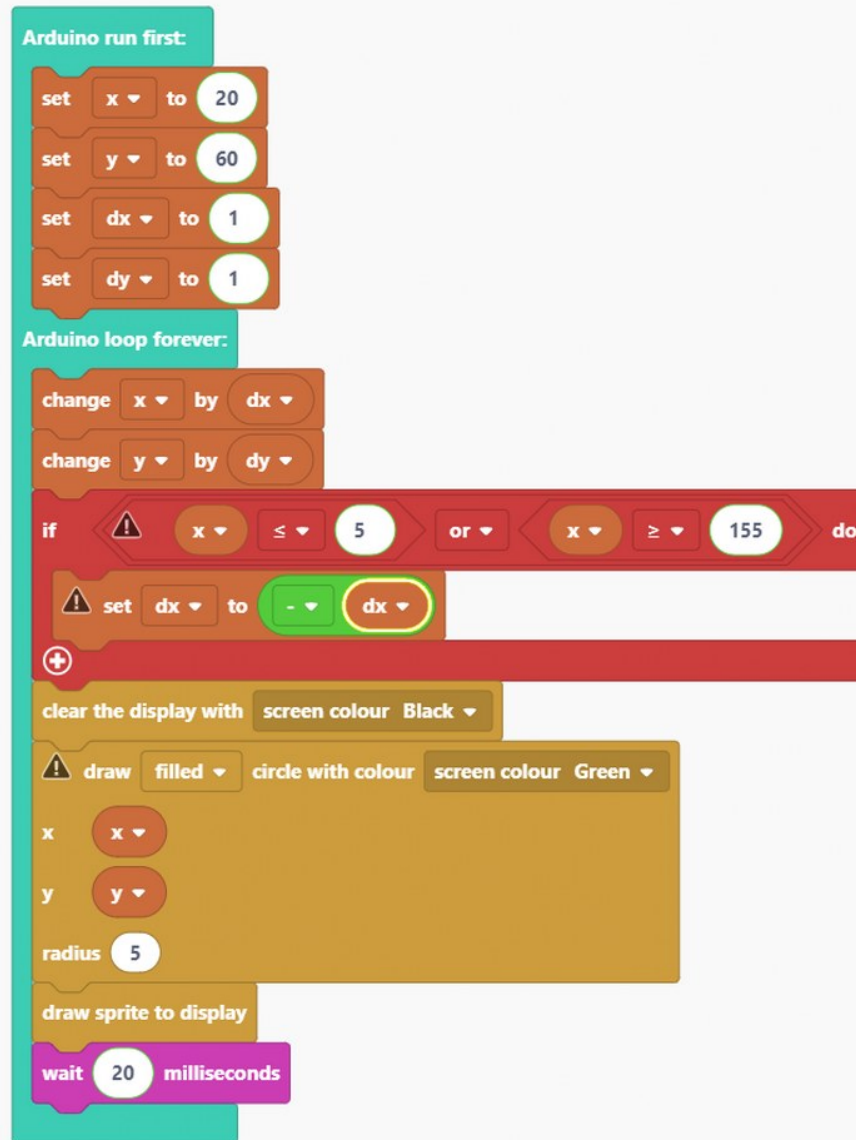


Also, we have to define a numerical value of the "**dx**" variable, so we need to take a "**set dx to**" block.

Find **single operand** blocks in the **Math** section.

An operand is the part of a computer instruction that specifies what data is to be manipulated or operated on.

Choose the circled one from the photo below:



Now you'll have to **duplicate** the last couple of blocks. You can do that by clicking on the right button on your mouse and choosing "**duplicate**".

We'll need the second group of blocks for the "**y**" variable.

Check out the values we put for the "**y**" variable:

Now hit the big red "**Run**" button and **see the code executed on your device**.

Cool, right?

Sound effects

The next fun thing we'll learn is how to increase and decrease the tone frequency on ByteBoi using two buttons.

You can open a **new sketch**, and let's start!

First, you'll need to create a new **variable** and call it "**frequency**".

Place a "**set frequency to**" block into the "**Arduino run first:**" branch:

Find this block named "**123**" in the "**Math**" section. , As we mentioned before, this block is a numerical value block, and you can type in the numerical value you want once you drop it onto the drawing area.

Let's put the frequency value to 1000.

Now that we have a variable created and set to 1000 let's change the variable when a certain event is triggered.

When you press ByteBoi's up and down buttons, we change the variable.

Luckily, we have a specific block defined for that, and it's under the **I/O** section. I/O stands for "**Input/output**". ByteBoi's buttons are the so-called input devices because they send an electrical impulse to ByteBoi's computer when triggered. ByteBoi's display is an example of an output device because ByteBoi sends signals to it to display information.

You need to find this purple block named "**When button up gets pressed**".

Place the block onto the drawing area.

You can duplicate the purple block.

We will need one of them to increase the tone frequency and the other to decrease it.

Now let's change the variable's value by 100 every time you press ByteBoi's buttons.

This is done by a block "**change frequency by**".

Place that block into the purple "When button up (down) gets pressed" event block.

Make sure the value in one block is positive and in the other negative.

The next block we will need is "Play tone with Hz for milliseconds" from the I/O section.

We'll put the "frequency" variable in the left circle and the number 500 in the right circle.

Now hit the red "Run" button, and don't forget to save your code!

Once your code is compiled, you can press the up button on your ByteBoi and increase the tone frequency or use the down button and decrease it.

This was fun. Let's see what else we can do.

Light show!

Now that you know how to play sounds and draw on display, let's **play with LEDs**.

LEDs are light-emitting diodes, and your ByteBoi has four red, blue, and green LEDs located in the back.

Here you can see the input test for your LEDs:

Start fresh by creating a new sketch!

Firstly, we'll do a new **variable** and name it "**color**".

We'll drag the "**set color to**" variable into the "**Arduino run first:**" branch.

Now, let's take the blocks we use to change the variable when a certain event is triggered. We'll use the buttons for triggering the LEDs.

There are **directional buttons** on the **left side** of the console, while on the **right side**, there are **A and B buttons and Menu (Select) buttons**.

As you remember, those are the purple ones from the **I/O section**.

Drag the "**when button-up gets pressed**" and place it on the drawing board.

Now let's change the variable "**color**" value by 1 every time you press one of ByteBoi's buttons.

To do so, you'll need to pick a "**change color by 1**" block from the **Variables section**.

Drag a block and place it inside of the purple I/O block like shown in the photo below:

We won't change the numerical value needed for pressing the button "**up**" because we need to increase the value with every new button.

The next thing you can do is duplicate the I/O block and the "change color by _" block.

You can do that by simply clicking the right button and choosing **Duplicate**.

We will need six of these blocks because our game console has six buttons.

This is how your CircuitBlocks' sketch should look by now:

Since we duplicated all of the blocks, they all have the same button and the same numerical value written on them.

We have to change that for our code to work!

First, we need to set every purple block to a different button and change every button's numerical value. That way, you'll ensure that pressing every button releases unique color from your LEDs.

Take a look at the photo below:

Now that we have assigned functions to these six buttons, we will use the **Select** button.

In CircuitBlocks, this button is called the **Menu** Button.

Once again, drag and drop the "**When button up gets pressed**" block from the I/O section to the drawing board.

Now, we'll define what the Menu button does. Let's add some logic to it. From the **"Logic"** section drag, and drop the **"if-else"** block inside of the **"When button Menu gets pressed"** block. Just like this:

Now, drag the **"set color to"** variable and the **123 blocks** from the **Math** section and place them inside the **Logic** block.

Let's find out what the functions are!

The new thing we will learn right now is how to make a new function.

Functions are "self-contained" modules of code that accomplish a specific task.

They are crucial if you want to keep your code clean and fast.

They are little pieces of code that you can tie together and call in your code multiple times..

Every program is made out of two main functions that we've already mentioned
- **void setup and void loop.**

You can create a new function by clicking on the **"Create a Function"** in the **Functions** section.

This will open the **"Edit Function"** window. There, we'll name our function **"setLED"**. To do that, delete the text that says **"do_something"** and type in **"setLED"** instead. Then click the **"Done"** button in the bottom right corner.

Once you do that, the function will automatically be added to the drawing board. Just like this:

Now we'll define what this function does. Let's add some logic to it. Drag and drop the **"if-else"** block from the logic section. Place this block inside of the **"setLED"** function just like this:

In the **"True"** slot, drag and drop the **comparison block** from the **"Logic"** section.

On the right side of the block, place a variable block "**color**", and on the left side of the block, place zero (0).

Your new function will look like this:

Maybe you noticed that the logic block has minus (-) and plus (+) signs.

Right now, we'll use the plus (+) one. To add more "**ifs**" on your logic block, click on the little '+' icon on the bottom left.

Click the '+' icon six times.

The logic block should look like this:

Duplicate the comparison block you used at the beginning of the logic block, and place them in an empty circle between "**if**" and "**else**".

Numeric values should be set from 0 to 6, just like this:

Now, go to the Functions section and choose the block saying "**call setLED**" (the one which looks like a piece of a puzzle).

Duplicate the "**call setLED**" block and place it into the purple blocks like this:

In the end, we'll have to drag and drop purple "**set RGB LED color**" inside the "**if-else if-else if-...**" block. As you can see, we put a different color for every button. Real light show it is!

Hit the red Run button!

ByteBoi will process your commands, and now you can press each button and get a different color from the LEDs.

Remember to save your hard work by pressing the save button on the top left.

Create your first game!

Let's try something more advanced now!

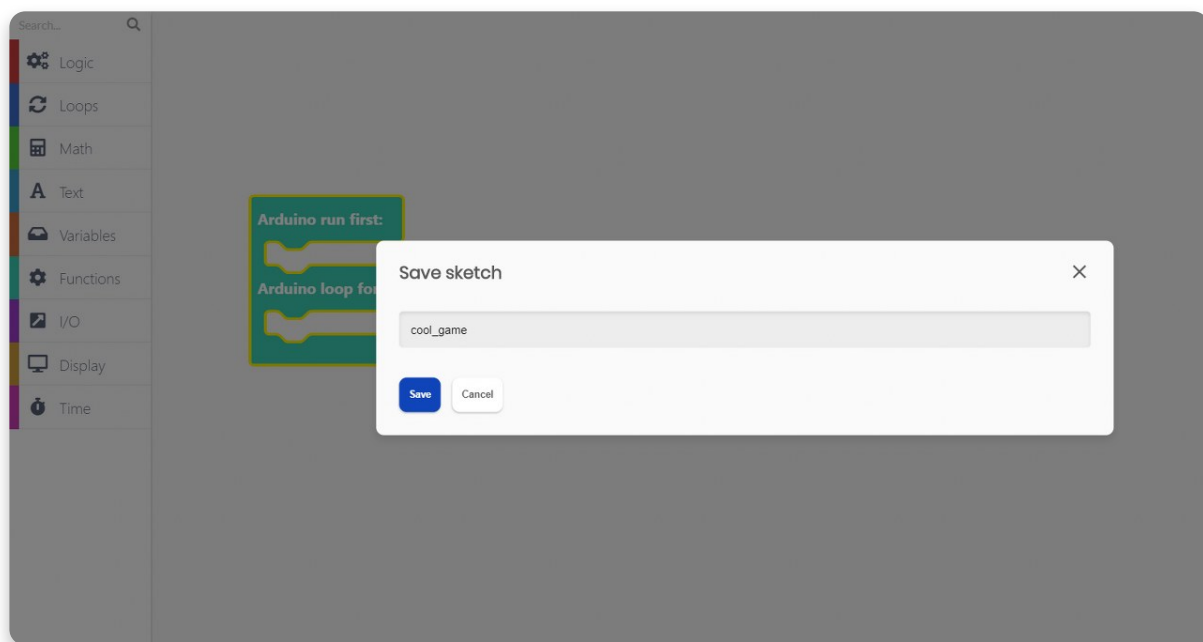
Creating a whole game is not an easy task. There are a lot of little details that need to be considered to make the game work as good as possible. When creating your own game, always set the goals at the beginning so that you can work from the bottom up with some structure in mind. Writing code without a bigger picture in mind can be a big problem later in code development.

The game that we'll create will showcase pretty much every feature we've gone through so far.

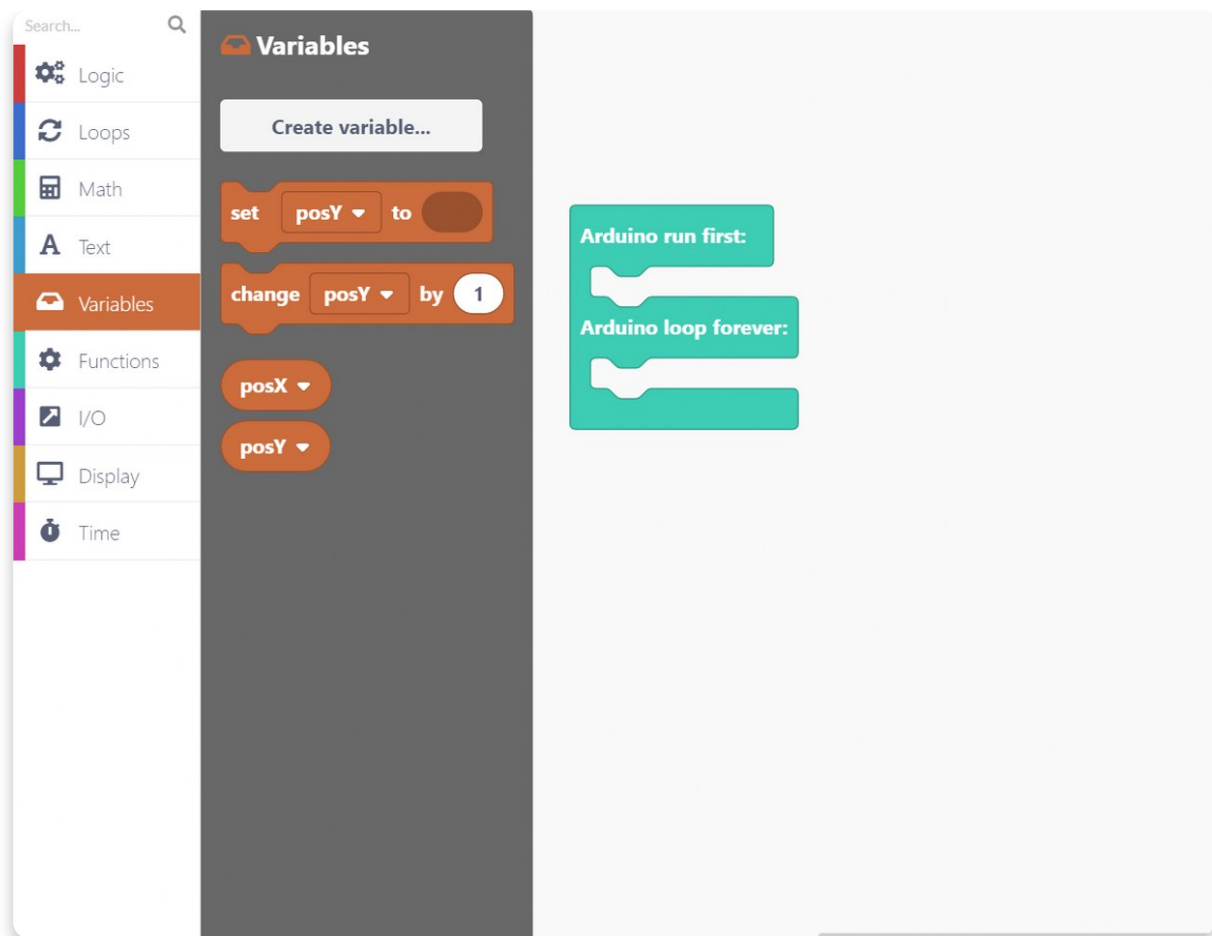
It will look like a simple video game, where you'll move a character that collects objects while counting a score.

Let's begin!

Start fresh by creating a new sketch. Save it right away and name it something snazzy (I've named mine "**cool_game**").



Now, let's make some variables and name them "**posX**" and "**posY**". Those will define your (player's) position on the screen.



Place them inside "**Arduino run first:**" branch like this:



We placed those variables in the "Arduino runs first:" branch because that's the position the ball will have when your ByteBoi turns on.

Find this block named "**123**" in the "**Math**" section. Type in the numerical value you want once you drop it onto the drawing area.

Write these numerical values in the circles:

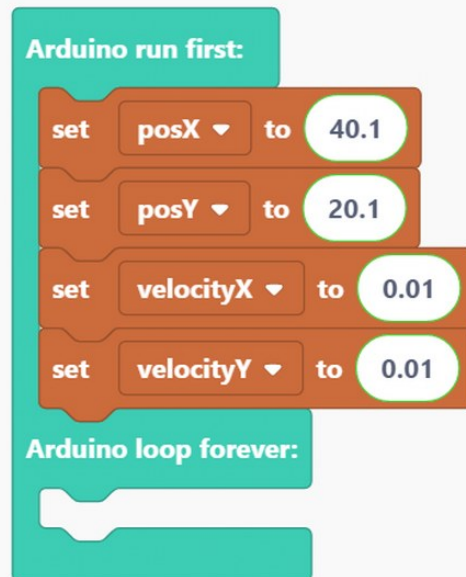


Create another set of variables that'll determine the speed of your ball.

Let's call them "**velocityX**" and "**velocityY**". Drag and drop the new variables under the variables we used for determining position.

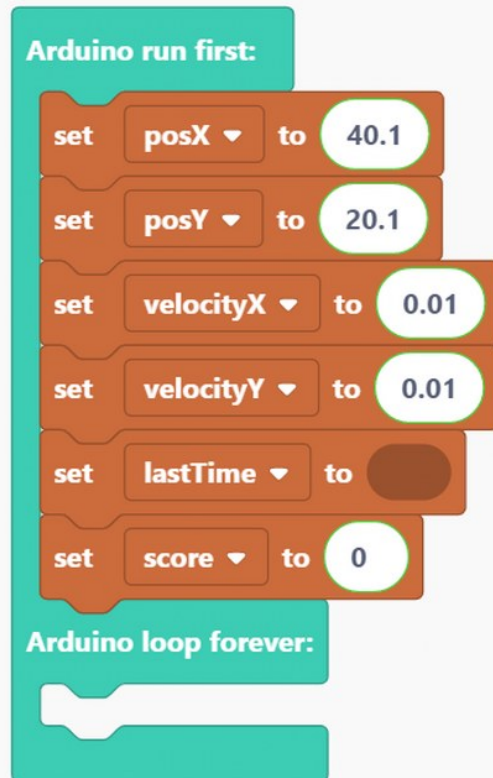
"**VelocityX**" means that this will be the velocity of the ball in the x-axis direction, and "**velocityY**" determines the velocity of the ball in the y-axis direction.

Make the numerical value of the velocity 0.01 for both X and Y.



Let's make a few more variables while setting up our game.

You can name them "**lastTime**" and "**score**". Of course, you should put the value of the "**score**" to zero (0) at the beginning of the code.



Drag and drop the "**lastTime**" block into the "**current elapsed Time (milliseconds)**" block.

Variable "**current elapsed Time (milliseconds)**" defines how much time has passed since the device was turned on.

You can find that block in the **Time section**.

Search...

Logic

Loops

Math

Text

Variables

Functions

I/O

Display

Time

Time

- wait forever (end program)
- wait 1000 milliseconds
- wait 100 microseconds
- wait 2 seconds
- current elapsed Time (microseconds)
- current elapsed Time (milliseconds)

Arduino run first:

- set posX to 40.1
- set posY to 20.1
- set velocityX to 0.01
- set velocityY to 0.01
- set lastTime to
- set score to 0

Arduino loop forever:

Arduino run first:

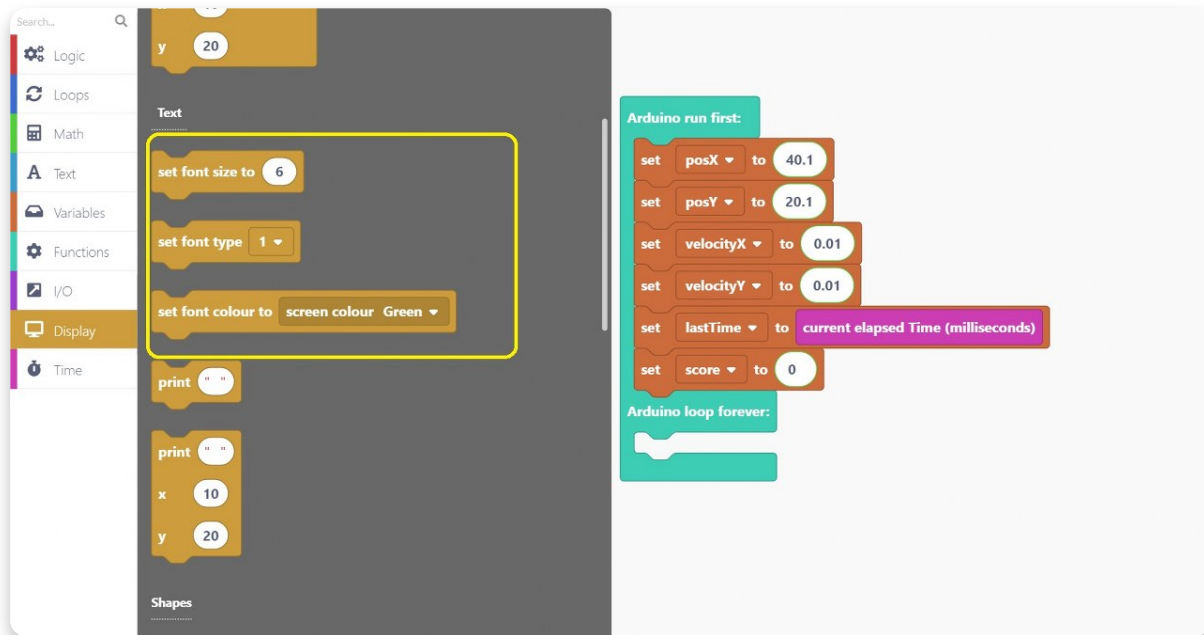
- set posX to 40.1
- set posY to 20.1
- set velocityX to 0.01
- set velocityY to 0.01
- set lastTime to current elapsed Time (milliseconds)
- set score to 0

Arduino loop forever:

The next fun thing we'll do is determine **the font size, font type, and color**.

If you are going to have any text on your display, the first thing you have to do is set font size, type, and color.

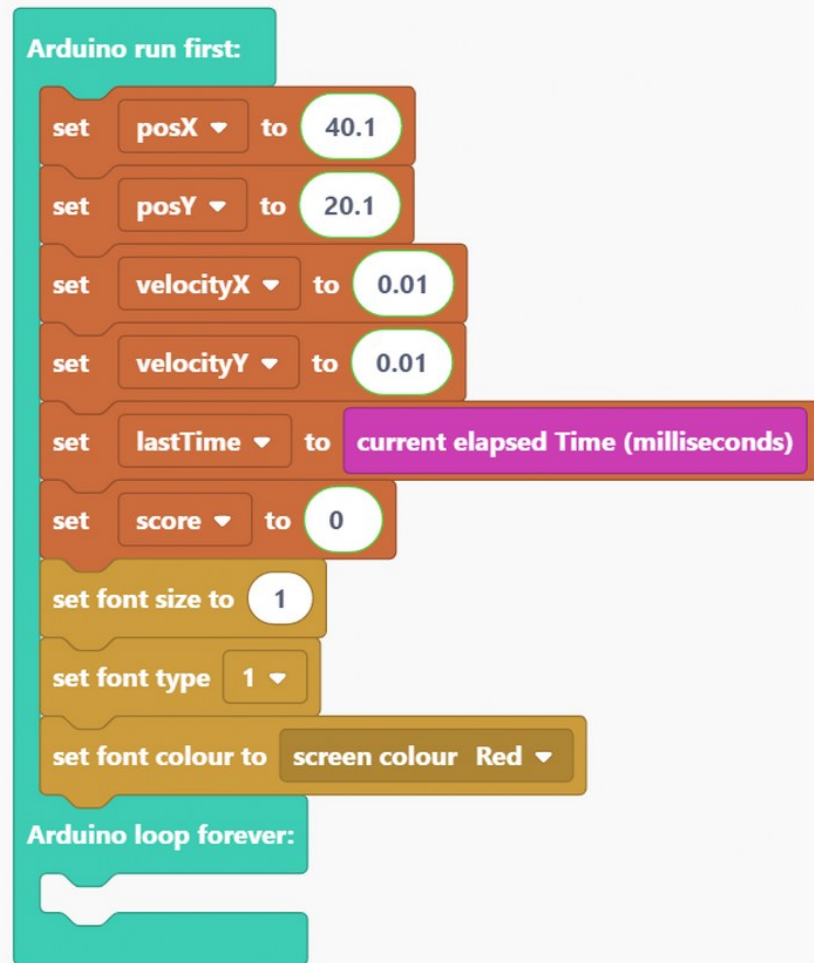
You'll find all of those blocks in the **Display** section, as shown in the photo below:



Place the font blocks under the variables you just made.

We chose the size and the type of the font to be one (1), and for color, we chose red.

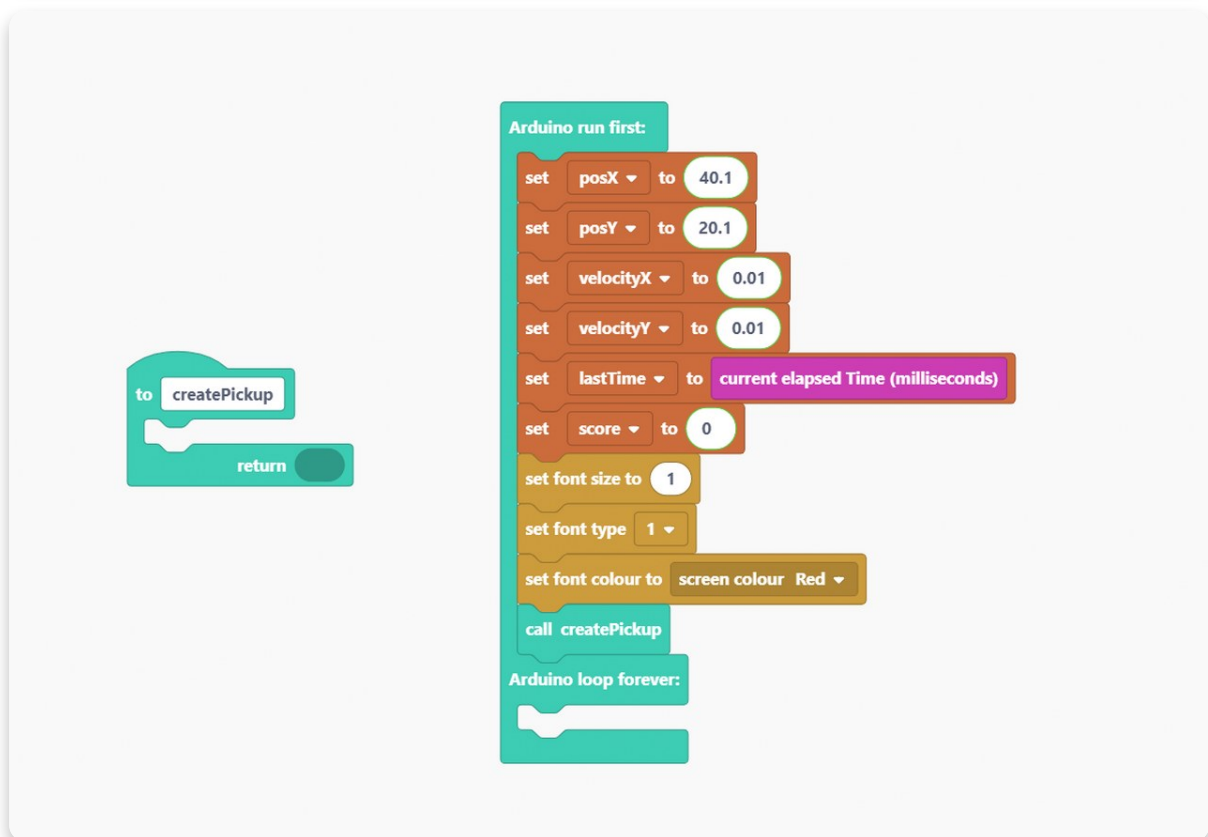
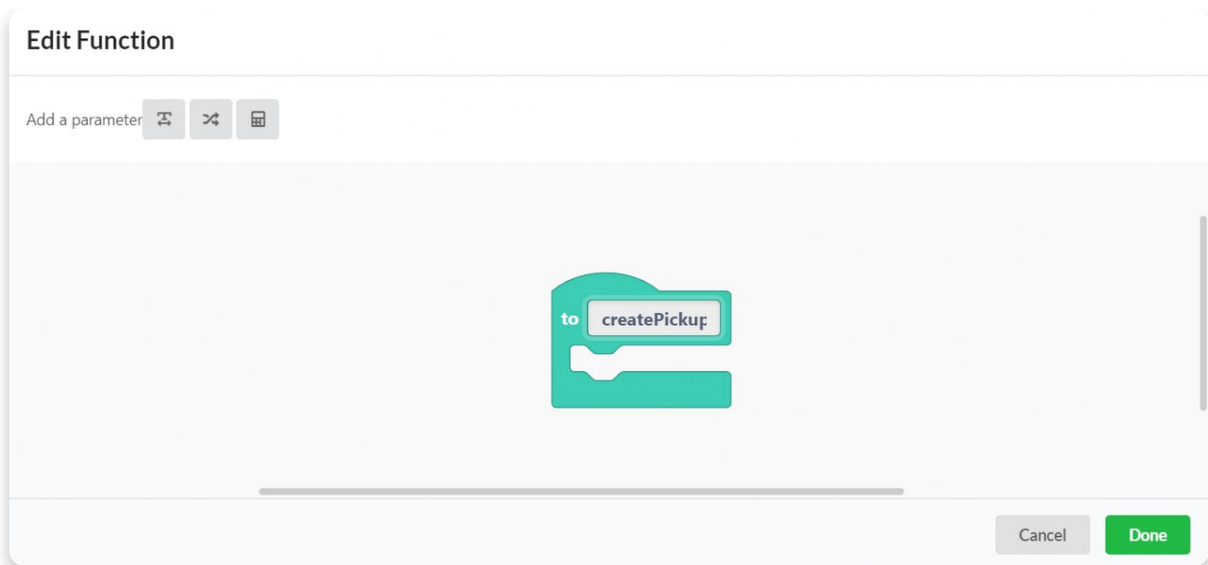
Font side one (1) is the smallest font size available.



We have to create a function if we want to execute some code.

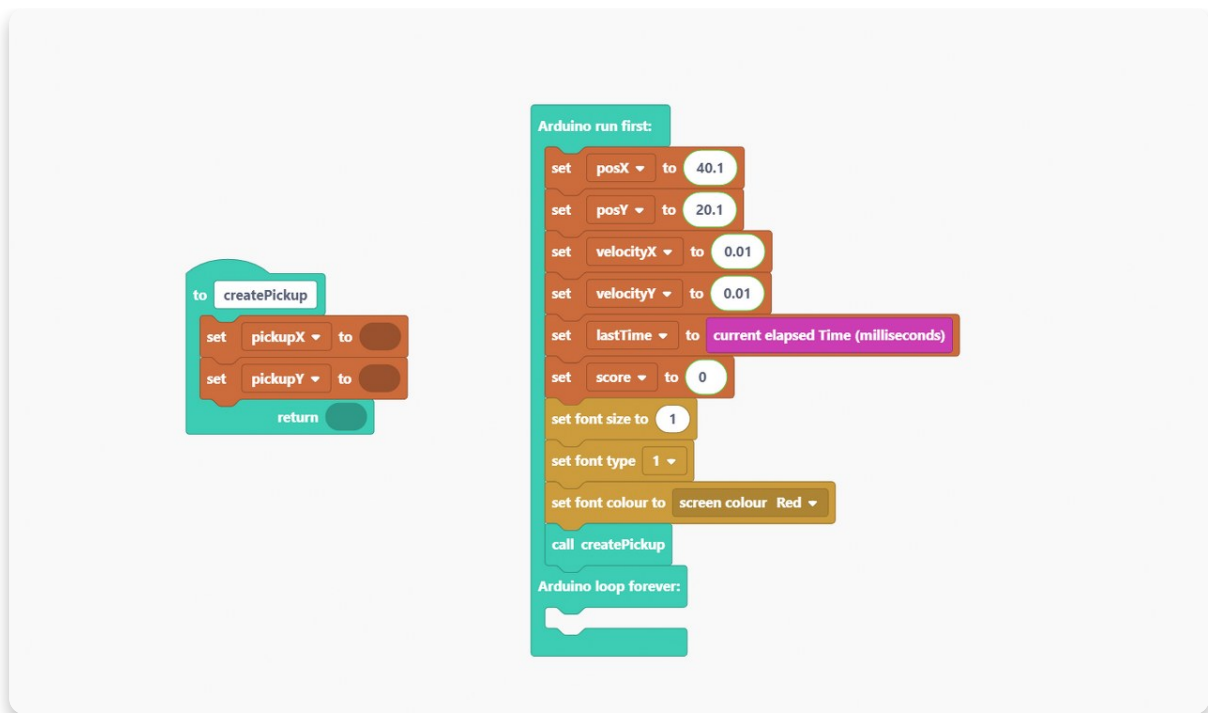
Let's name our function "**createPickup**" and place it under the "**set font color**" in the "**Arduino run first:**" branch.

By doing this, we have created a small ball that a player will have to collect at the game.



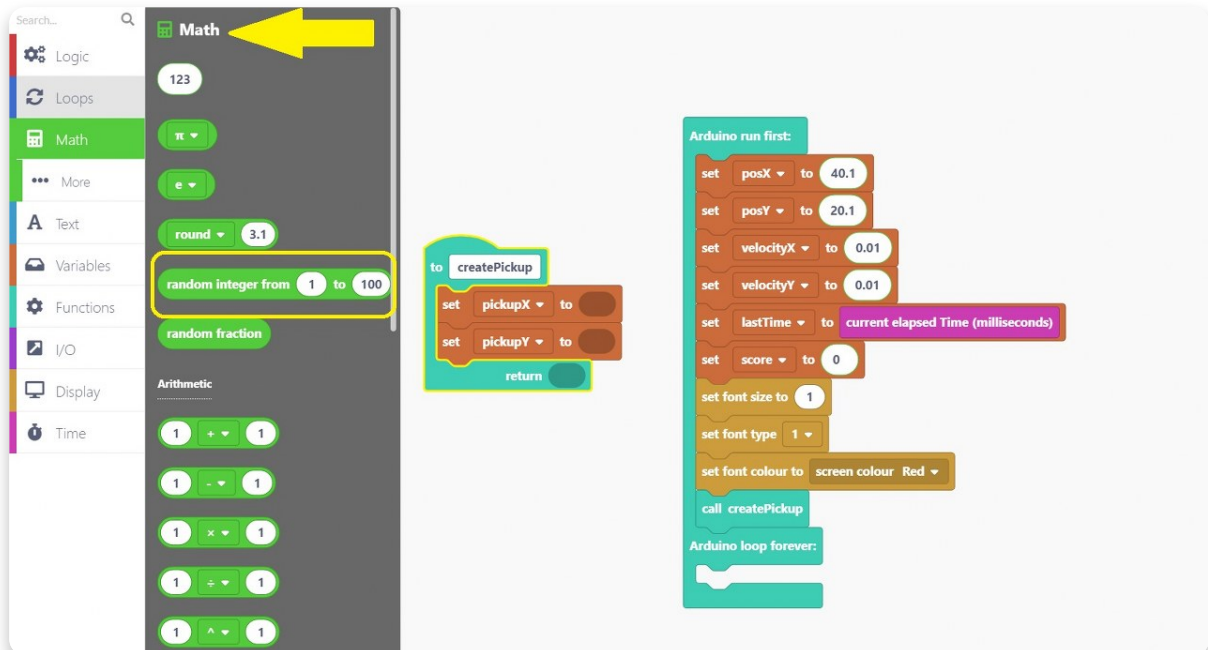
You probably guessed the next step – creating variables for the newly made function.

Let's call them "**pickupX**" and "**pickupY**" as they'll define the position of the pickup (the smaller ball you'll have to catch).



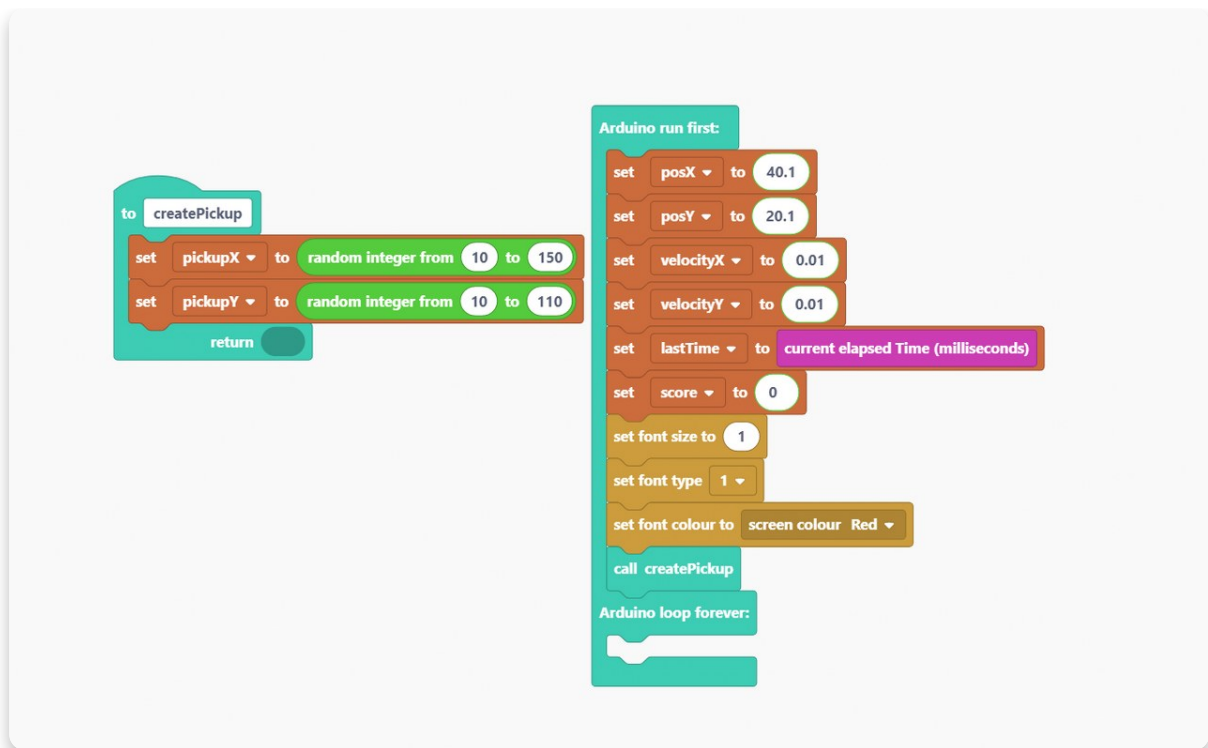
We'll set those variables to a **random integer**, and you'll find that block in the **Math** section.

An integer is a whole number that can be positive, negative, or zero. Examples of the integer are: '5, 1, 5, 8, 97.



Place those blocks in the pickup variables, and write the numerical value.

For "**pickups**" and "**pickupY**" you have to use a random number from 0 to 150 because the display resolution is 160*120 pixels.



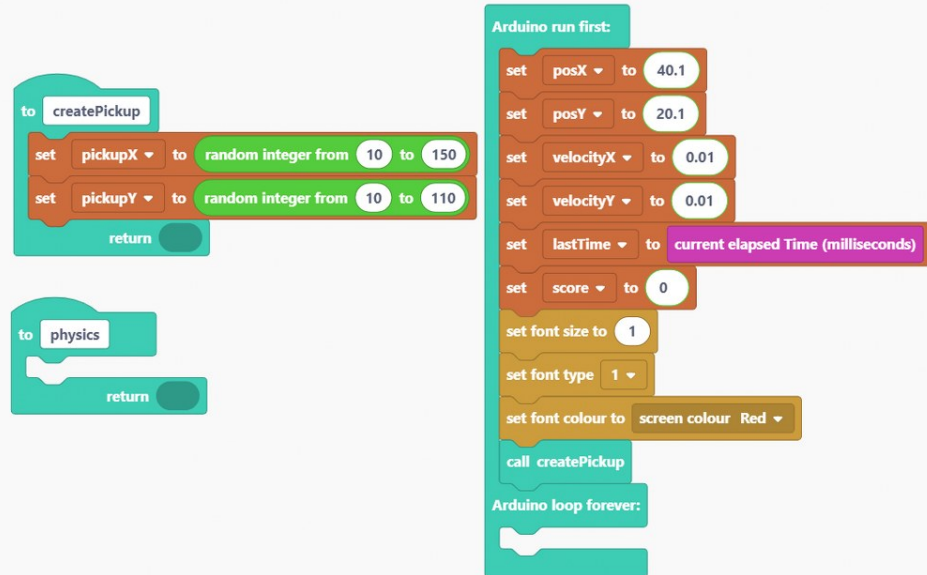
We're done with the "**Arduino run first:**" branch! That means that we created everything we wanted to see once the device turns on.

In the "**Arduino run first:**", we set the appearance of the text as well as default values for the character variables.

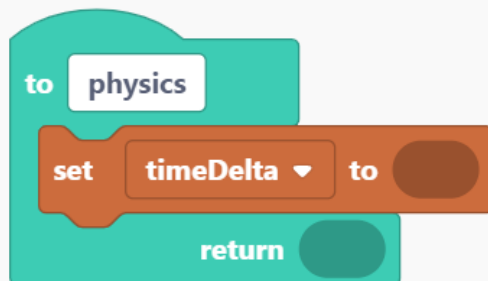
Let's put some functions in the "**Arduino loop forever:**" branch of the code.

The "**Arduino loop forever:**" branch is repeated all the time. We'll use this branch to update the character's physics and draw the character.

Firstly, we'll create a new function and call it "**physics**".

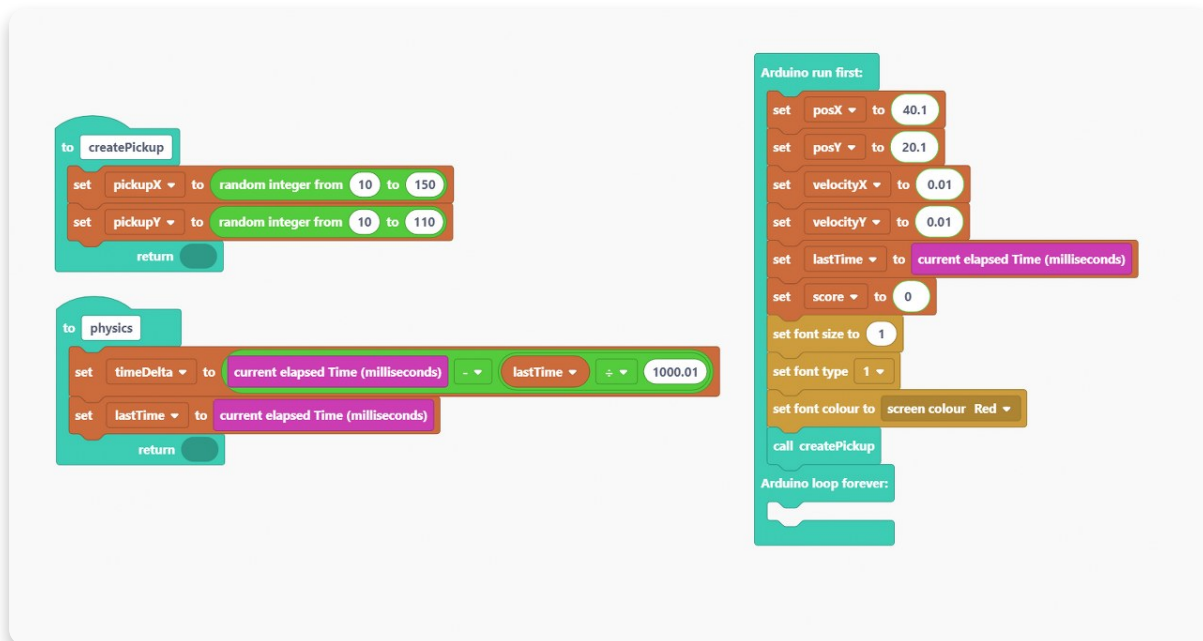


The first thing we have to do in the physics function is create a new variable called **"timeDelta"**.



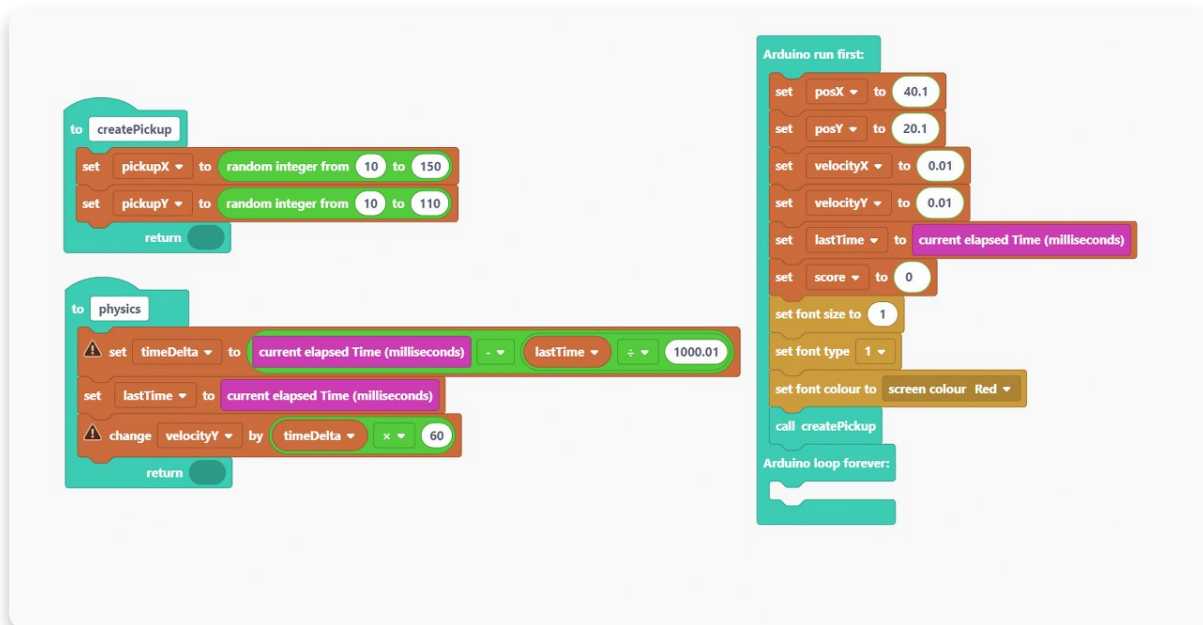
A Greek symbol delta is used to note the difference between the two things. We created the variable **"timeDelta"** to note the elapsed and measured time difference.

We will calculate "**timeDelta**" by subtracting from the current time the last time the function was executed. Also, we'll have to divide it by 1000.01 to get the time in seconds.



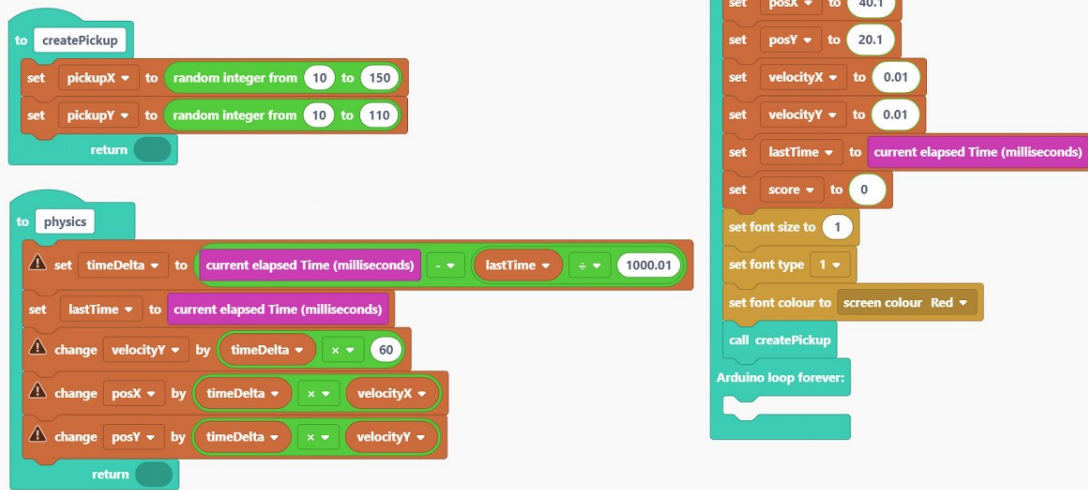
Now, let's play with the physics formulas a little bit.

"**VelocityY**" will increase by multiplication of "**timeDelta**" and 60.



We have to update position X and position Y based on velocity.

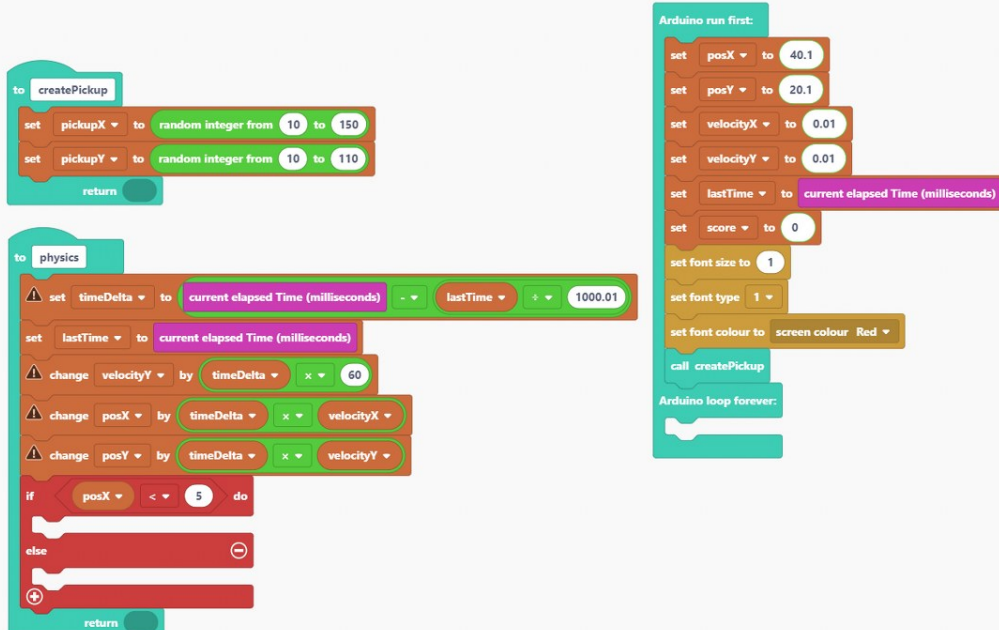
You can do that by using "**change posX/posY**" blocks and placing them under the "**change velocityY**" block.



Now, let's see what will happen **if the ball hits one of the edges** of the display.

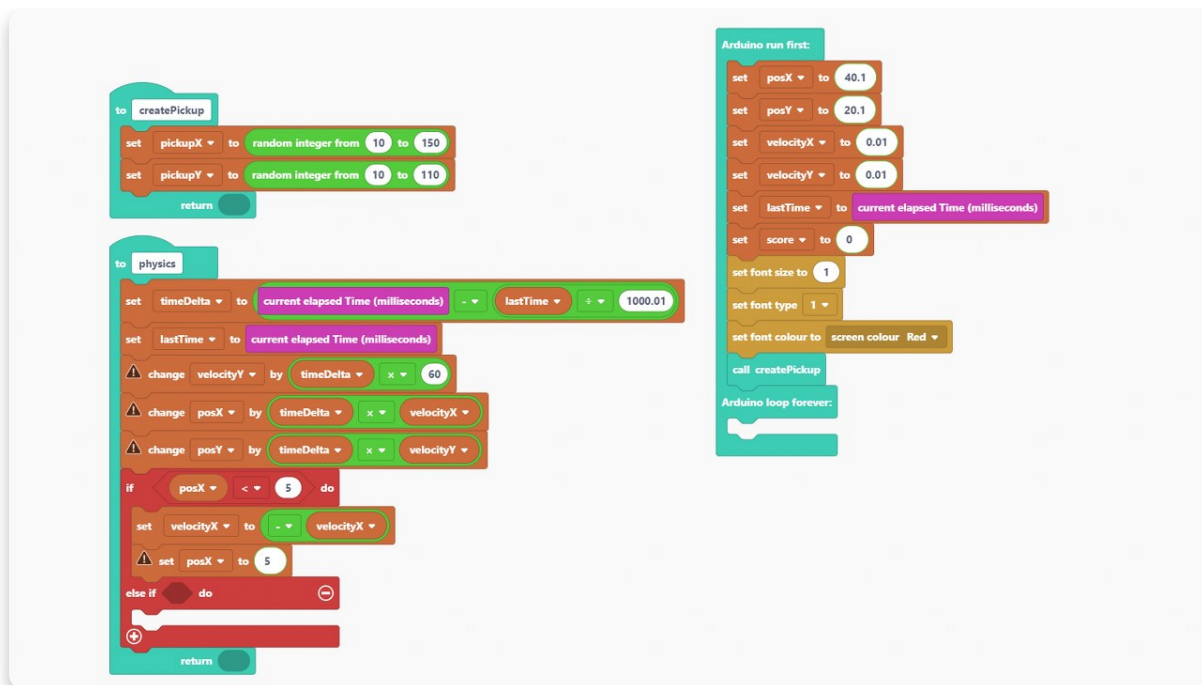
We'll use the **"if-do-else"** block from the **Logic** section and place the comparison block in it.

Your code will check if the ball's position on the x-axis is under five (**5**). As we mentioned before, the x-axis goes from 0 to 150.

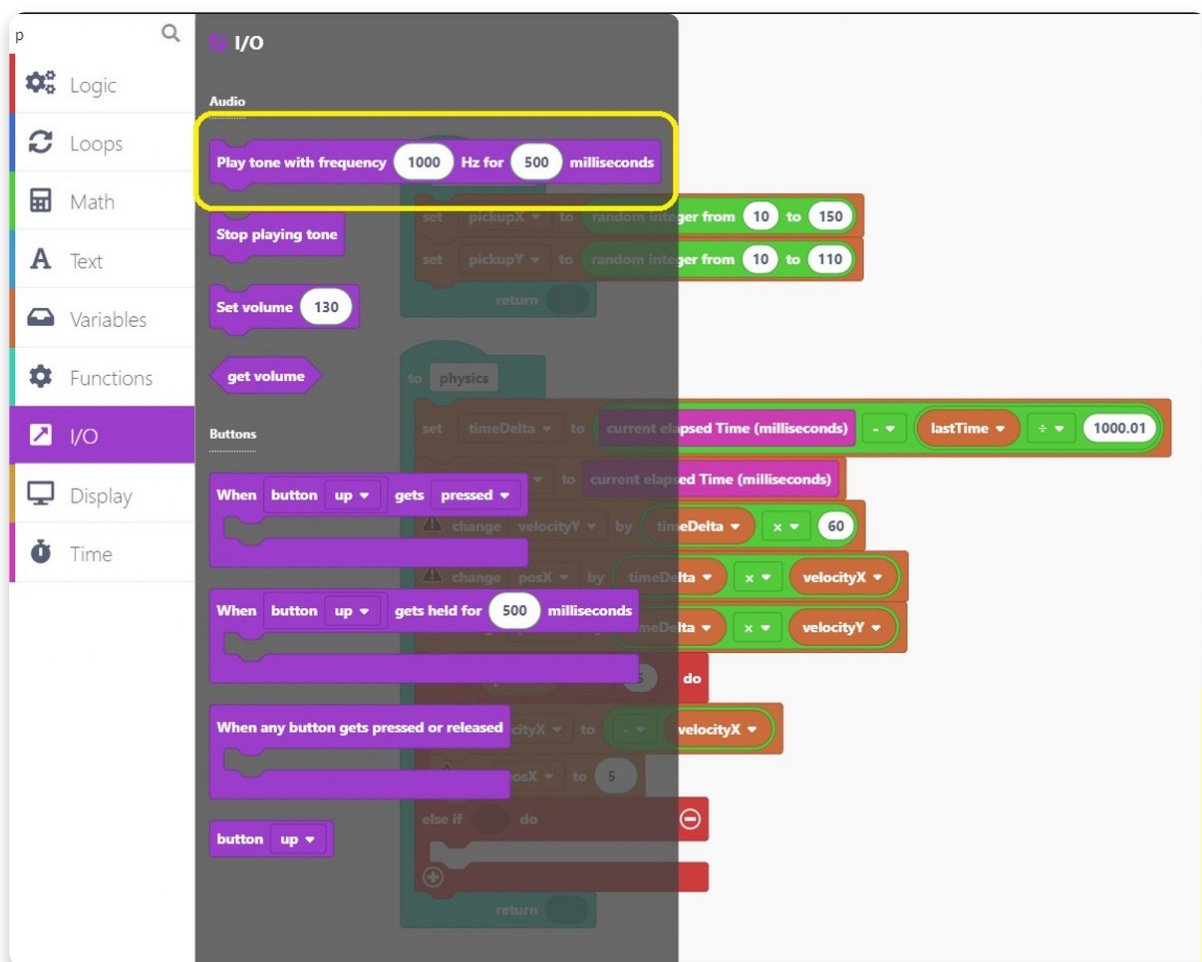


If position X is under the numerical value we gave it, which is five, **velocityX** will change into **-velocityX (negative on the axis)**.

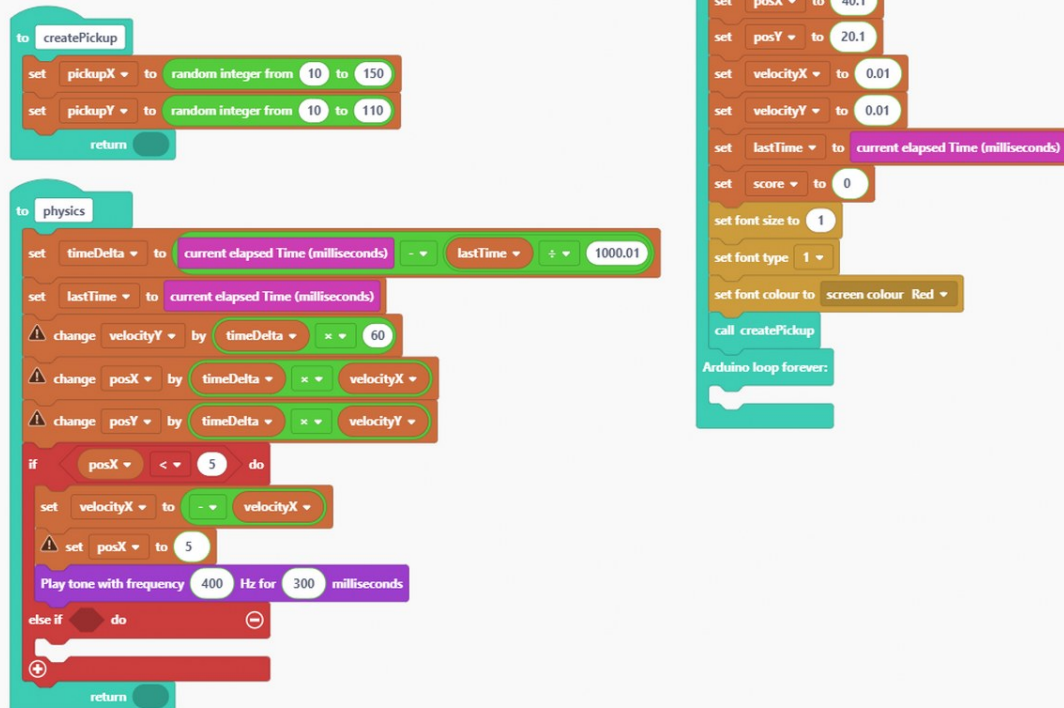
In other words, the ball will change its direction when it hits the edge.



If we want **the ball to make a sound** when hitting the edge, we have to go to the **I/O section** and pick this block:



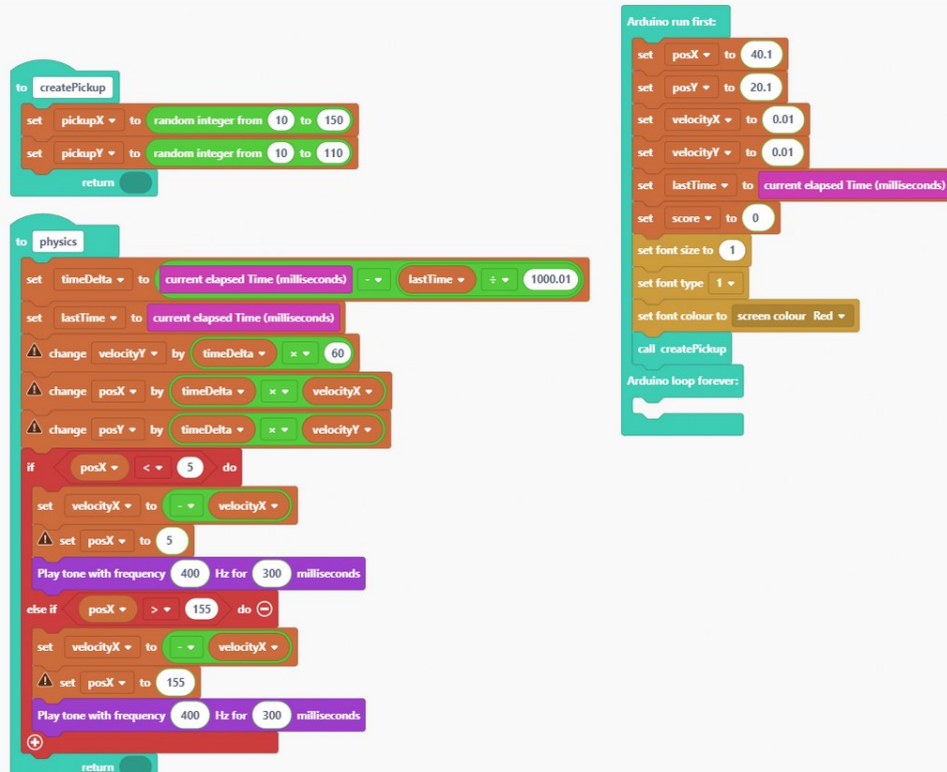
We decided to play a tone with a **frequency** of **400 Hz** for **300 milliseconds**.



Now, let's see what happens if the ball is in position over 155.

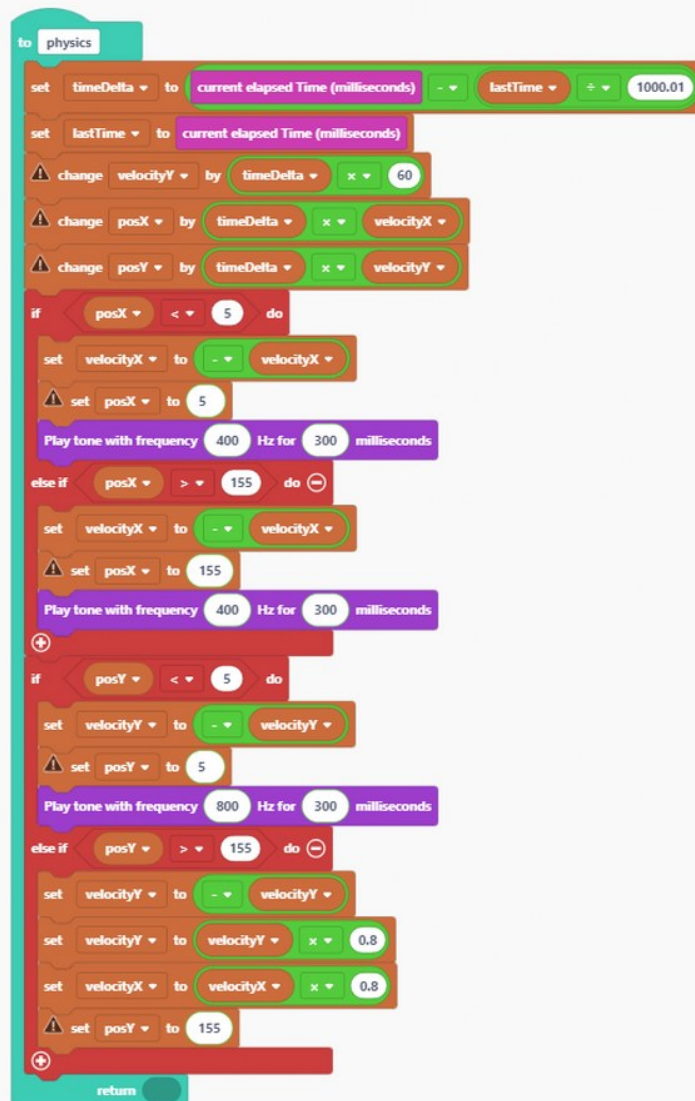
Once again, we have decided that the **posX** is set to 155. So, if the **velocityX** is bigger than 155, **the ball will change its direction**.

Don't forget to put the block for the sound!



Now, let's **repeat** the procedure for the **Y position**.

The **physics** function should look like this:



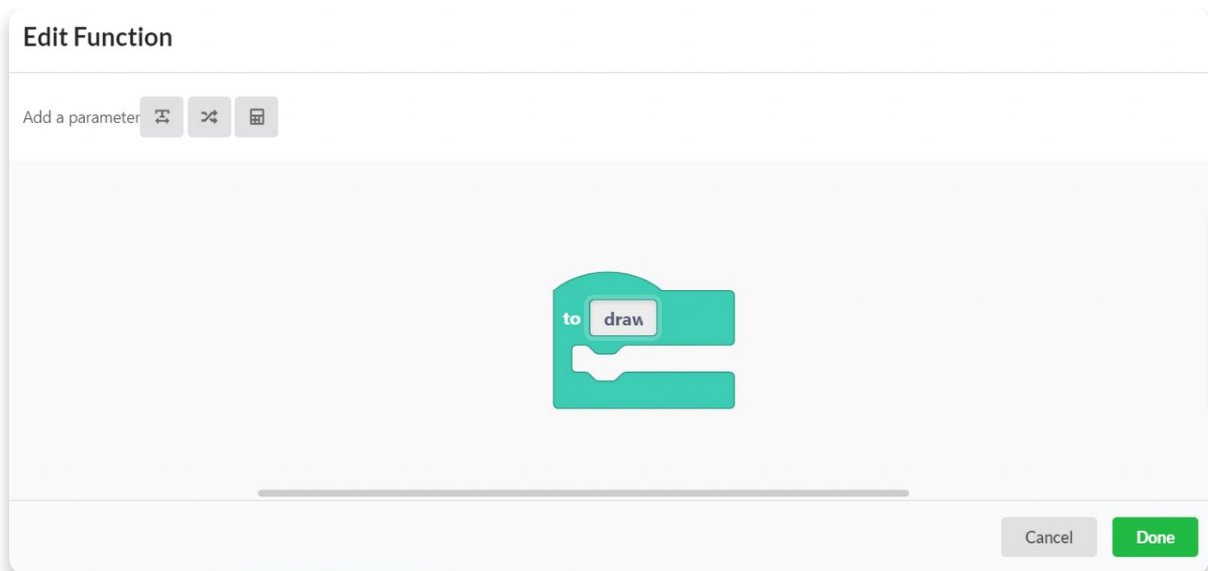
As you can see, we did the same thing as we did with position X.

We've done the hard part! Congratulations!

Now, let's do some drawing.

Can you guess what the next step is?

That's right! Creating a new function – let's call it **draw**!



Every time you draw something on your display, the first thing you want to do is **clear the display with some color**. We chose black.

You can find that particular block in the Display section.

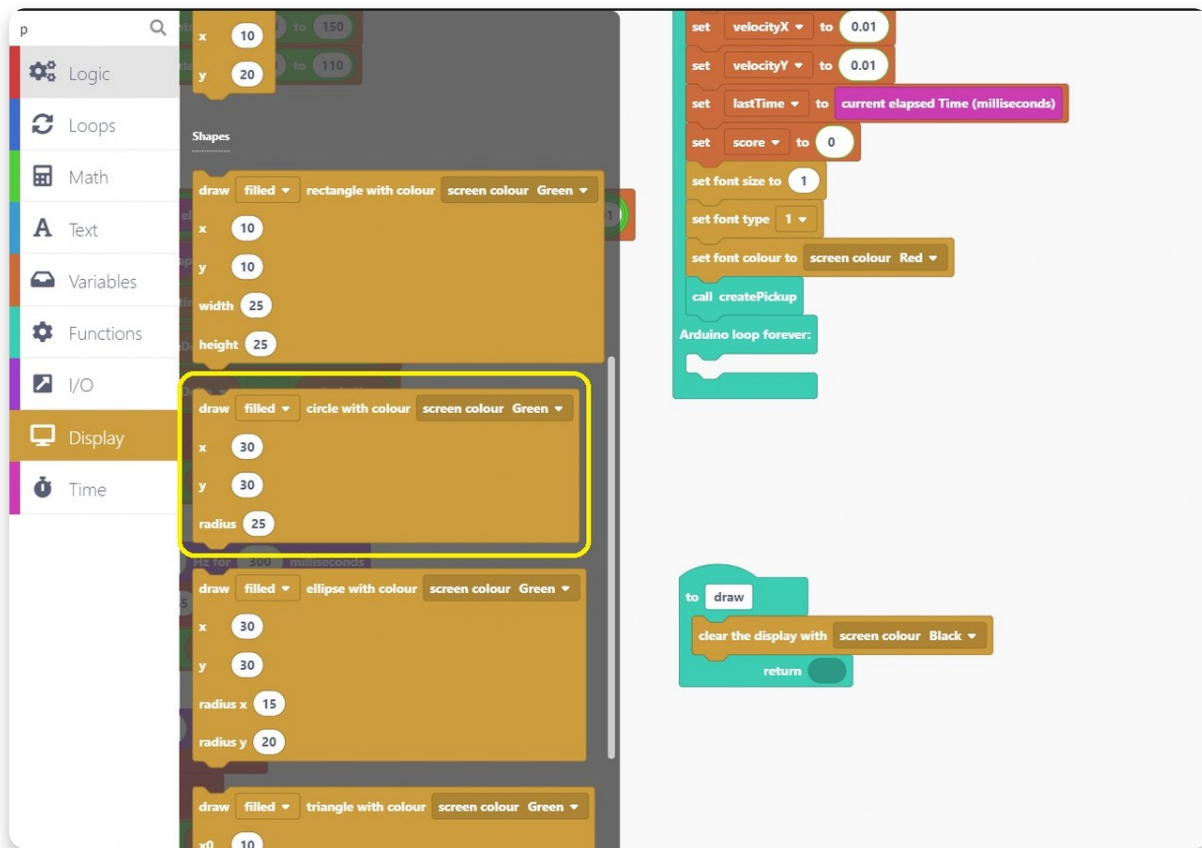


Once we color our screen black, we can start to draw.

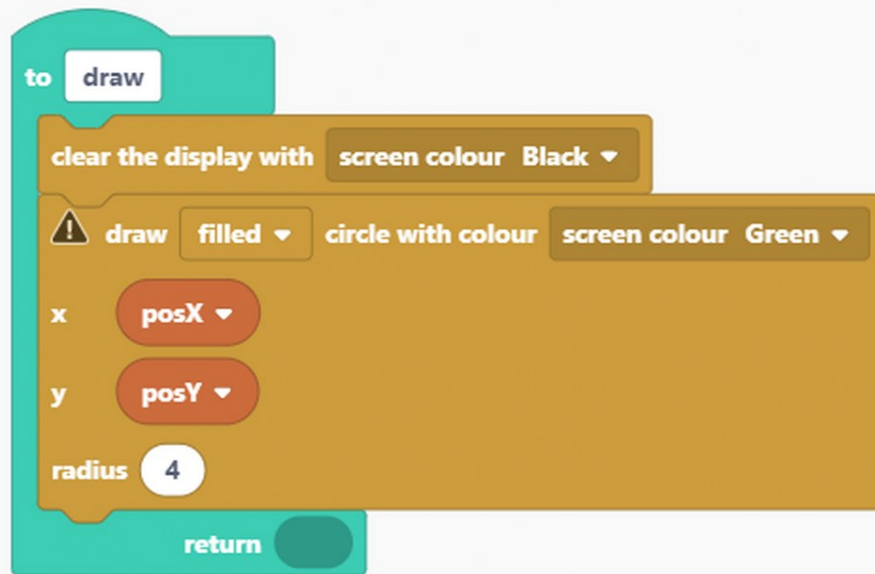
As you already know, the game consists of a player represented by a ball and the other smaller balls that a player needs to catch.

Once again, let's go back to the Display section and choose one of the blocks.

This is the one you'll need:



Place it in the draw function just like this:



The x coordinate will be the variable "**posX**", and y coordinate will be the variable "**posY**".

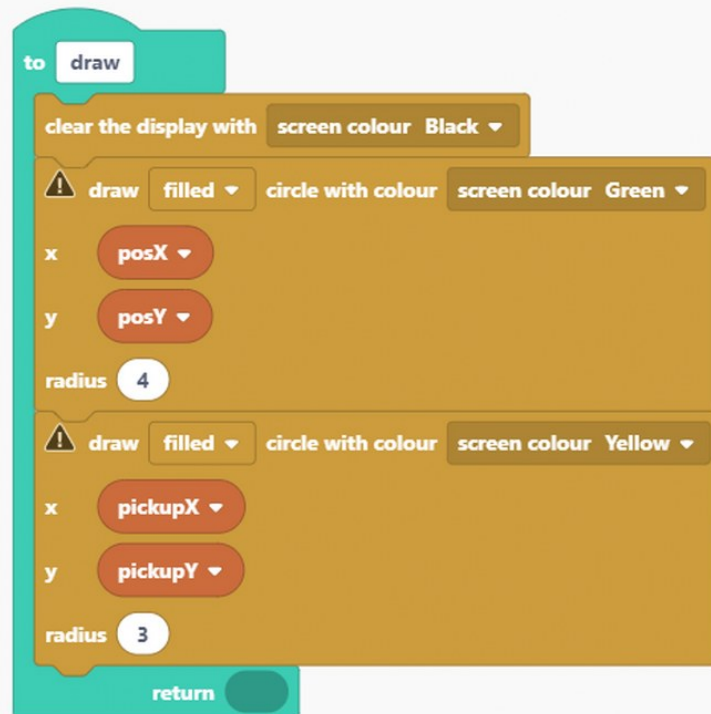
Radius is the **size** of the **player ball**, and we decided the size to be **4**.

Now, **let's draw the smaller balls** you'll need to catch. The procedure is the same as we used to make a player ball.

Since we have to recognize two balls, we need to change the color of the other ball. You can put any color you want in your game.

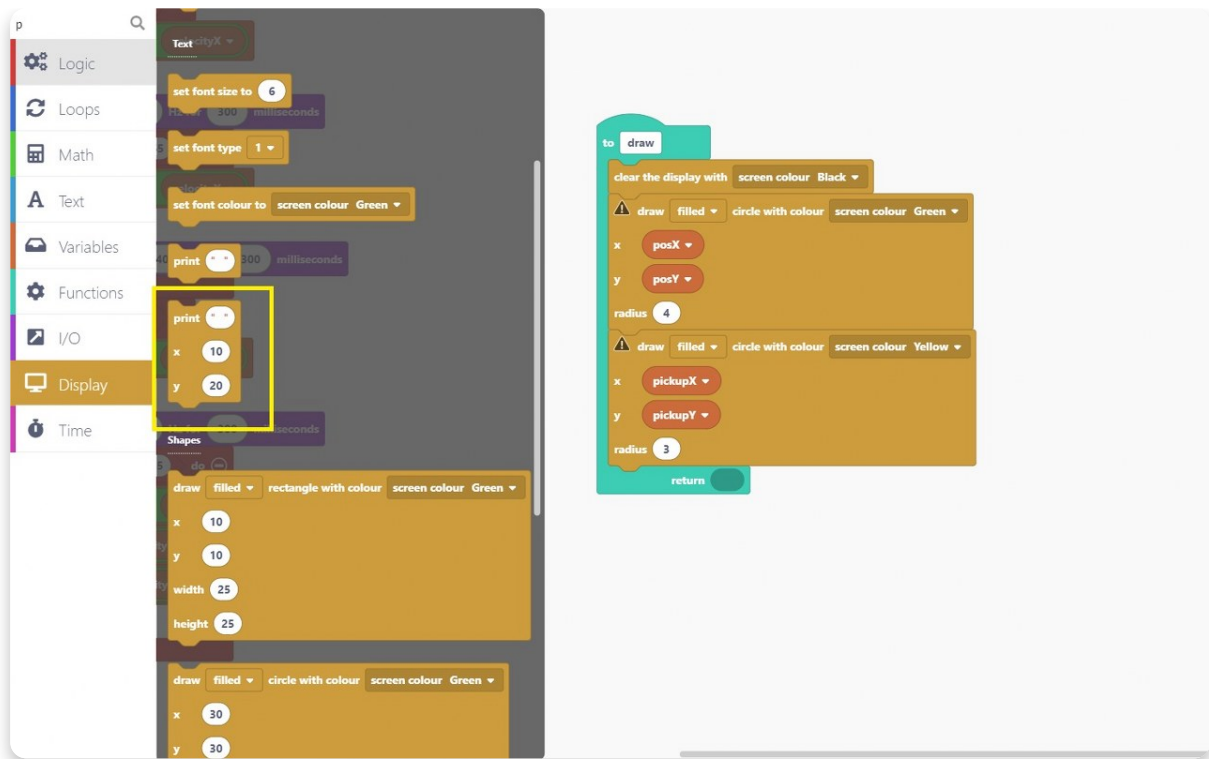
Also, the x coordinate of the drawing function is variable "**pickupX**" and the y coordinate is the variable "**pickupY**".

As this is a **smaller ball**, its **radius** will be **3**.



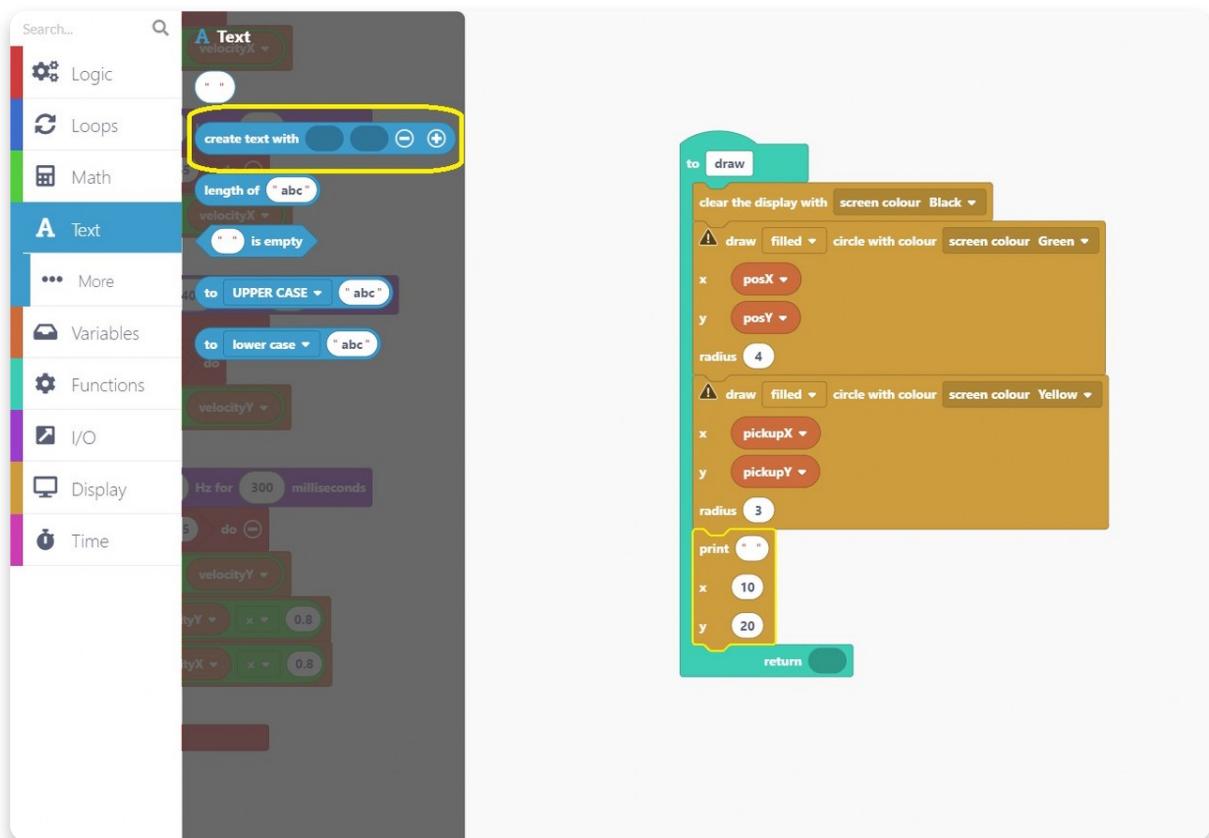
Only one thing is left for us to draw, and that's the score. Making a game without keeping the score wouldn't make any sense.

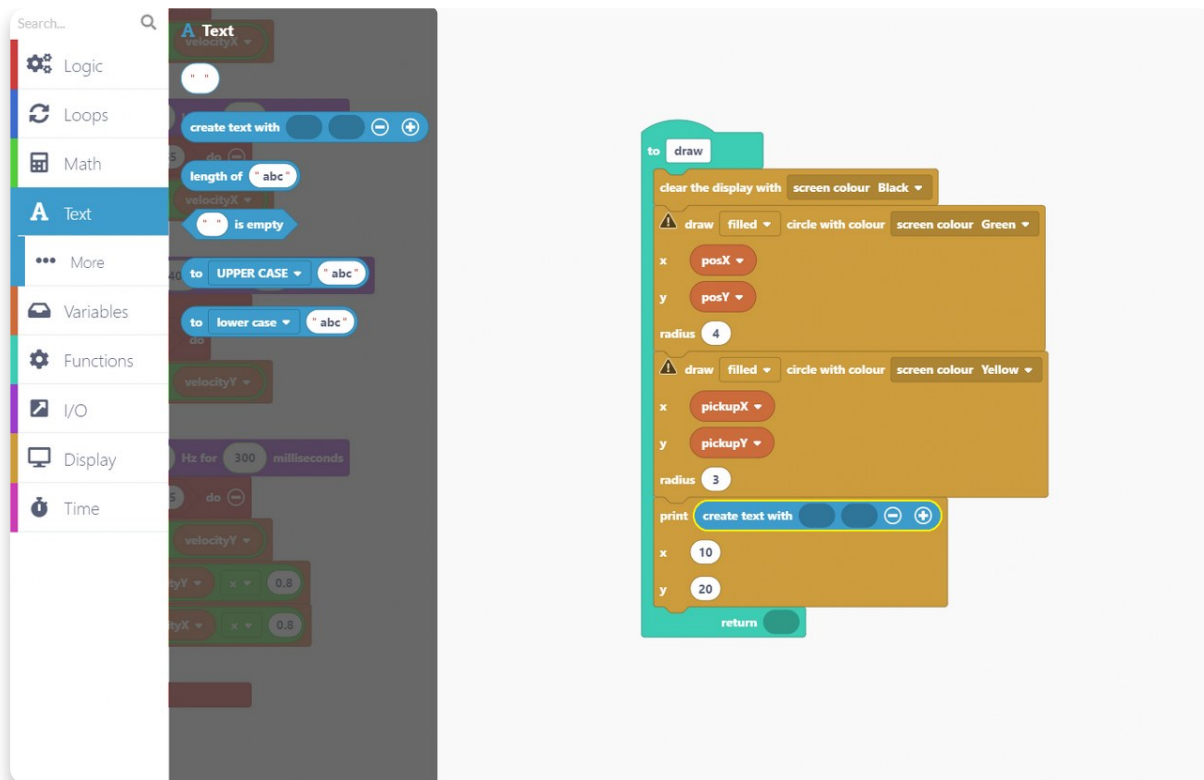
You'll do that by dragging and dropping this particular block from the **Display** section:



In order to display the score on the screen, we need to find the **"create text"** blue block.

You can find this block in the **Text** section.

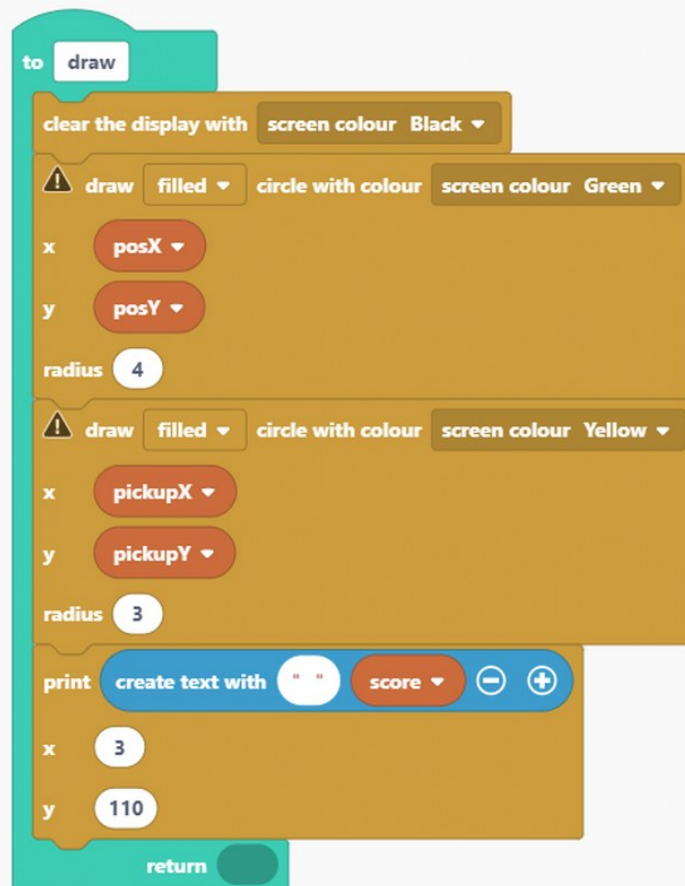




We'll put the word "**Score**:" in the left circle, and for that, we need the first block from the **Text** section.

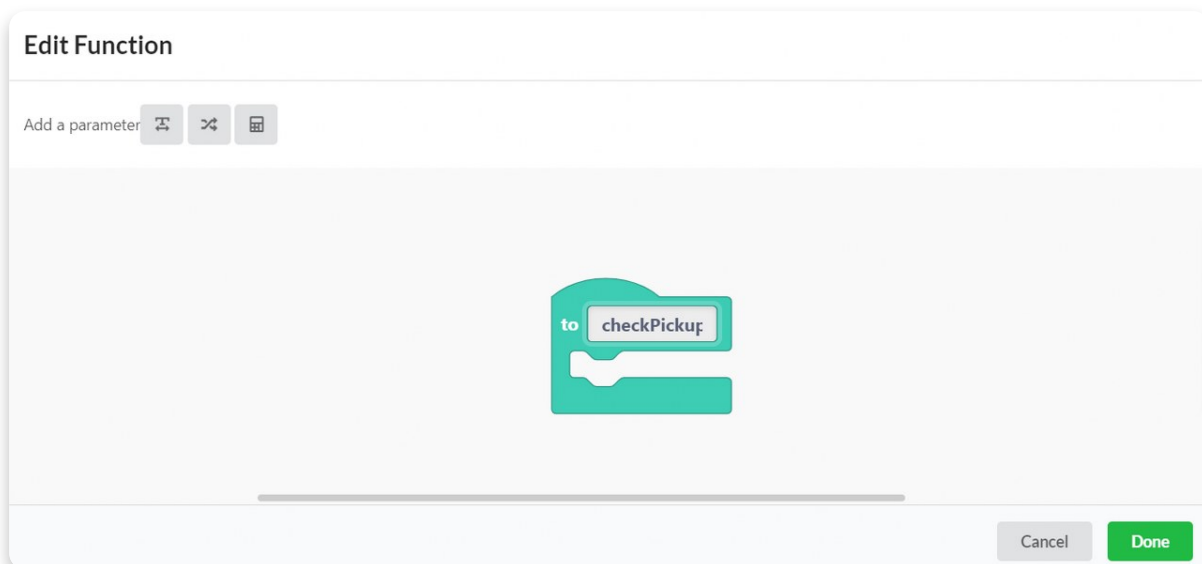
On the right side of the block, we'll put a variable called "**score**".

In x and y, you'll write the desired position where the score will be written on the screen.



Good job, we're done with drawing!

Create a new function, and call it "**checkPickup**". This function will detect whether the player has collected a ball.



The first thing we'll have to do with this function is to calculate the distance between the player and the pickup ball. We used the formula you see below.

That's the formula you can use for calculating the distance between two points in a 2D space.



We need to use the "if" block from the **Logic** section to check if the distance we mentioned is less than 8 pixels.

We'll use a **comparison block** to check that.



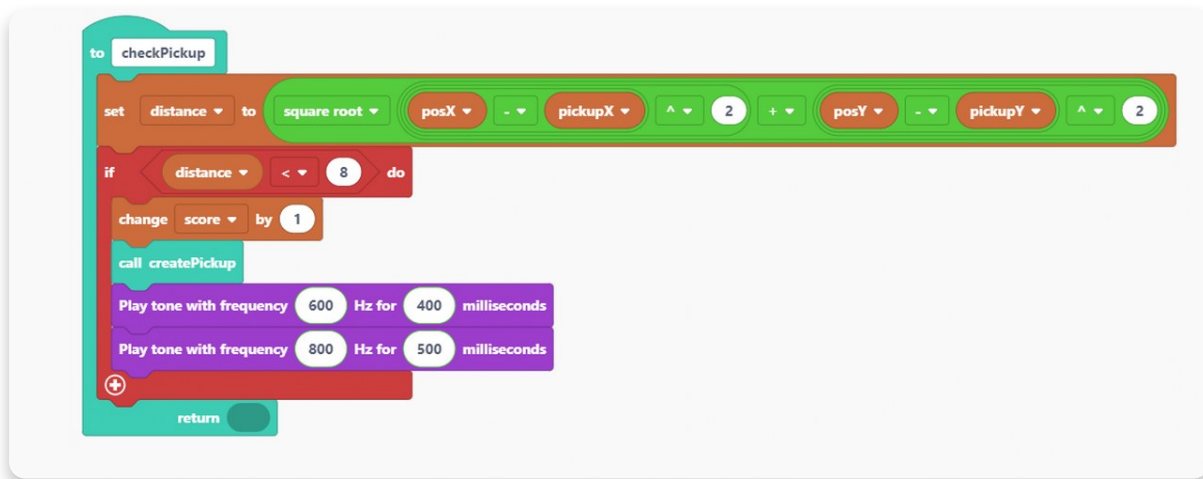
If the distance is less than 8 pixels, that means **that the player caught the ball**, and your scoreboard increases by one point.

To keep the score, you'll have to take this block:



So anytime you pick a ball, you'll get one point!

If you want to put your function in action, you need to call it. So open the **Functions** section, and grab the "**call createPickup**" block.



Also, we put some cool sounds to play once you collect the pickup ball.

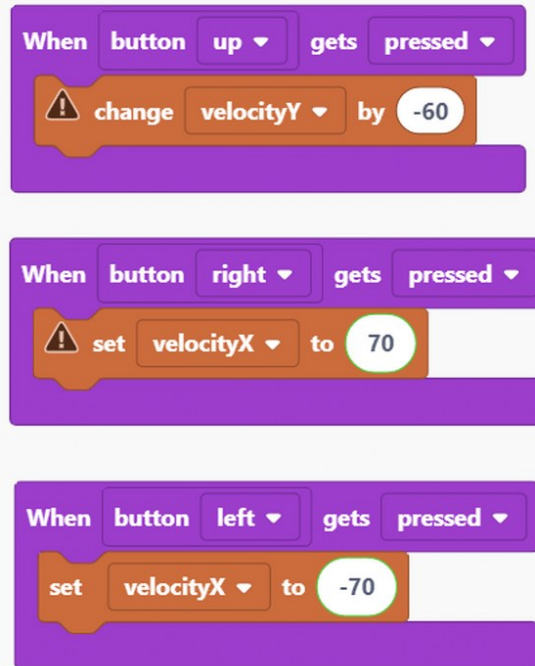
The very last thing we'll do for our game is something we have already learned.

We'll use ByteBoi's pushbuttons to **increase the velocity of the ball**.

Open the **I/O section**, and click on the "**When button up gets pressed**" block.

If the right button gets pressed, velocityX will set to 70 in the positive direction, and if the left button gets pressed, **velocityX** will be set to 70 in the negative direction on the x-axis.

If you press the button up, **velocityY** will change by 60.



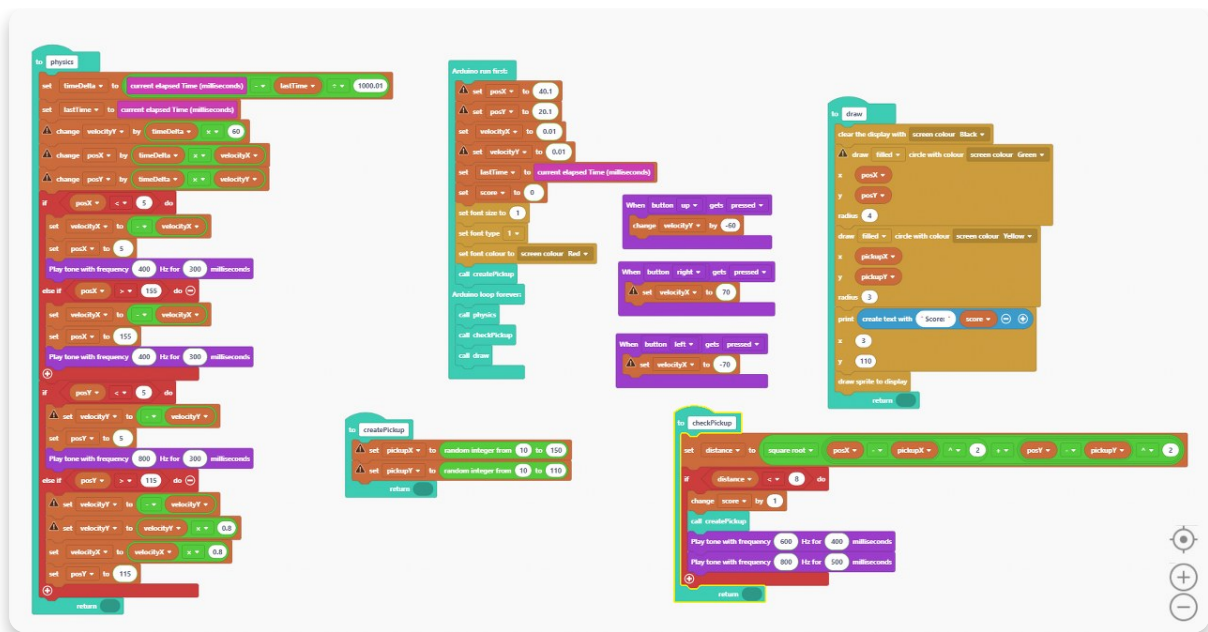
Complete game

This is it!

We tried to explain every part of the game, and now here it is all in one place.

We hope this tutorial has managed to help you make your first steps in video games creation.

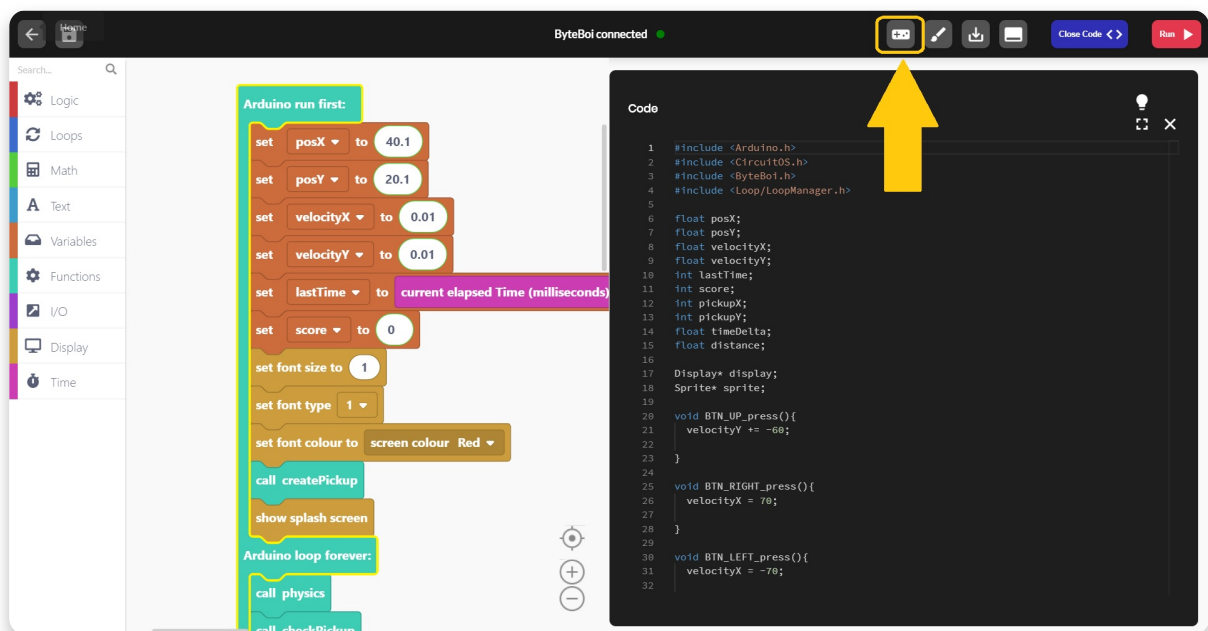
Don't forget to save and name your sketch. Then, click the **"Run"** button and grab the ByteBoi!



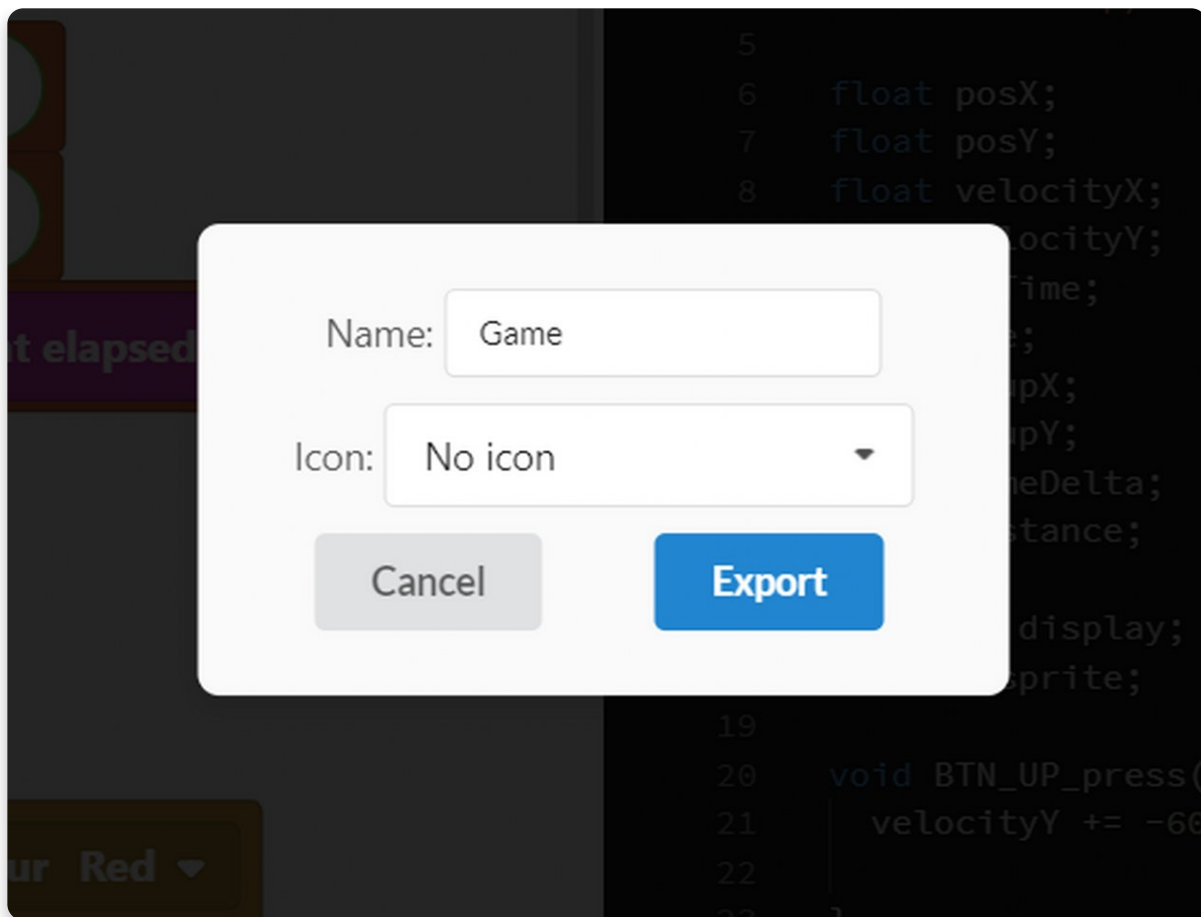
Export your game...

After you have successfully created your first game, you'll probably want to export it to the SD card.

The first thing you'll have to do is click on the Export game button in Toolbar.



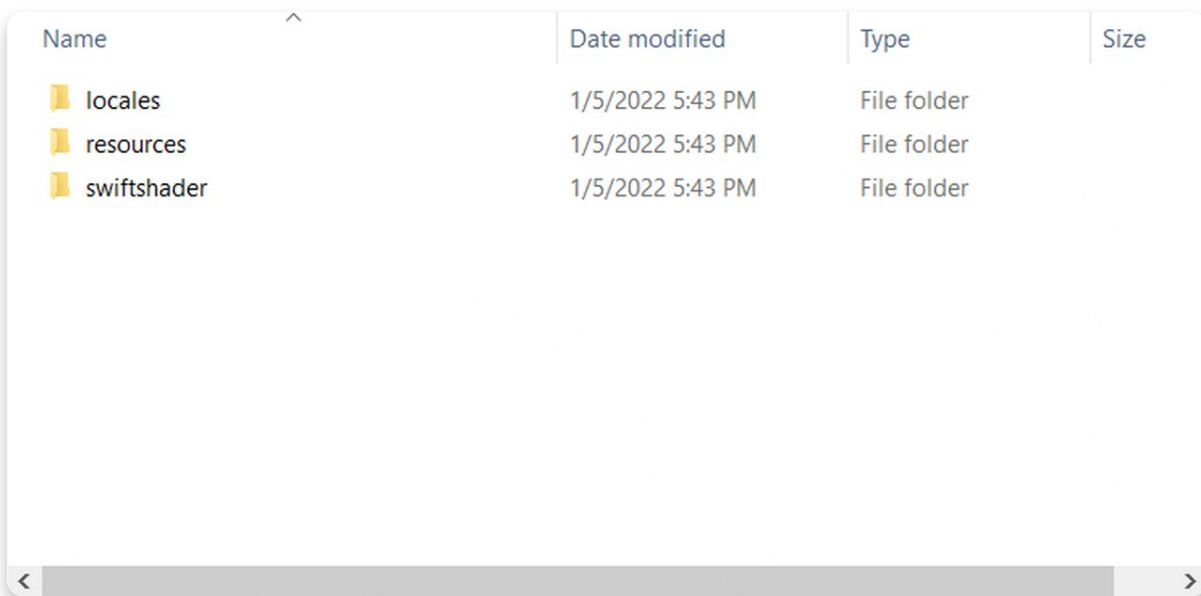
After clicking on it, the window to choose a name and an icon for your game will pop up.



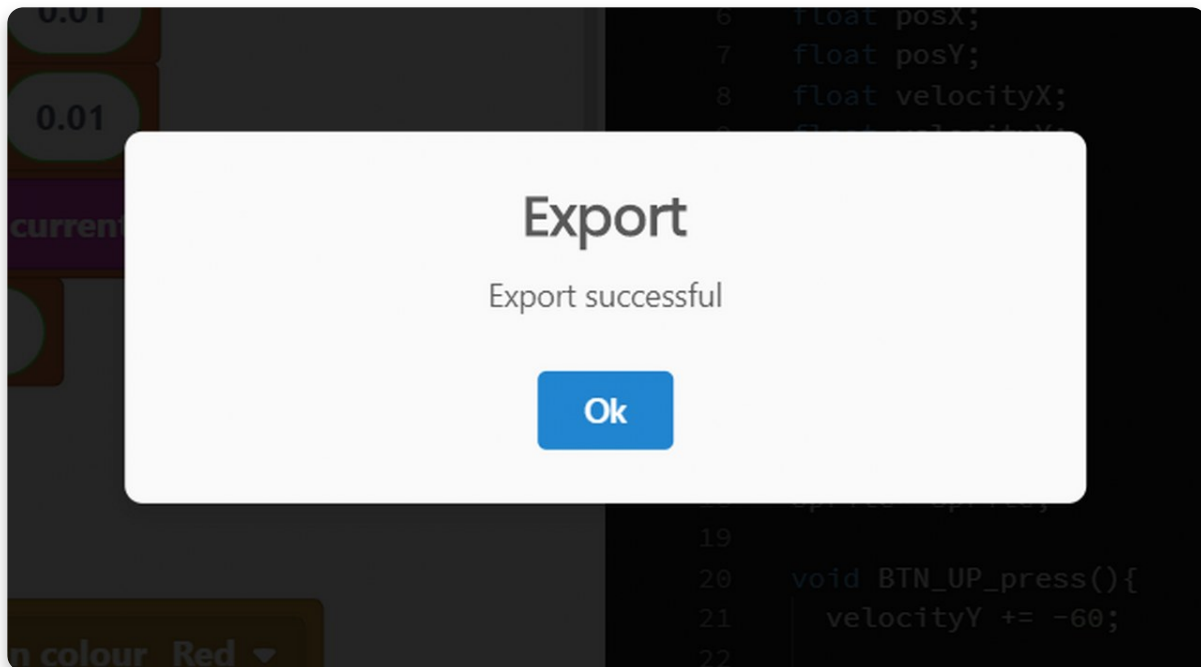
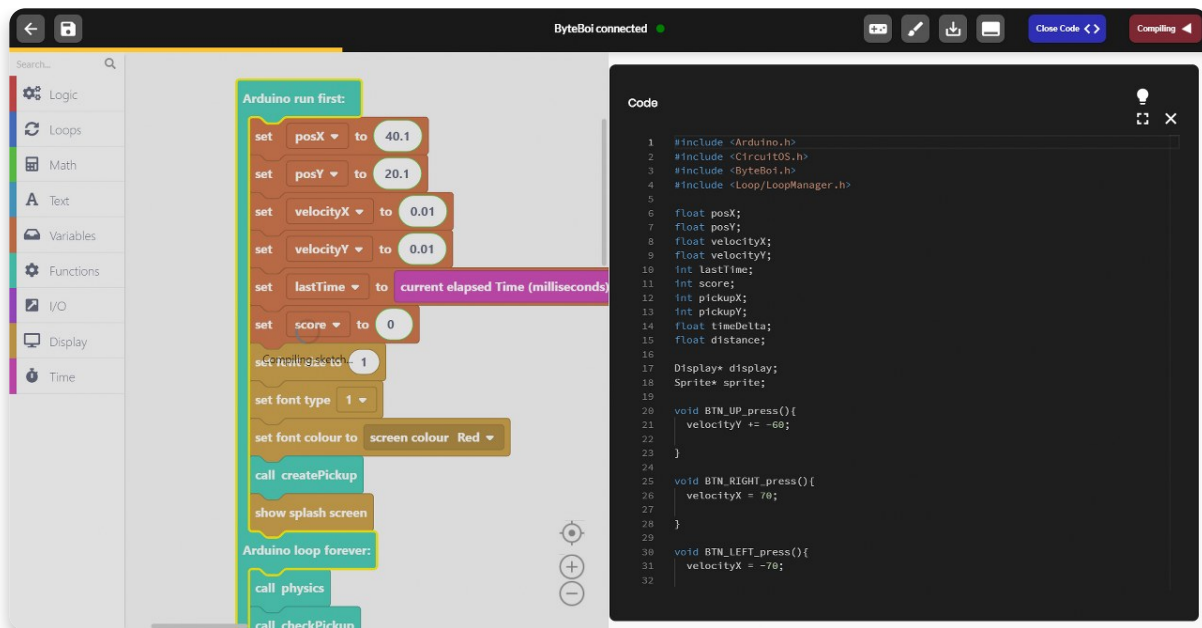
You can choose whichever name you want for your game and every pre-made icon or one you made.

After doing that, simply press the big blue Export button.

Now, you have to save the game either on your PC or on ByteBoi's SD card.



The sketch will compile in a minute, and at the end, you'll get a pop-up message saying that the export was successful.



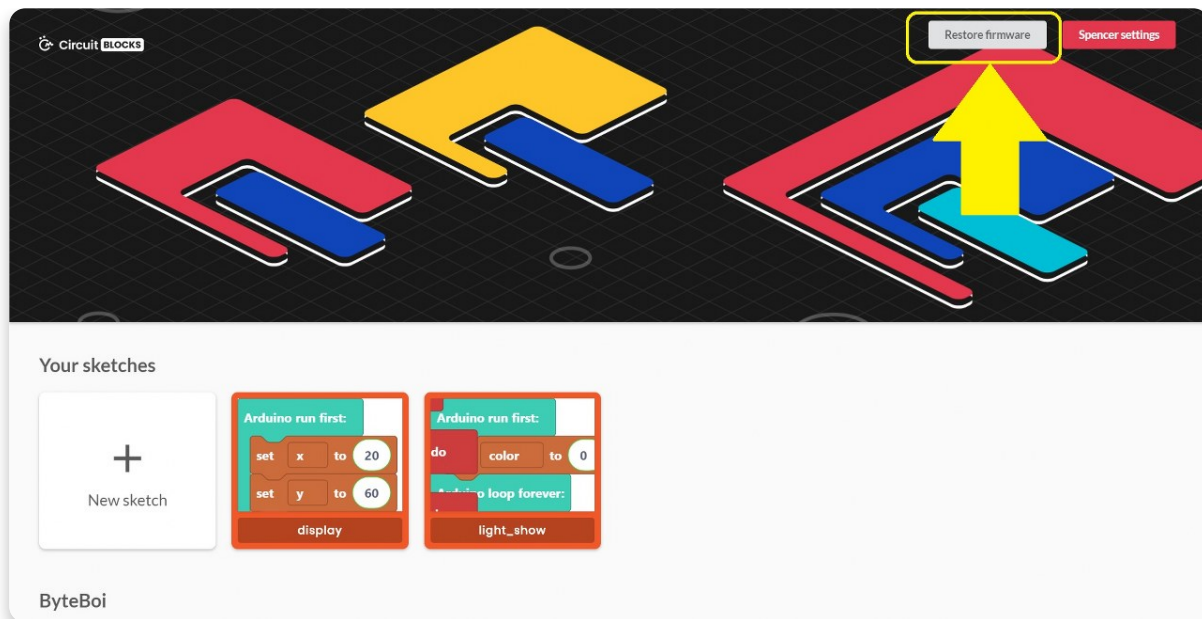
Great job!

Restore ByteBoi's base firmware

Restore ByteBoi's firmware

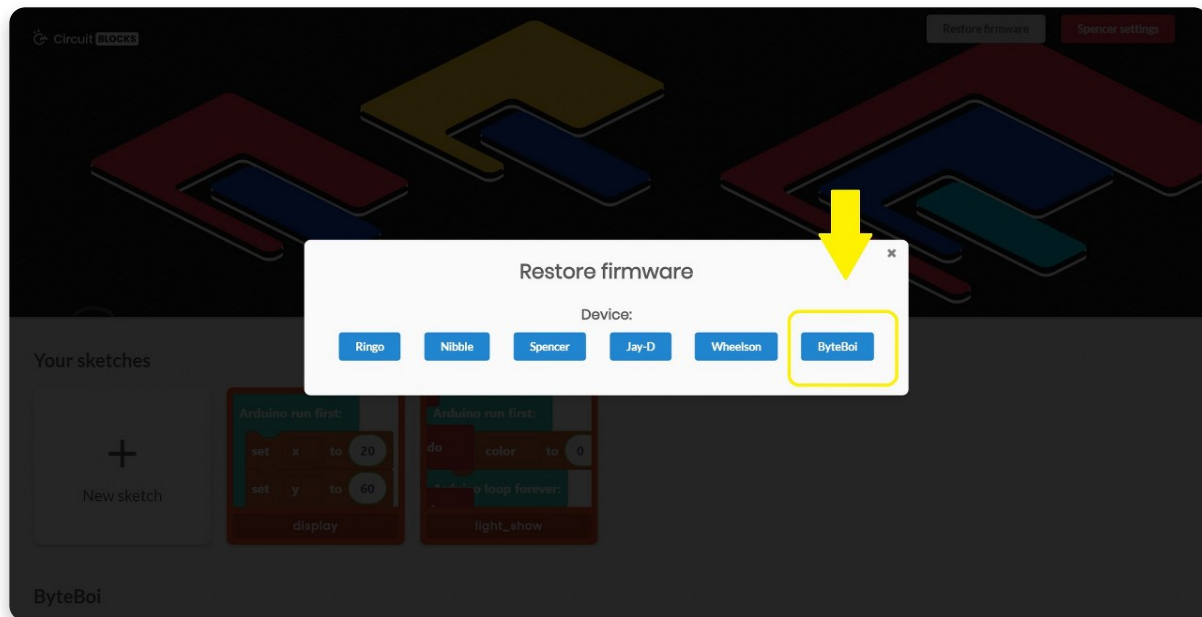
Once you're done coding and just want your ByteBoi to be "normal" again, you need to restore his base firmware.

This is quite simple, just connect your ByteBoi 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 ByteBoi, of course.



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

You need to do this whenever you're done coding your ByteBoi if you want him to revert to his initial out-of-the-box functionality.

What's next?

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

I hope you're as excited as we are about ByteBoi'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 ByteBoi 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!