# Ringo coding - first steps

## Introduction

## #1 - Installation

Welcome to the **CircutBlocks** tutorial!

Here is where you learn basic **CircuitBlocks** functionalities as well as start working on your first programs.

If you don't know, **CircuitBlocks is a Scratch-based** (a visual block programming language) **IDE** in which you can easily and effectively create and upload your projects to the Ringo phone.

This tutorial is going to be broken down into several chapters, each representing one of the important aspects of the IDE.

**NOTE: CircuitBlocks will be referred to as CB in the future**

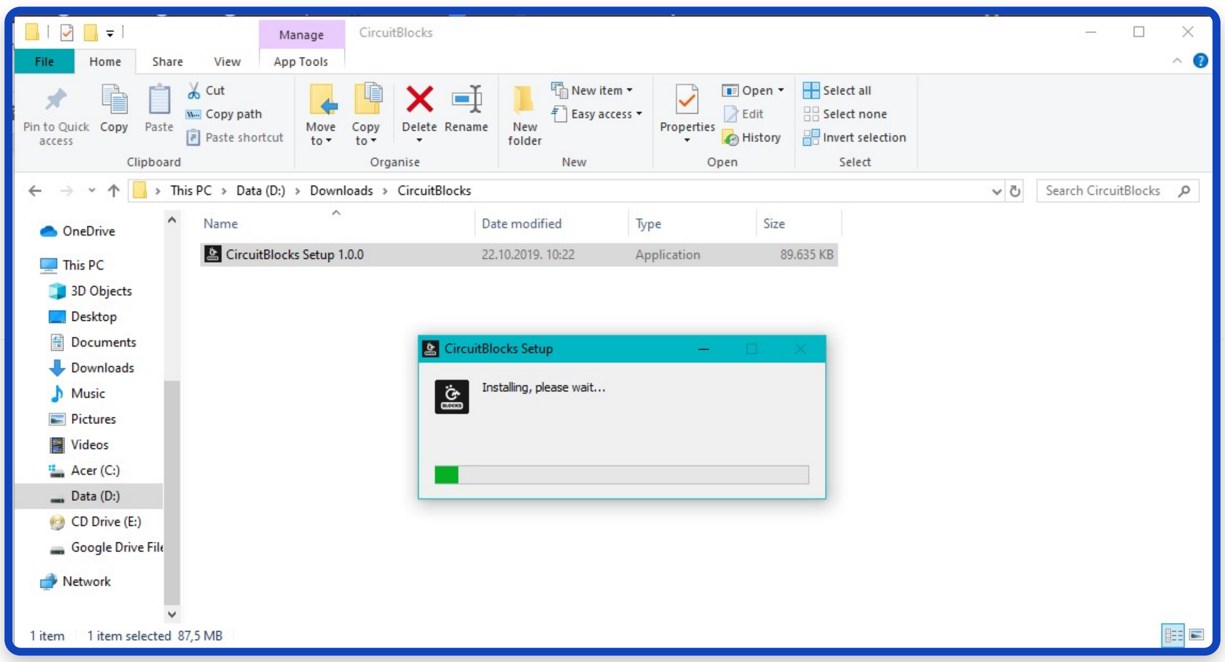
## Installation

CB is supported on all major platforms.

The installation process is really easy as you just need to download the file and install it just like you would with any other program on your preferred platform.

### Windows

- **Go to the [CircuitBlocks download page]()**
- **Download the latest version *Windows.exe (ex. CircuitBlocks-1.1.0-Windows.exe) -** 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.
- Double-click the file to run the executable
- CB will automatically install and create a new desktop shortcut



| Name | Date modified | Type | Size |
|---|---|---|---|
| CircuitBlocks Setup 1.0.0 | 22.10.2019. 10:22 | Application | 89.635 KB |

CircuitBlocks Setup

Installing, please wait...

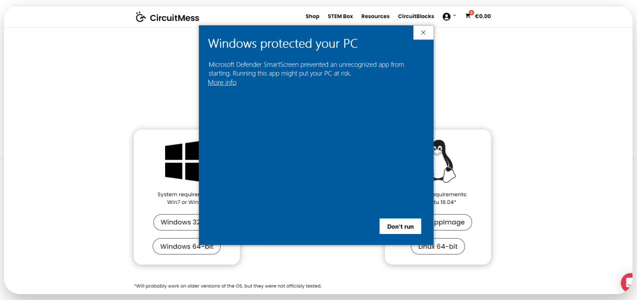
1 item    1 item selected 87,5 MB

**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 trying to install CircuitBlock on your PC. Windows reports a threat despite the program being safe to download and run.

Please proceed with installing by clicking on 'More info' option.



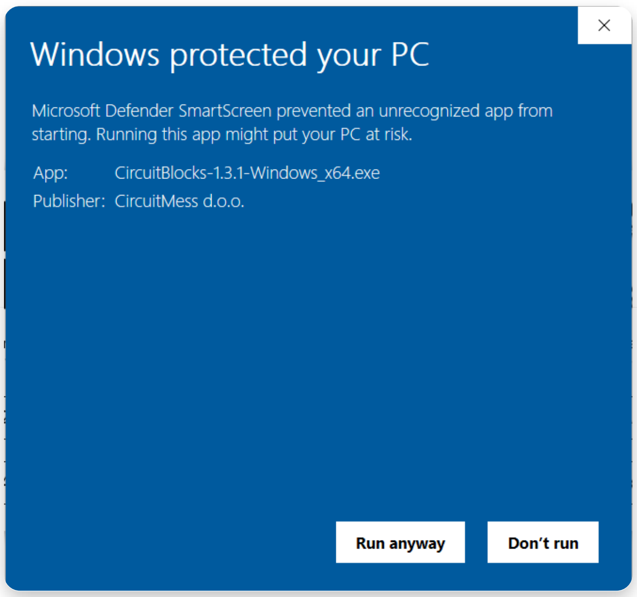
Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

More info

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

Proceed by clicking on 'Run anyway'.



Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

App: CircuitBlocks-1.3.1-Windows_x64.exe
Publisher: CircuitMess d.o.o.

## Linux

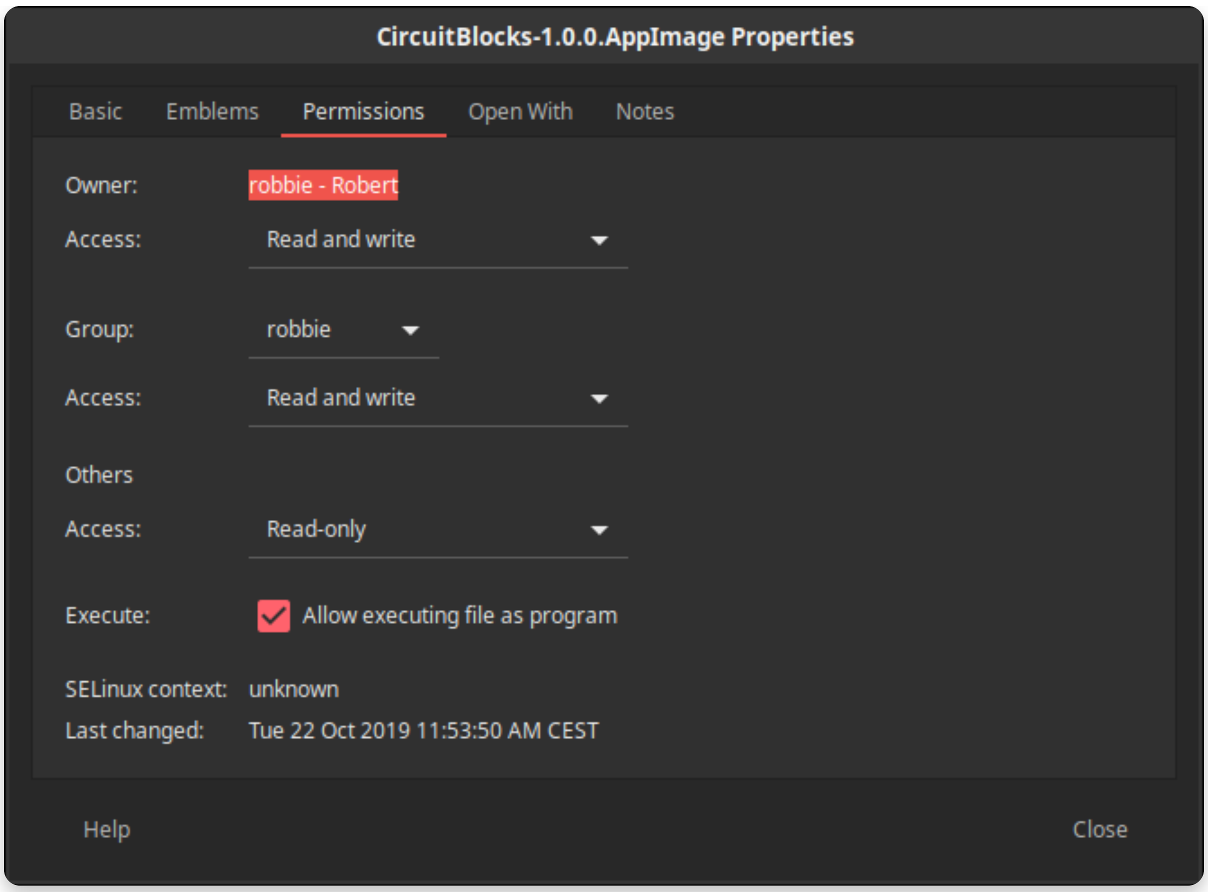
There are two ways of installing CB on Linux

**Installation:**

- **Go to the [CircuitBlocks download page]**
- **Download the latest version *Linux_amd64.deb (ex. CircuitBlocks-1.0.1-Linux_amd64.deb)**
- Double-click the file to run the installation (Ubuntu)
- Write inside a terminal `sudo dpkg -i  <path to the downloaded file .deb>` (Other Linux distros)
- CB will automatically install and create a desktop entry

**Stand-alone (AppImage):**

- **Go to the [CircuitBlocks download page]**
- **Download the latest version *Linux.AppImage (ex. CircuitBlocks-1.0.1-Linux.AppImage)**
- Right-click on the file and select **'Properties'**
- Go to Permissions and tick 'Allow executing file as program'
- Double-click the file and the installation will complete automatically



CircuitBlocks-1.0.0.AppImage Properties

Owner: robbie - Robert
Access: Read and write

Group: robbie
Access: Read and write

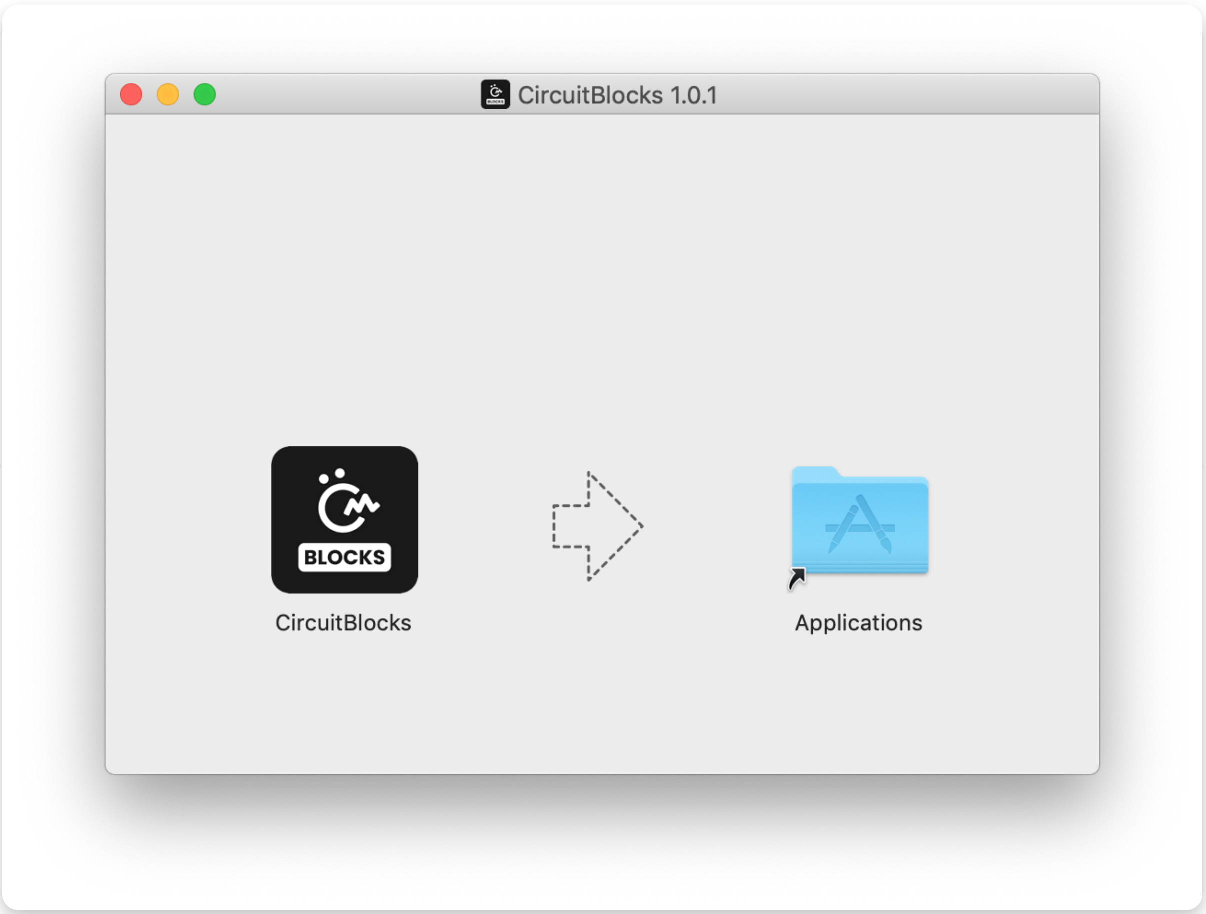
Others
Access: Read-only

Execute: ✓ Allow executing file as program

SELinux context: unknown
Last changed: Tue 22 Oct 2019 11:53:50 AM CEST

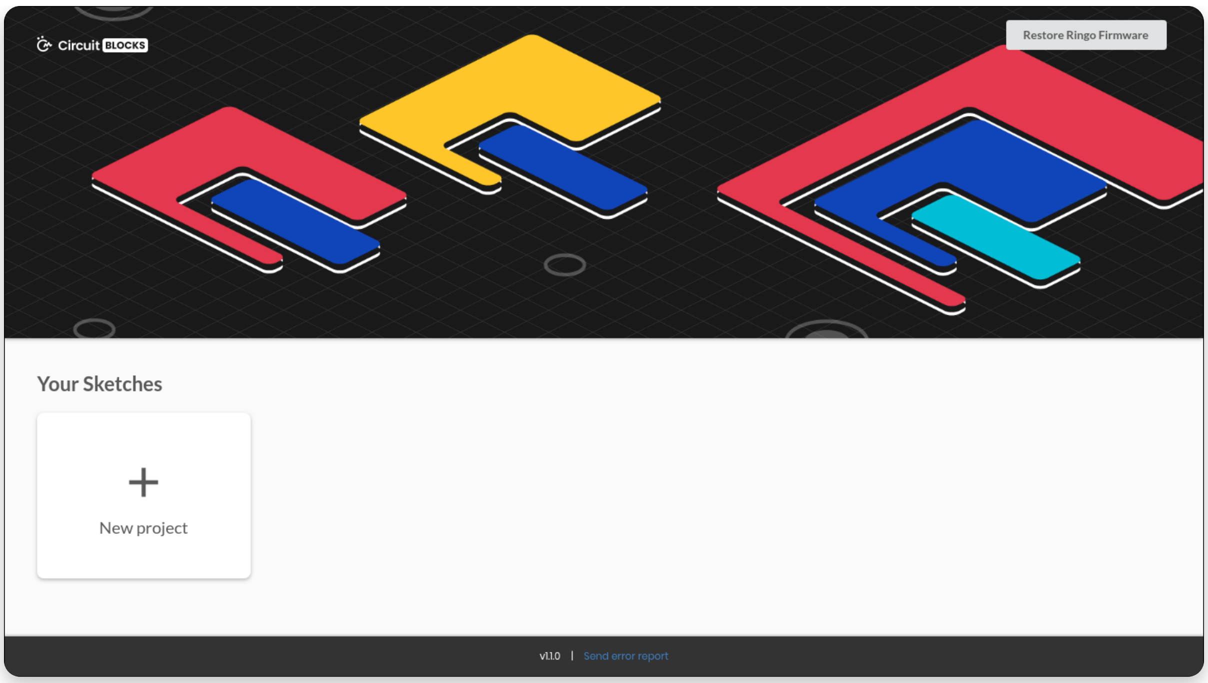
## Mac OS

- **Go to the [CircuitBlocks download page]**
- **Download the latest version *Mac.dmg (ex. CircuitBlocks-1.0.1-Mac.dmg)**
- Move the files to **'Applications'** folder
- CB will be installed automatically



# #2 - Basics

## Interface



Starting CB will open a window like this.

It's pretty simple – starting a **new project (sketch)** can be done by clicking the 'New project' button.

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

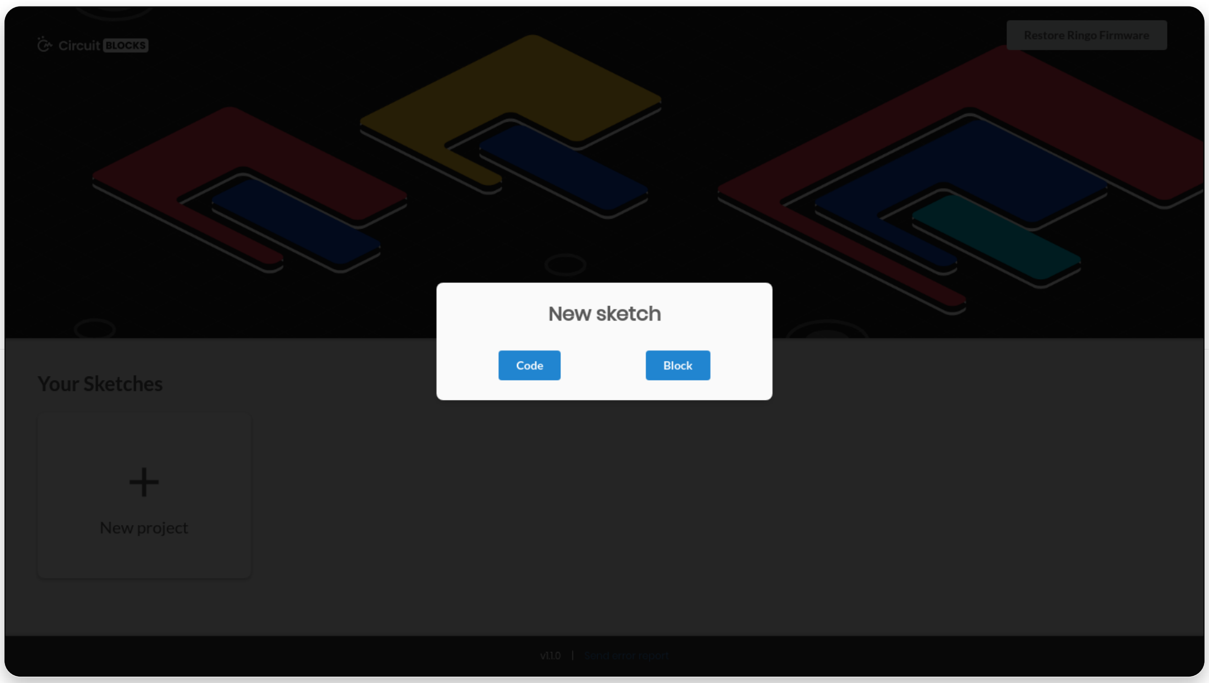
One very important button is **'Restore Ringo Firmware'**. If your Ringo is connected to the computer, it will automatically detect it and allow you to restore the newest software with just a push of a button.

**So any time you want to revert to the default software after working on a sketch, push that button**.

Whenever you encounter an error, press the **'Send error report'** at the bottom of the main screen. You'll get the report and the error number - you can use that in the CircuitMess community forum so that our team members can help you more efficiently.

## Starting a new sketch

When starting a new sketch you'll get an option to choose from a **code project** and a **block project**.

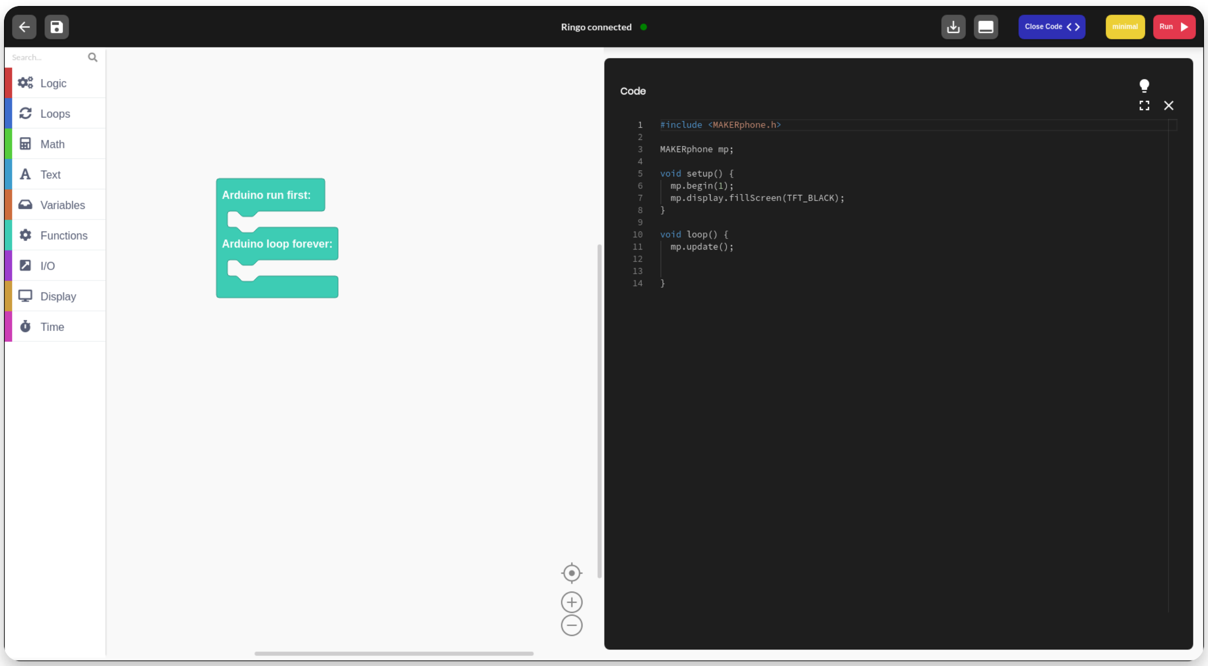


**Code** project is pretty much the same thing you'll get in **Arduino IDE** - straight up

Arduino C/C++ that will run your programs based on the code you've written.

On the other hand, **block** projects are the real ones here. There you'll be able to generate program code from the **blocks that you drag-and-drop.** This is a real Scratch-like experience and it is highly recommended to everyone beginning their programming career or learning the ins and outs of the Ringo library. The code that is generated can then be copied and modified in the code project.

**We advise that you start the block project regardless of your programming skills just so you can get to know the phone and the firmware better!**



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

}
```

This is the main interface you will be looking at most of the time.

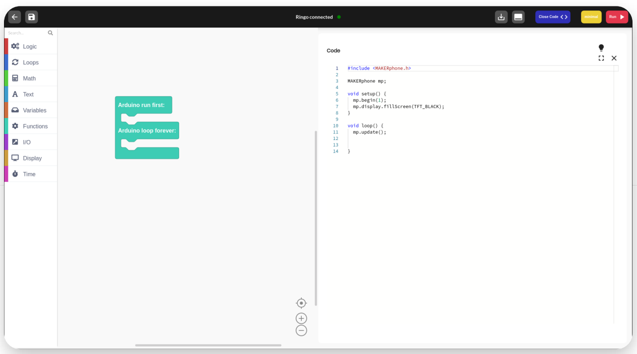
On the **top of the screen**, there is a **toolbar** from which you can access main program functionalities.

The **block selection** tool is located on the **left side**.

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

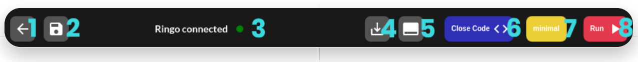
On the **right** is where those drawings will be translated into **code**. It is the same code editor used in the VS Code, but as of right now, it is read-only and non-editable.

If you don't like the dark mode (blasphemy if you ask us!), you can easily switch it by pressing the light bulb button in the top right corner of the code editor.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

}
```

## Toolbar



There are eight main components here:

1. **Back to the main menu** – returns to the main menu without saving
2. **Save/Save As...** – saves the file in the default sketch directory
3. **Ringo connected indicato**r – indicates whether the phone is connected or not
4. **Export to binary** – creates .bin file of the code which can be directly uploaded to the phone
5. **Open serial** - open Serial port
6. **Close Code** – closes the code editor to expand the 'drawing' area
7. **Minimal** - Toggles minimal option
8. **Run** – compiles and uploads the code to your phone

When the phone is not connected, the red 'Run' button will turn grey and the indicator will say '**Ringo disconnected**' with the red dot instead of a green one.

While a code you've written is being uploaded, a progress bar will appear right below the toolbar which indicates how much data has been compiled/uploaded so far.

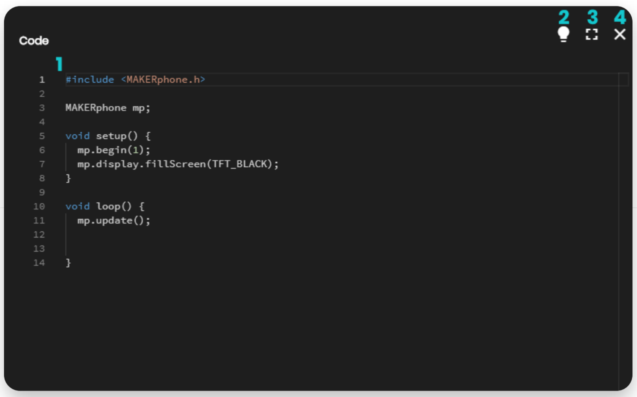
When it reaches 50% it means that the compilation is over and when it reaches 100% it means that the upload to the phone is finished.

✓

## MINIMAL BUTTON

One important thing to know about the yellow button 'Minimal' is that it does the following - compiling it with the minimal on, your program will compile much faster, but it will not have the basic phone functions - calling, texting, main menu... This is particularly useful when developing an app and when you're fixing small bugs - we recommend using it ONLY when developing an app.

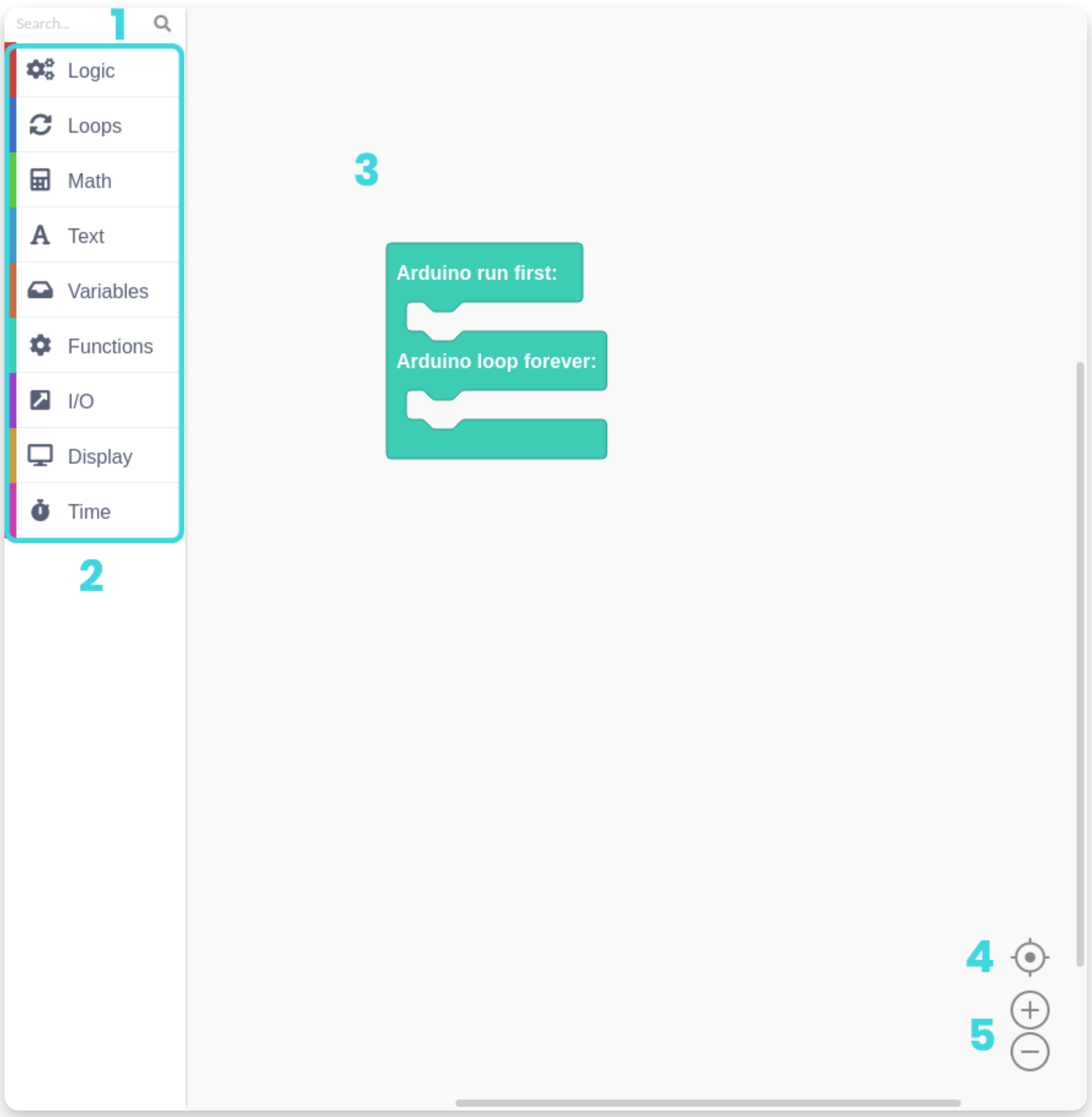
1. **Main code screen** – this is where the blocks will appear in the textual format, code words are colored while the regular text is white
2. **Light mode switch** – switch the background of the code editor between black and white
3. **Expand** – stretches the code editor over the whole window
4. **Close** – closes the code editor, same functionality as 'Close Code' button from the toolbar



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

If you are writing some more complex apps, you can copy this code and paste it to one of the other, more complex IDEs (like Arduino or VS Code) from where you will be able to edit it and write more lines.

The 'Drawing board' is the most complex part of the IDE. It's where all the magic happens. It is divided into two main sections. On the left is a board where you select the blocks and on the right is a board where you place them. Each type of block has its own color so it's easily recognizable.



1. **Search bar** – dynamic search bar with which you can easily find any

component you're looking for

2. **Component selector** – divided into categories by names and colors
3. **Drawing board** – a place where you create your programs by placing the components in a certain order
4. **Center icon** - places your blocks in the center of the board
5. **Zoom buttons** - zoom in and out of the board

Each of the components will be explained in detail and come with a few examples of how to use them together

# #3 - Types of blocks

There are a total of **nine** block types in CB. Each of them is represented by their own 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 for starting the software, initializing and declaring variables and running functions that only have to run once (ex. Intro screen in a video game).

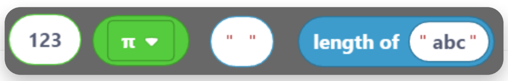
The **void loop()** is where everything else takes place. It basically runs every bit of code inside it over and over again (speed depends on the device – just imagine it's ultra-fast!). It should pretty much follow the refresh rate of the screen and make the program do things accordingly.

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

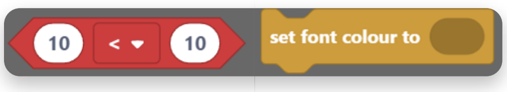
If you wish to put something in the **void setup()**, you have to drag the main block from **Functions** and place your blocks inside as you wish, but more on that a little bit later.

## 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 that have elliptical shape return either integer or float values.

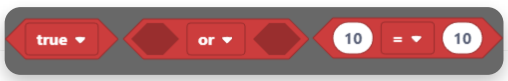


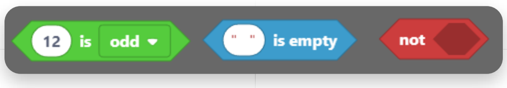
When ever you find circular "holes" inside some blocks, that is the place where variables can be inserted. It's most commonly found in **comparison** or **action** blocks.

## Triangular blocks

Boolean variables are represented by triangular blocks.

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



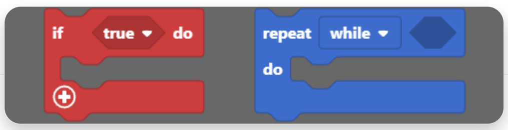


Regardless of color, each of these blocks returns either true or false.

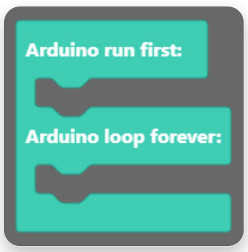
Triangular "holes" require boolean blocks to be inserted.

## 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 first have to be placed inside of the building blocks in order 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 that is placed inside **Arduino run first** goes into **void setup()** and everything that is placed inside **Arduino loop forever** goes into a **void loop().**

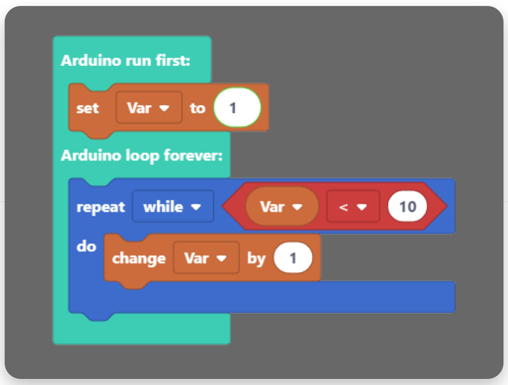
## Inserting blocks

Now, this is the main part.

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

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

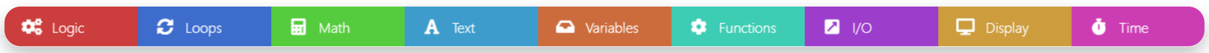
Here is an example of a program that will set the variable **Var** to **1 and then increase that variable while it is smaller than 10**.



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.

# #4 - 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 maximum two clicks.

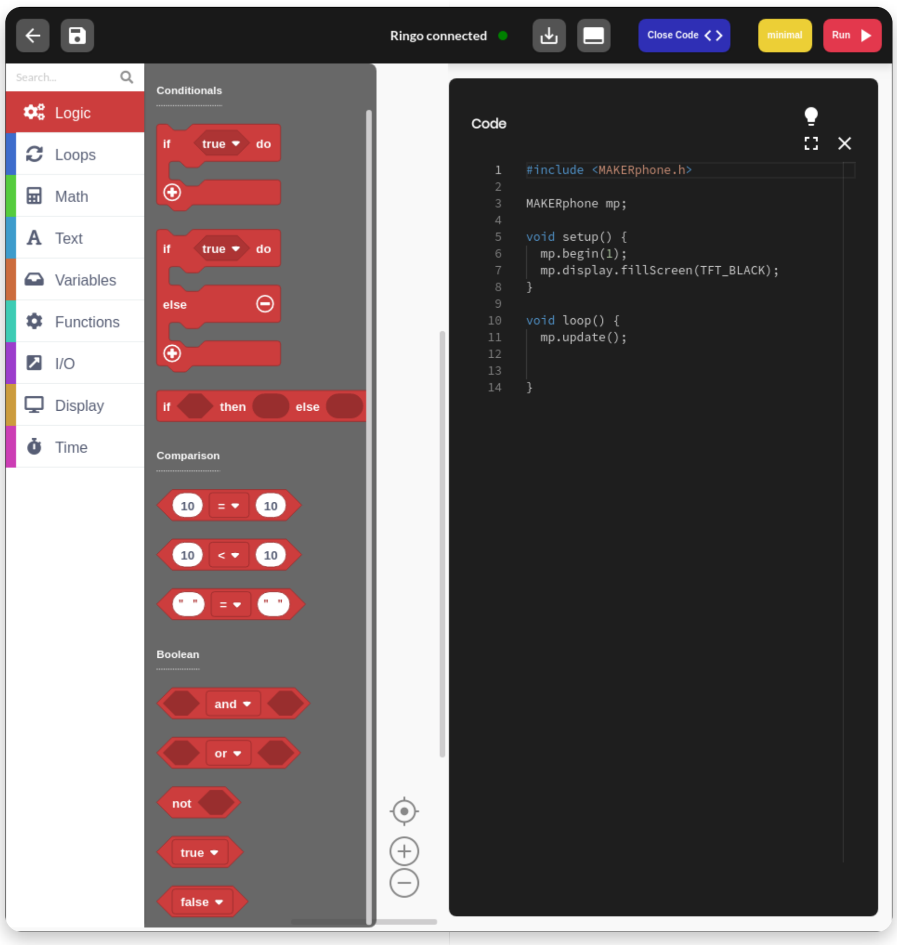
Sections themselves are pretty explanatory but we'll go through them all just so you can get a little bit better understanding of the whole concept.

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, as well as comparisons, **and/or, not, true/false** and other logical operators.

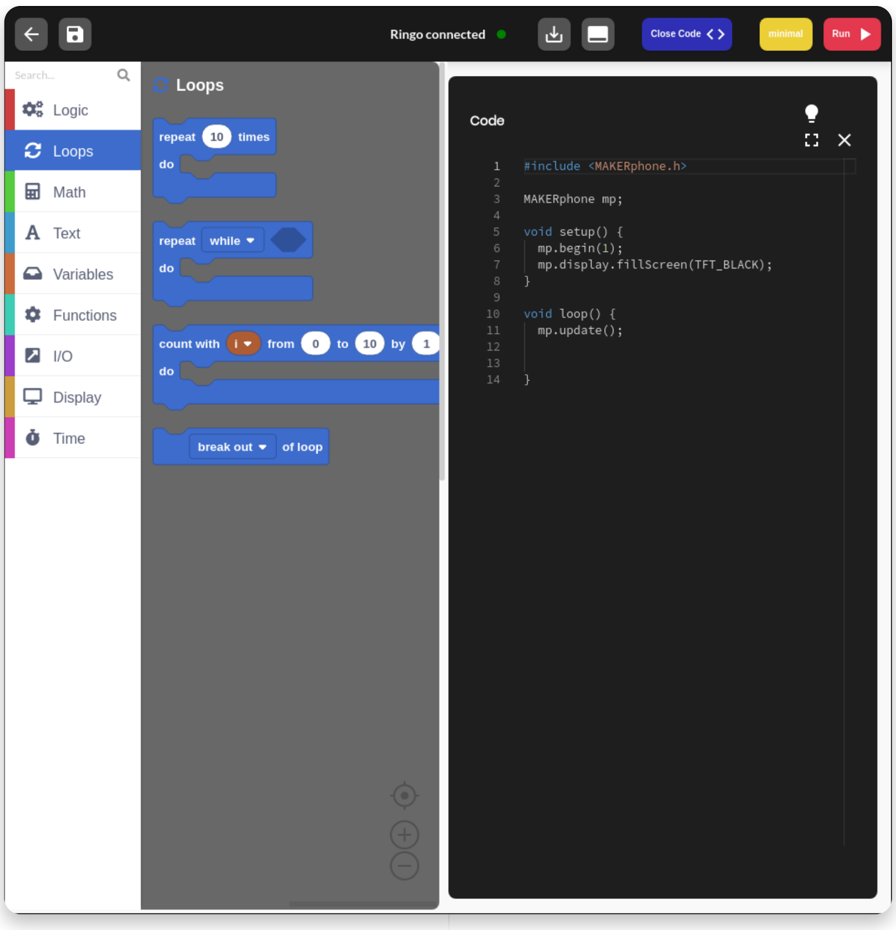


```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Loops

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

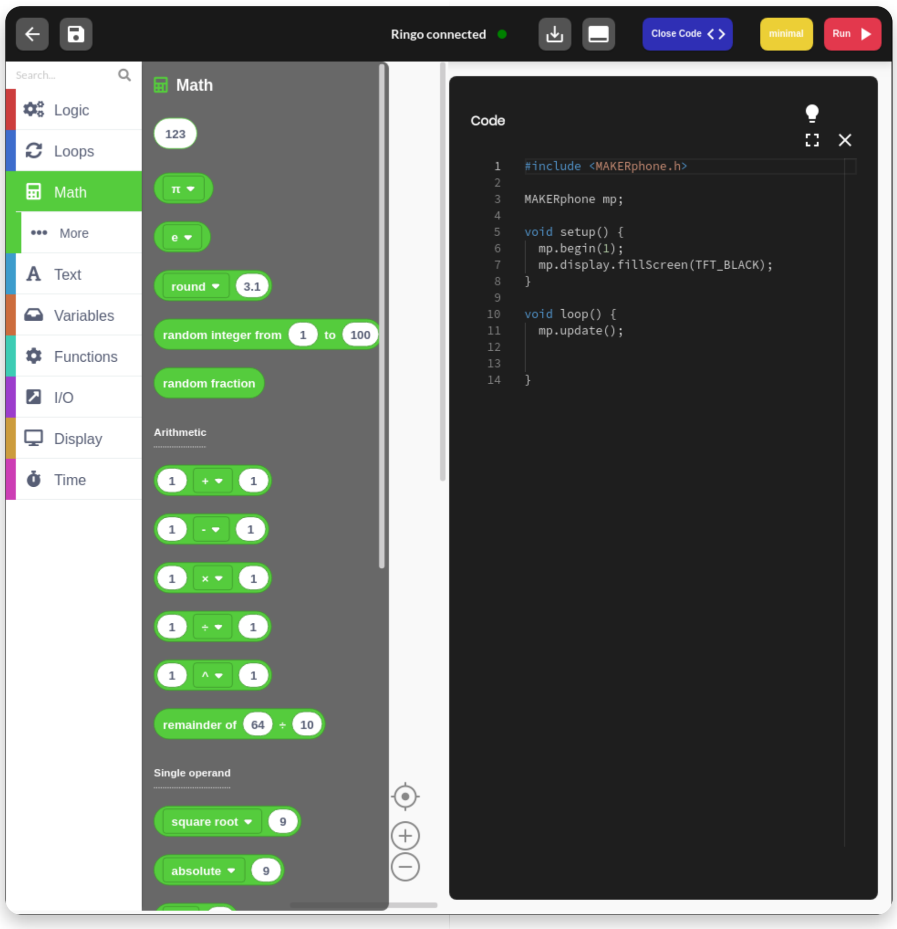
They can either have conditionals, and repeat for as long as that condition is met, or they can have a pre-determined amount of repeats.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Math

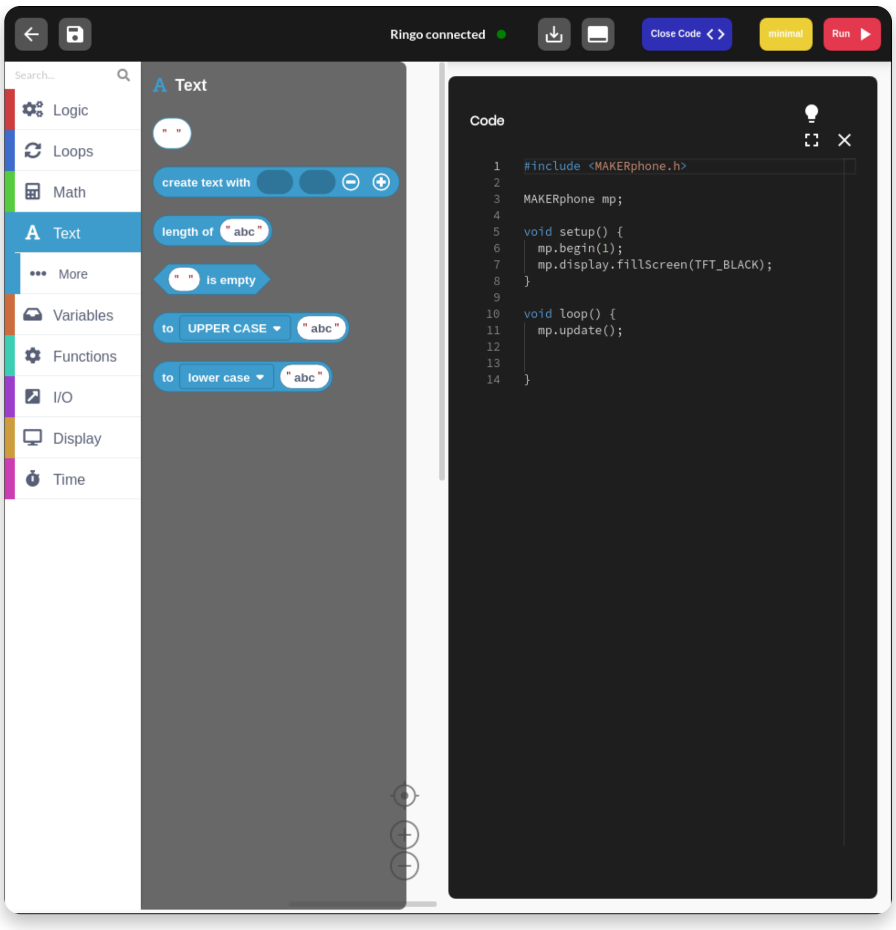
Pretty much every math function is located here. From basic operations to rounding numbers and working with angles, you will easily trigger your inner Einstein or Pythagora in a matter of seconds!



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Text

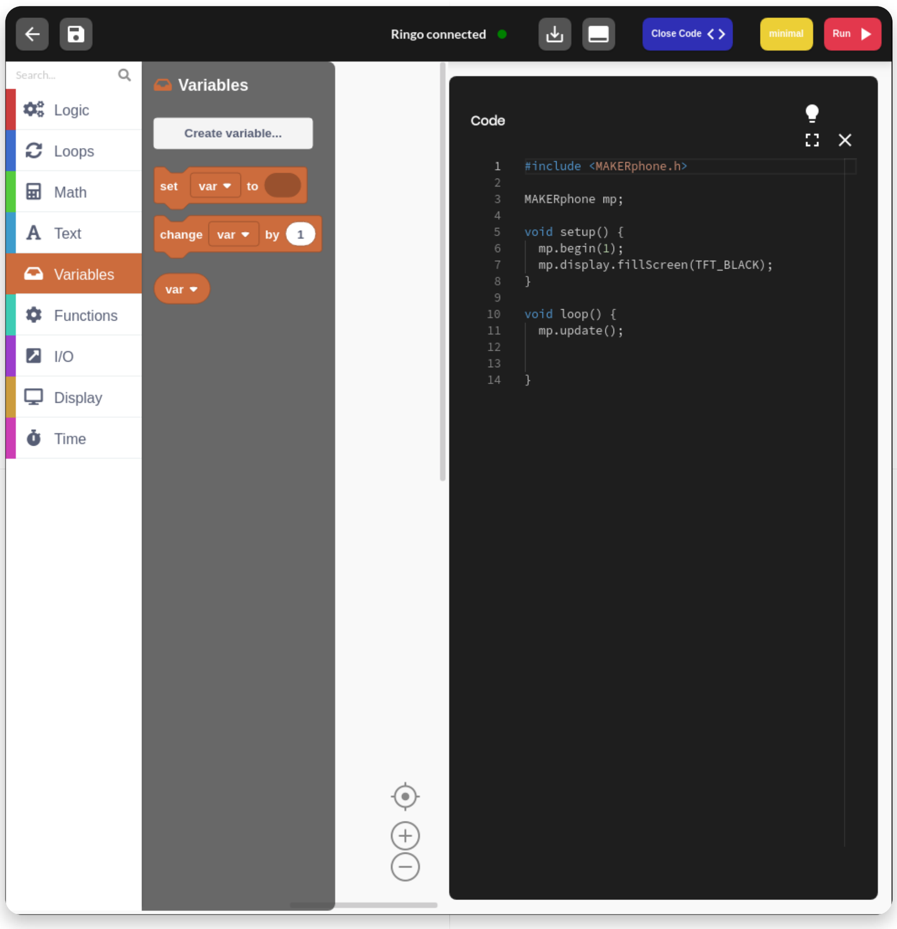
Strings, characters, and string manipulation. Great place for creating new text and implementing it to your programs.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Variables

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

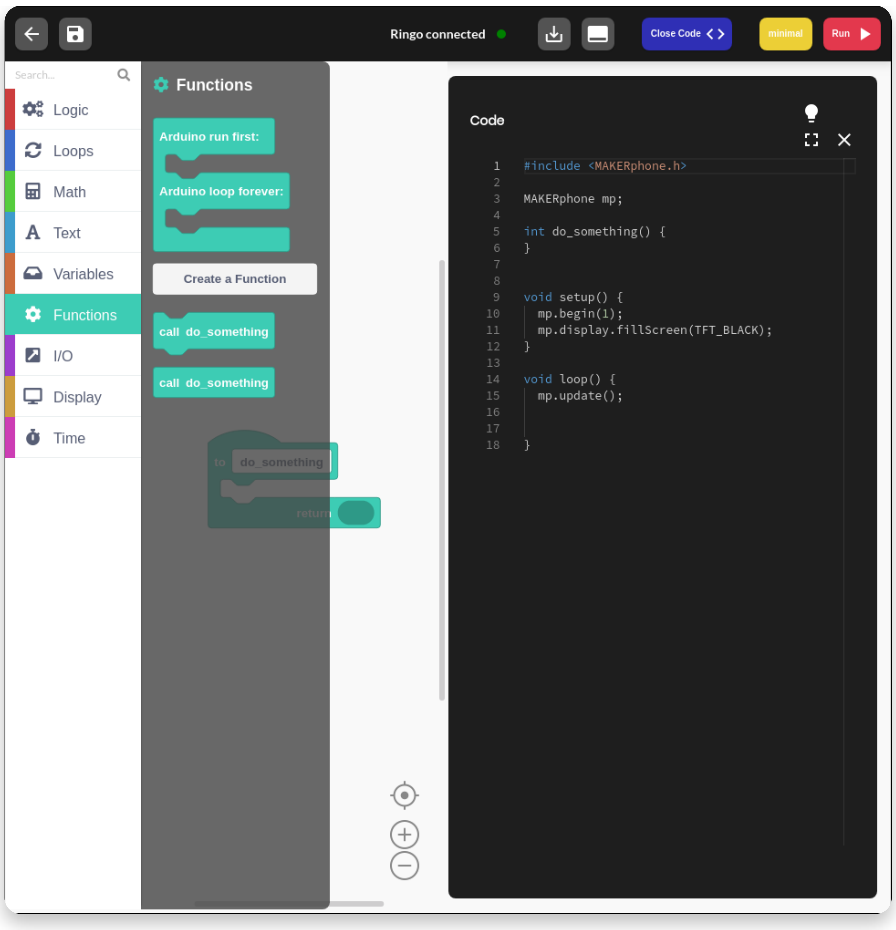


```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Functions

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

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

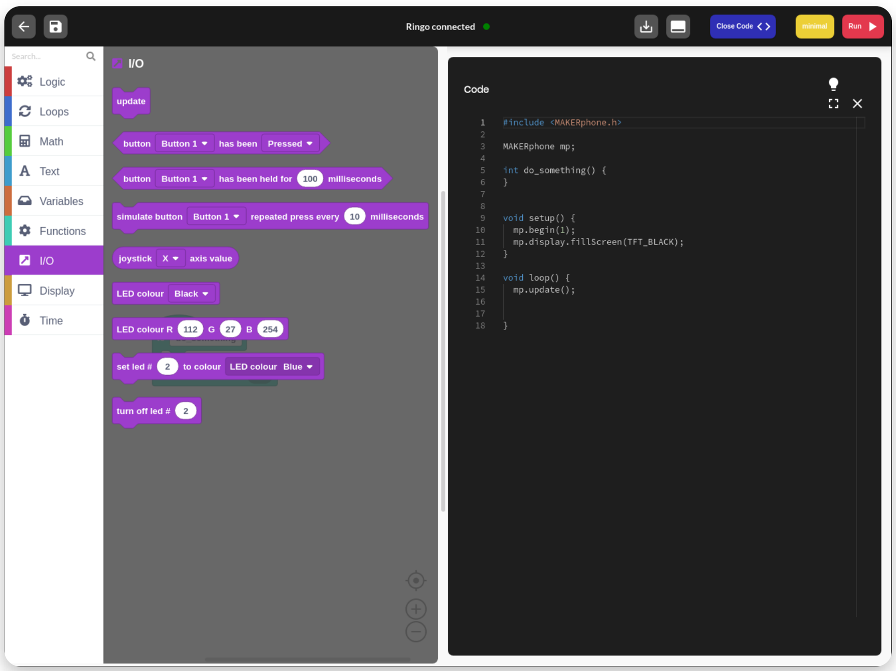


```
#include <MAKERphone.h>

MAKERphone mp;

int do_something() {
}


void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Input/Output

Everything regarding Ringo's components is located here.

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

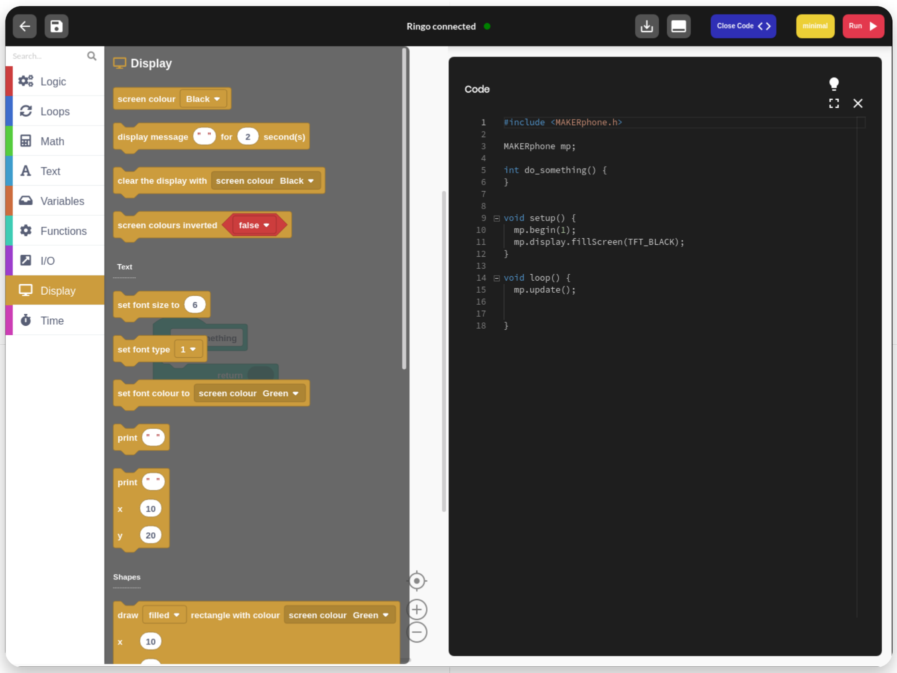

Ringo connected
Close Code
minimal
Run
Search...
Logic
Loops
Math
Text
Variables
Functions
I/O
Display
Time
I/O
update
button Button 1 has been Pressed
button Button 1 has been held for 100 milliseconds
simulate button Button 1 repeated press every 10 milliseconds
joystick X axis value
LED colour Black
LED colour R 112 G 27 B 254
set led # 2 to colour LED colour Blue
turn off led # 2
Code

```
#include <MAKERphone.h>

MAKERphone mp;

int do_something() {
}


void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Display

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

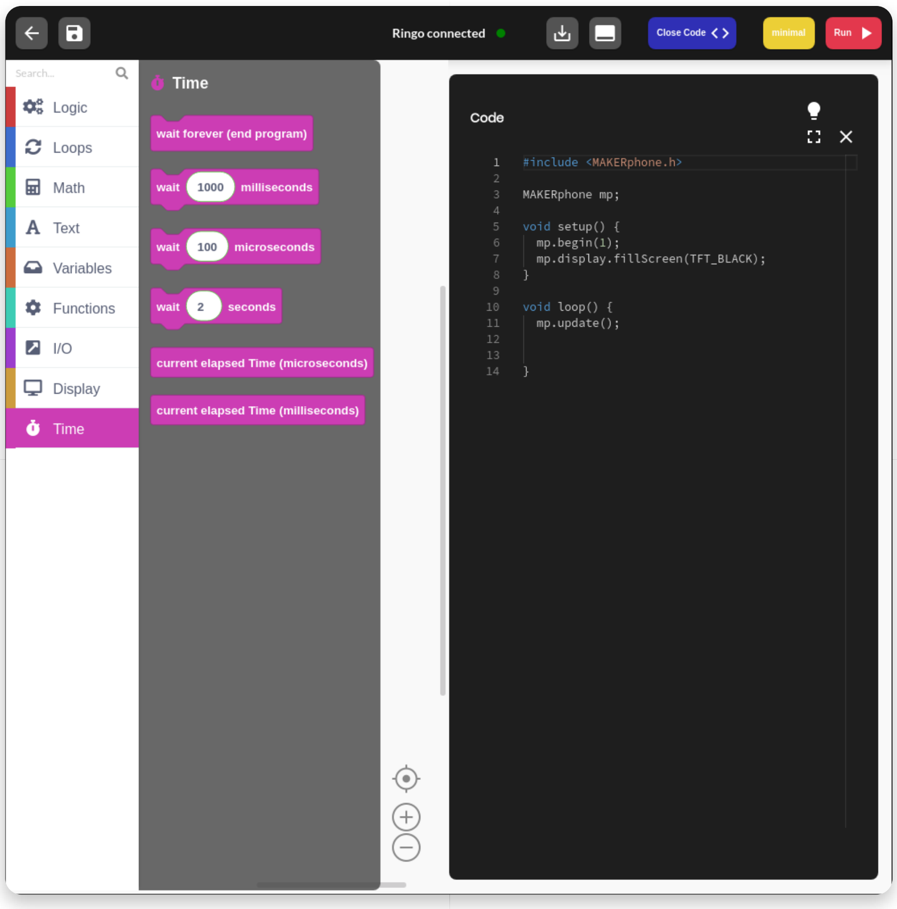
Here is where all the magic translates to those colored pixels. You can create so much through these blocks.



```
#include <MAKERphone.h>

MAKERphone mp;

int do_something() {
}


void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Time

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



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Search bar

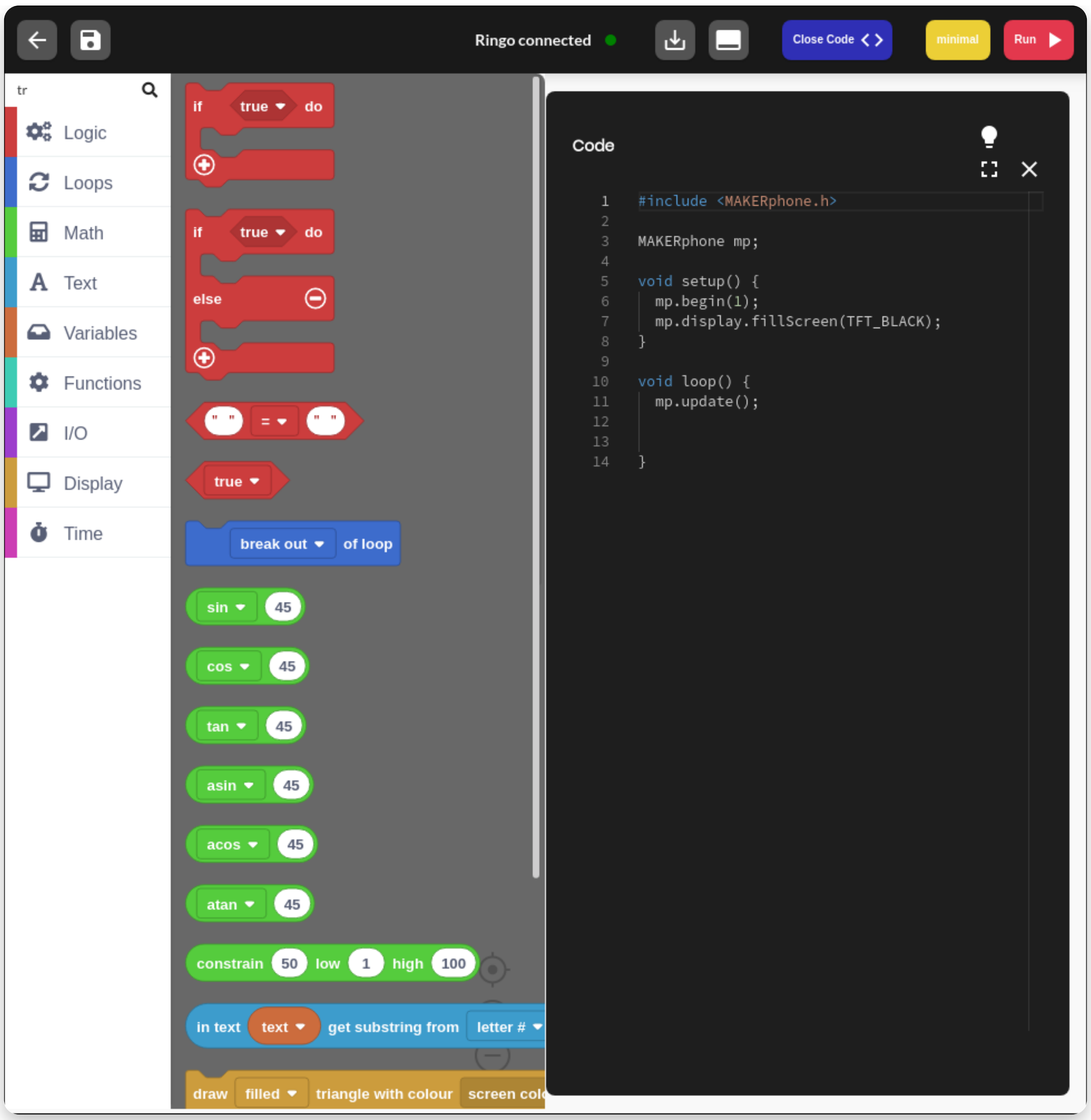
There is also a **search bar** above all function sections to ease the search for that one specific block you just can’t seem to find (where is that PRINT???).

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...



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();


}
```

## Using blocks

## Changing screen color

Now that you're accustomed to every type of block, it's time to learn how to use them!

There will be a series of small examples where you'll be able to get an understanding of how the Ringo library works.

Each example will show different functionalities. In the end, we'll just bring all of that together to create a cool app!

**Let's begin!**

### Example #1

Let's kick things off as simple as possible.

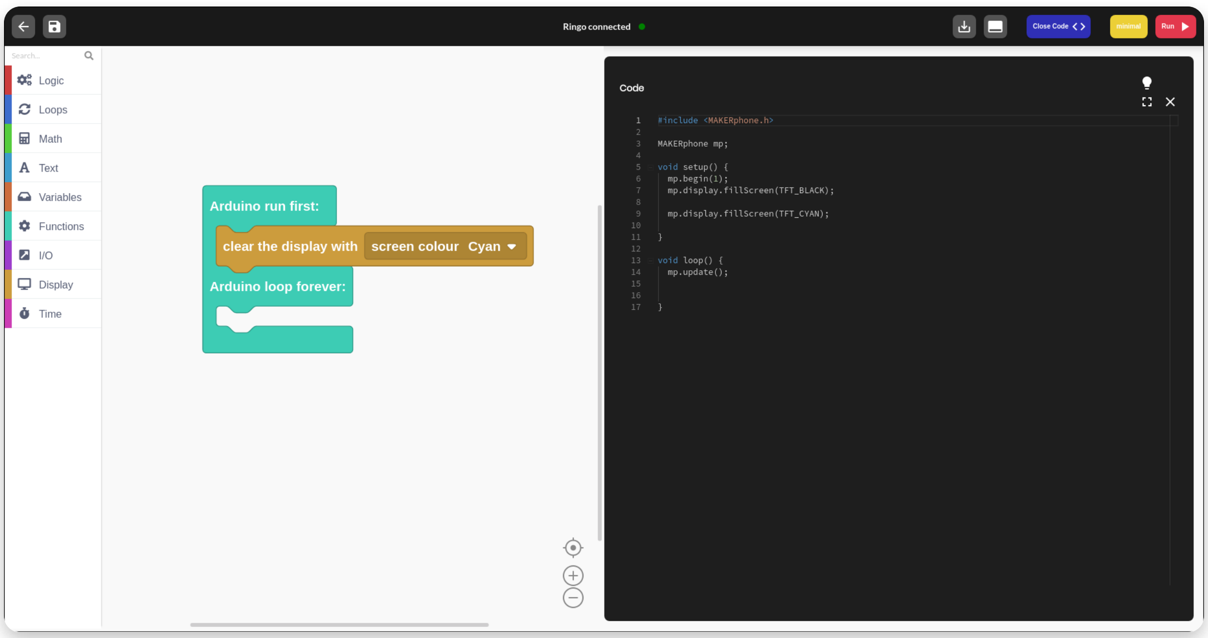
The main component on Ringo is definitely the screen since working without one would be

pretty much impossible.

What we want to do as a test is to change the screen color of our program.

The default screen color is **black**, or as it is known in our library, **TFT_BLACK**. All

colors from this
library are labeled with **TFT_** prefix because they are made to be used on TFT screens.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  mp.display.fillScreen(TFT_CYAN);

}

void loop() {
  mp.update();


}
```

The first function that is there by default is **mp.display.fillScreen(TFT_BLACK)**. It acts as an **eraser - everything will be wiped off the screen at the beginning of the program**.

Now let's move to the code generated by the blocks.

Since we want to turn our screen **cyan**, we are calling another fillScreen() function from the **Display section**, just with a different parameter.

It's only necessary to do this once, so we're placing it in the **Arduino run first**: **a section which is equivalent to the void setup().**

Placing it in the loop part would call the functions multiple times a second which is in this example a complete waste of time.

Now that we've made sure everything is set, we can run the program. It will first be **compiled and then uploaded** to the phone.

Ringo's screen should now be cyan. It's really as simple as that.

This is just a beginning but you can imagine that the possibilities of this screen are countless.

✓ **Colors**

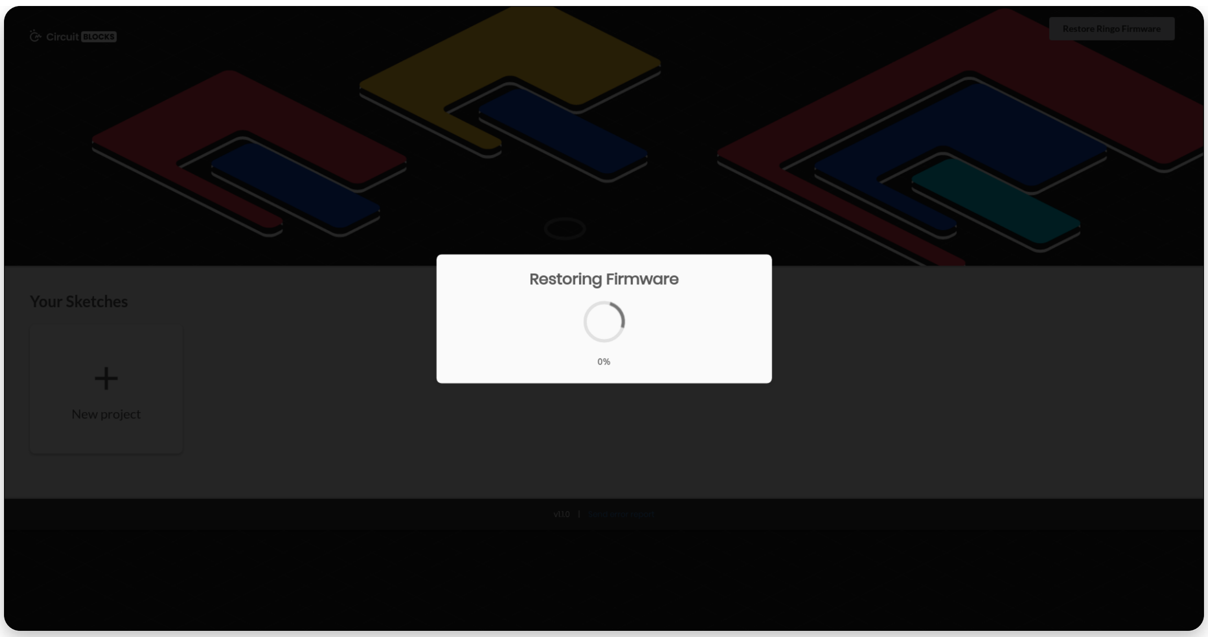
The full list of available colors can be found in the Display section

## Restoring defaults

Now, what if we want to stop making cool things and just use our phone? Well, it takes only a couple of clicks to get everything back to normal.

**Anytime you want to restore the original Ringo firmware to your phone, you can do it by pressing the 'Restore Ringo firmware' button on the main menu.**

**Just don't forget to save the project before exiting!**



Now that we've covered the basics, let's head out to something a little bit more advanced.

## Controlling the display

### Scrolling colors

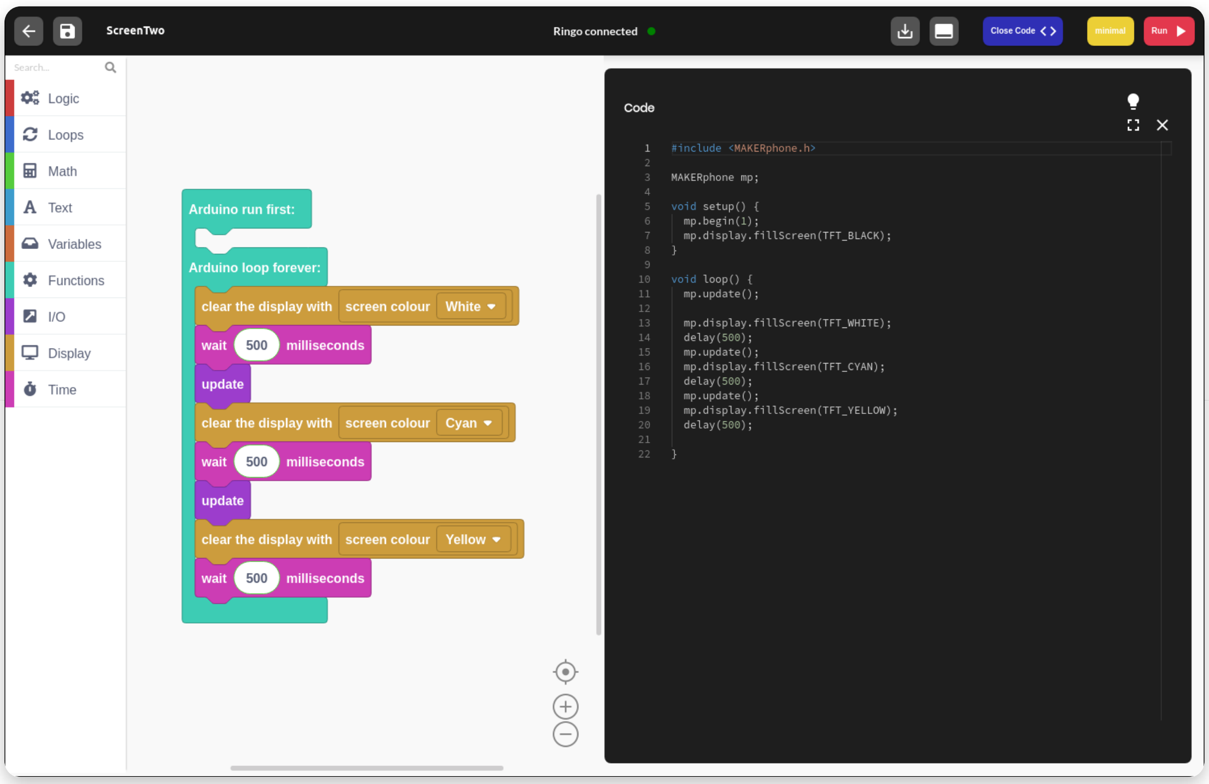
Now we're going to use the **loop()** part of the code.

See that **mp.update() function**? It's probably the most important function of all and it's located under the **I/O section.**

Its mission is to read everything that has happened between the last time it was called and now and to transfer those changes to the hardware.

For example, fillScreen() function would not change anything if the mp.update() isn't called.

**Besides the screen, it also refreshes buttons, the speaker, LEDs and pretty much every other component on the phone.**



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  mp.display.fillScreen(TFT_WHITE);
  delay(500);
  mp.update();
  mp.display.fillScreen(TFT_CYAN);
  delay(500);
  mp.update();
  mp.display.fillScreen(TFT_YELLOW);
  delay(500);

}
```

One mp.update() is there by default but for this program, we've added two more.

We've also used another new block called **wait()**.

**It translates to the delay() function which stops everything for a certain amount of milliseconds.**

The number value block can be found in the Math section and its value can be changed to suit our needs.

Since the loop() function is really fast, sometimes we need this delay to actually see what is going on on the screen.

Using delay when **expecting a fast and responsive program**, however, **is not the way to go**,
especially when we're using buttons. More about that in the next lesson.

What we're actually doing in this program is alternating the screen color between three values.

Firstly we change it to white, then wait 500 milliseconds or 0.5 seconds.

Then, we change it to cyan and wait another 500 milliseconds.

Lastly, we change it to yellow, wait 500 milliseconds and return back to the beginning of the loop, where the screen is again changed to white.

**Notice that we're calling the mp.update() function after each color change in order to transfer that color to the screen.**

✓ **Update function**

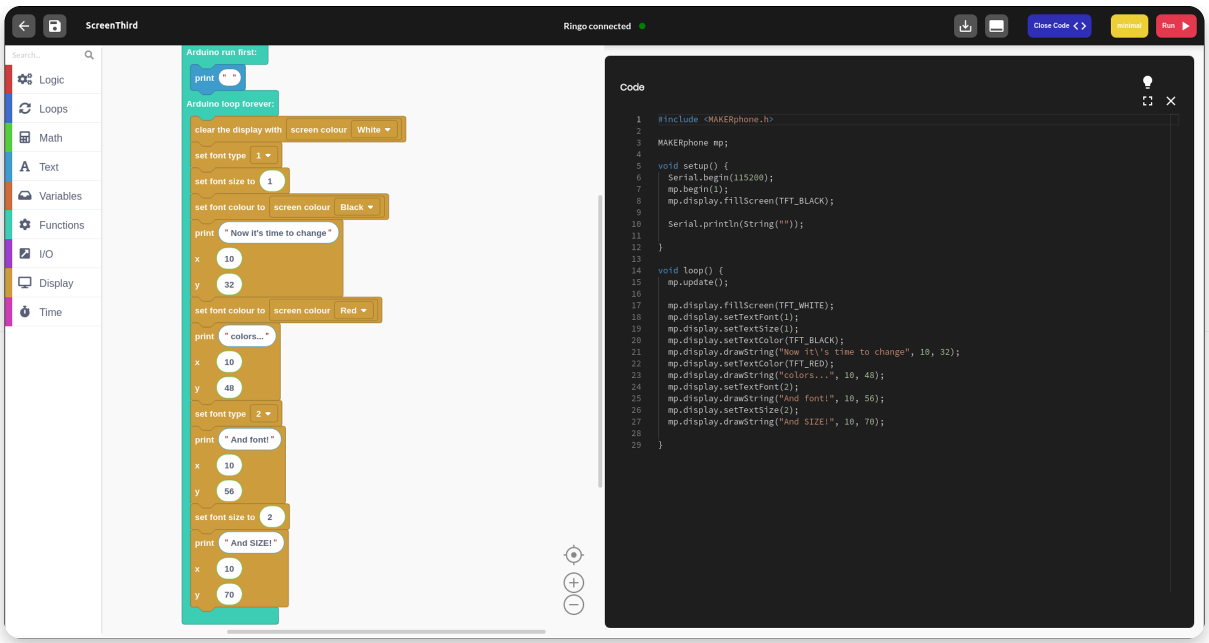
Be careful when using mp.update() since it is not the fastest function.Calling it too many times in one loop will slow down the program significantly - USE IT WELL!

## Writing out some text

Now that we know how to change the background color, it's time to start writing something on that canvas.

**Ringo library offers three different fonts that can be re-sized and re-colored as we like**.

Functions for that are rather simple and should not be a problem to understand for anyone.



```
Arduino run first:
  print " "
Arduino loop forever:
  clear the display with screen colour White
  set font type 1
  set font size to 1
  set font colour to screen colour Black
  print " Now it's time to change "
    x 10
    y 32
  set font colour to screen colour Red
  print " colors... "
    x 10
    y 48
  set font type 2
  print " And font! "
    x 10
    y 56
  set font size to 2
  print " And SIZE! "
    x 10
    y 70
```

Code

```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  Serial.begin(115200);
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  Serial.println(String(""));

}

void loop() {
  mp.update();

  mp.display.fillScreen(TFT_WHITE);
  mp.display.setTextFont(1);
  mp.display.setTextSize(1);
  mp.display.setTextColor(TFT_BLACK);
  mp.display.drawString("Now it\'s time to change", 10, 32);
  mp.display.setTextColor(TFT_RED);
  mp.display.drawString("colors...", 10, 48);
  mp.display.setTextFont(2);
  mp.display.drawString("And font!", 10, 56);
  mp.display.setTextSize(2);
  mp.display.drawString("And SIZE!", 10, 70);

}
```

With this program, we're writing some words out on the screen in **different fonts, sizes, and colors.**

The font type is easily changed with the **drop-down menu** of the set font type block. The font size, however, **multiplies the size of the selected font** by the desired value. **By default, both of those values are 1,** so setting them at the beginning of the program to that same value actually made no difference.

Later both of those values have been changed. In the last example, we're printing out two words **"And SIZE!"** in a different font that is **twice the size.**

The **print()** function here is something that we're going to use quite often. The first empty slot is a string of characters that are meant to be written out.

You can find an empty one in the **Text section**.

The second and third slot are the location of the first letter. **Setting both of those values to 0 will print out words in the upper left corner of the screen.**

With these numbers, we're actually referencing **pixel locations**. **Our screen is 128x160 pixels** and we can manipulate each and every one of them.

**Everything that we write/draw between the x values of 0 and 127 and y values of 0 and 159 will be visible on the screen.**

If any of those values is out of that range, we will not see the component.

## Shapes and animations

### Shapes

There are plenty of blocks that are still unexplained but offer so many different possibilities.

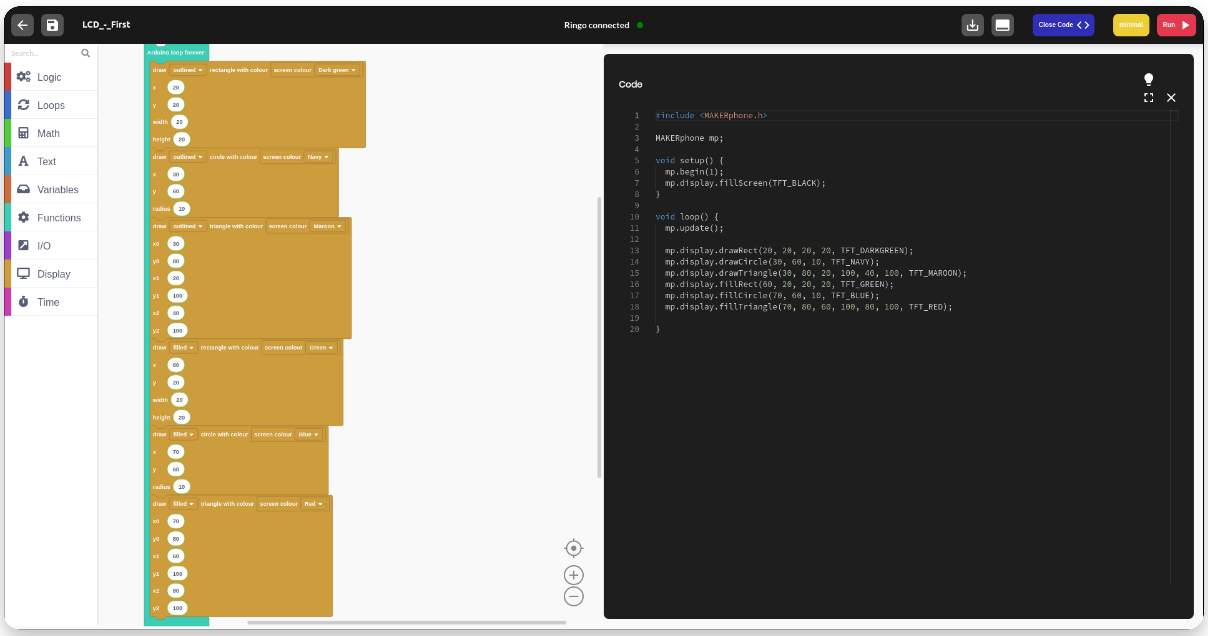
For now, let's focus on ones from the **Display** section.

We know how to paint the entire screen with one color, but what about only the part of it?

There is a function for drawing some of the most used shapes - r**ectangle, circle, triangle, and ellipse.**

All of these can be either filled or not while using the same colors as for the screen.

Here is an example of one such program.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  mp.display.drawRect(20, 20, 20, 20, TFT_DARKGREEN);
  mp.display.drawCircle(30, 60, 10, TFT_NAVY);
  mp.display.drawTriangle(30, 80, 20, 100, 40, 100, TFT_MAROON);
  mp.display.fillRect(60, 20, 20, 20, TFT_GREEN);
  mp.display.fillCircle(70, 60, 10, TFT_BLUE);
  mp.display.fillTriangle(70, 80, 60, 100, 80, 100, TFT_RED);

}
```

There are some variables that might cause some question marks here.

All of these functions have a **location, size and color parameters**.

The first two parameters for a **rectangle** are the location of its **upper left corner**. The **next two** are its **width and height**. This means that it will occupy the area on the screen which **starts in the coordinates x and y** and spread all the way to the **x+width and y+height**.

**Circle,** on the other hand, is defined by the location of its **center**.

**X and y** represent the center and **radius** just how far from it are the furthest points.

**Triangle** is defined by the coordinates of its **vertexes**. It doesn't matter in which order you put them, every triangle with the same coordinates will look the same.

You can combine those shapes, overlap them, draw them outside of the screen and many more things. Pretty much every more complex shape can be created with the basics ones.

## Animations

We already talked about how the screen refreshes 25 times in a second, which is fast enough to create the effect of a moving image.

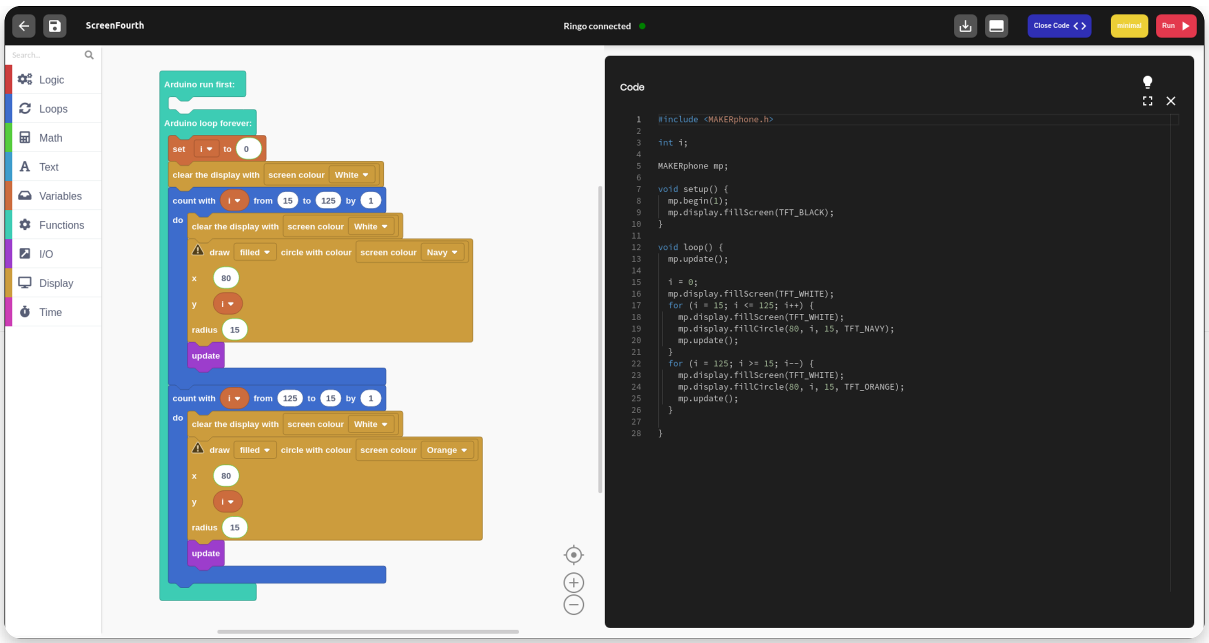
**Animations** are just that - shapes and images moved so slightly that they have a smooth movement.

It's best that we show it by example.

We're going to take a circle and move it from the top of the screen to the bottom. When it reaches the bottom, it will change its color and direction in which it moves.

For this one, a new block is going to be introduced - **a for loop**. It's a pretty basic function in programming and you will use it quite often.

This loop and many more can be found in the **Loop section**, the second one on the list.


ScreenFourth
Ringo connected
Close Code
minimal
Run
Logic
Loops
Math
Text
Variables
Functions
I/O
Display
Time
Arduino run first:
Arduino loop forever:
set i to 0
clear the display with screen colour White
count with i from 15 to 125 by 1
do
draw filled circle with colour screen colour Navy
x 80
y i
radius 15
update
count with i from 125 to 15 by 1
draw filled circle with colour screen colour Orange
Code

```
#include <MAKERphone.h>

int i;

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  i = 0;
  mp.display.fillScreen(TFT_WHITE);
  for (i = 15; i <= 125; i++) {
    mp.display.fillScreen(TFT_WHITE);
    mp.display.fillCircle(80, i, 15, TFT_NAVY);
    mp.update();
  }
  for (i = 125; i >= 15; i--) {
    mp.display.fillScreen(TFT_WHITE);
    mp.display.fillCircle(80, i, 15, TFT_ORANGE);
    mp.update();
  }

}
```

What exactly does **for loop** do? It repeats whatever is inside it a specific amount of times.

As we can see in the example, we set the **i variable to 15** and increase it by 1 every time the loop loops.

When it gets to **125, the for loop breaks and the program continues.** That means that everything inside it ran **111 times**!

But what does this exactly have to do with animation? Well, we used that same i as one of the coordinates for the circle.

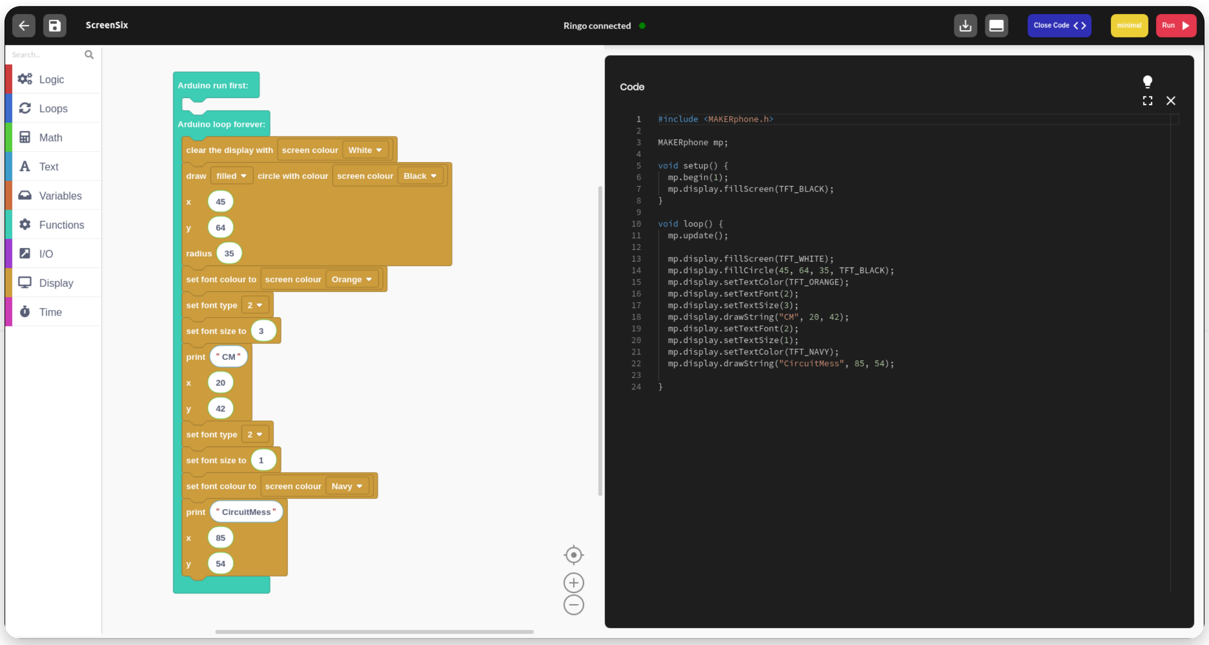
Every time the loop ran, the circle went down by 1 frame, but the loop ran so fast that we only saw the smooth movement of the circle.

## Combining text and shapes

Here is a quick example of using both text and shapes at the same time. Both shapes and text can be combined, with one drawn over the other.

Of course, on the top will always be the one whose function was called later in the loop.

In this example, we're recreating our **CircuitMess** logo by using simple shapes and text.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  mp.display.fillScreen(TFT_WHITE);
  mp.display.fillCircle(45, 64, 35, TFT_BLACK);
  mp.display.setTextColor(TFT_ORANGE);
  mp.display.setTextFont(2);
  mp.display.setTextSize(3);
  mp.display.drawString("CM", 20, 42);
  mp.display.setTextFont(2);
  mp.display.setTextSize(1);
  mp.display.setTextColor(TFT_NAVY);
  mp.display.drawString("CircuitMess", 85, 54);

}
```

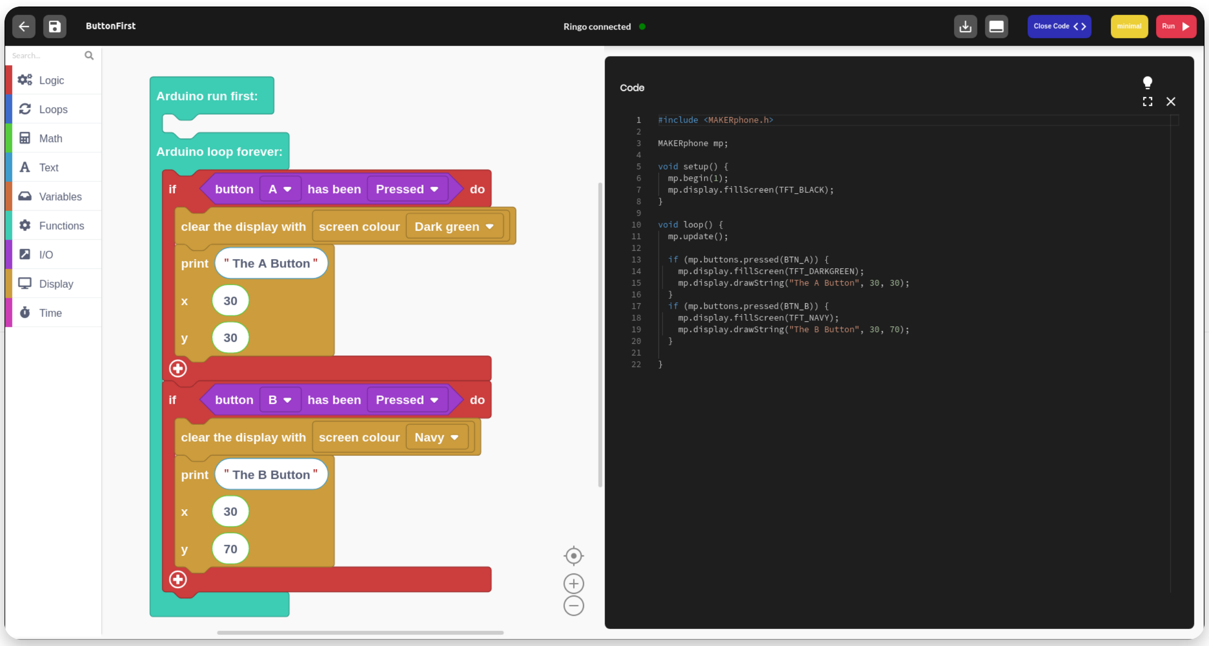
You can try adding some animations to it and make it come to life!

# Pressing the buttons

Every Ringo is equipped with **18 buttons and a two-axis joystick**, so it would be a shame not to use them.

We're going to focus here on using the functions of drawing on the screen, but this time we're controlling it with a simple press of a button.

This gives us much more flexibility and dramatically improves the possibilities of a program.



```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  if (mp.buttons.pressed(BTN_A)) {
    mp.display.fillScreen(TFT_DARKGREEN);
    mp.display.drawString("The A Button", 30, 30);
  }
  if (mp.buttons.pressed(BTN_B)) {
    mp.display.fillScreen(TFT_NAVY);
    mp.display.drawString("The B Button", 30, 70);
  }

}
```

Here is the first time we're introduced with the most basic logical function of all - **if**. This one, along with other logical functions can be found in the **Loop** section at the top of the section list.

What **if** does is it checks the condition and if the condition is **true**, executes the code inside it.

If the condition is not met and considered **false**, the program will not execute the code inside the function.

There is also a more advanced version of this function called **if-else**, which executes another part of the code if the original condition is not met.

Our conditions in these examples are checking whether the **buttons have been pressed or not**.

For that, we're using the function **mp.buttons.pressed(BUTTON)** whose block can be found in the **I/O section**.

This function will return a true boolean value if the specified button has been pressed and false if it hasn't.

It can also be modified that it checks whether the button has been **released**.

Every time you press the button the screen changes color and you get an indicator which button has been pressed.

Pretty much every button works this way, besides, of course, **the joystick**.

## Joystick

This component works very differently than the buttons.

It has two main values, **X-axis and Y-axis**. Both of those variables can take a value **between 0 and 1023**.

When the joystick is in the middle, they will take the middle value. Since the precision of it isn't perfect, that will be somewhere around **500**.

As you move the joystick **left the value of the X-axis increases and while moving it to the right, decreases**.

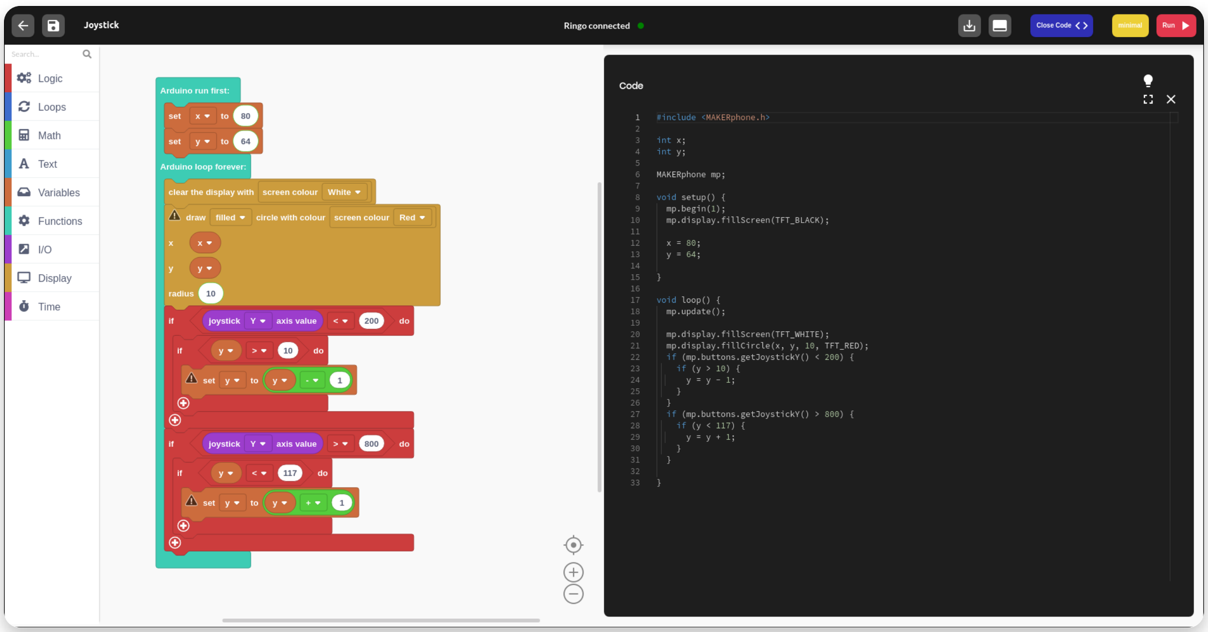
Moving the joystick up decreases the **Y value and moving in down increases it**.

This logic is a little bit different than the default screen x and y pixel location values, so be careful when working with it.

### Where is joystick right now?

Function mp.buttons.getJoystickX() and mp.buttons.getJoystickY() returns the current value of those variables, thus allowing you to check the joystick's current location.

Let's see this in an example.



```
#include <MAKERphone.h>

int x;
int y;

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  x = 80;
  y = 64;

}

void loop() {
  mp.update();

  mp.display.fillScreen(TFT_WHITE);
  mp.display.fillCircle(x, y, 10, TFT_RED);
  if (mp.buttons.getJoystickY() < 200) {
    if (y > 10) {
      y = y - 1;
    }
  }
  if (mp.buttons.getJoystickY() > 800) {
    if (y < 117) {
      y = y + 1;
    }
  }

}
```

Every time the joystick is pushed up or down, the circle will change its location until it gets to the edge of the screen.

Green **Math functions** are used for changing variable values, just like we did here.

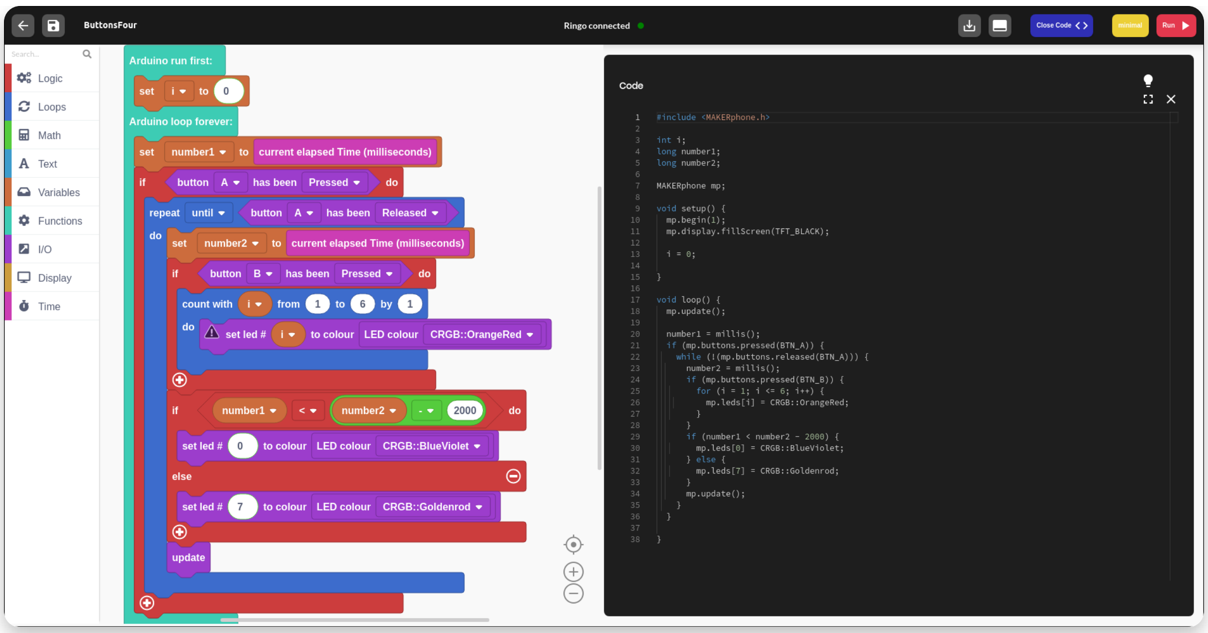
The code is simple enough and it should be no problem to understand it.

## Buttons and time

Just pressing the buttons is cool, but what about **long presses**? Can you create a program that does different things depending on how long the button has been pressed?

Well, of course you can!

The logic behind this next example is a bit more complex, so make sure everyone pays close attention.



```
#include <MAKERphone.h>

int i;
long number1;
long number2;

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  i = 0;

}

void loop() {
  mp.update();

  number1 = millis();
  if (mp.buttons.pressed(BTN_A)) {
    while (!(mp.buttons.released(BTN_A))) {
      number2 = millis();
      if (mp.buttons.pressed(BTN_B)) {
        for (i = 1; i <= 6; i++) {
          mp.leds[i] = CRGB::OrangeRed;
        }
      }
      if (number1 < number2 - 2000) {
        mp.leds[0] = CRGB::BlueViolet;
      } else {
        mp.leds[7] = CRGB::Goldenrod;
      }
      mp.update();
    }
  }

}
```

Here we see the appearance of one very important function - **millis()**. What millis() does is it returns some unknown time value depending on when it's called.

Even though we cannot do much with the exact value, we can use it for comparison.

The timer starts counting milliseconds every time we turn on the phone, so when this function is called, we'll get a specific timestamp back.

Then we can call that function a little bit later to determine how many milliseconds have been since the last time we've called it.

### Comparing two timestamps

The values are compared just like two regular numbers. So if we want to know has it been more than two seconds between the two calls, just add 2000 milliseconds to the first value or subtract 2000 milliseconds from the second value before comparing.

In this example, we enter the specific subsection of the code **only after the number one has been pressed for two seconds.**

At that moment the LED color changes and we get a cool effect. You can use this function in many ways, not only for buttons.

## Flashing LEDs

The most important functions for this lesson can be found in the **I/O section**.

Also, we're going to use some loops as well as time delays to create some cool effects on the back of the phone.

## Simple light

LEDs are actually all located in one array called **mp.leds** that has a length of **8**.

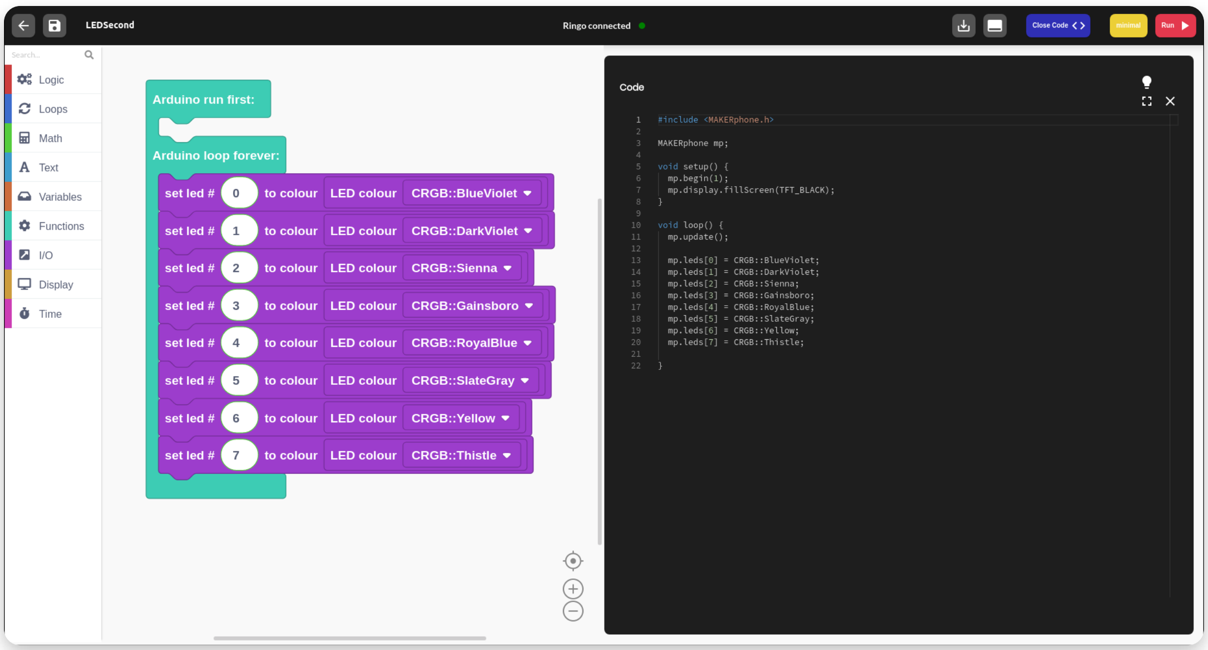
The one in the **upper right corner** of the phone, when you turn it around, is **led[0]**.

The number increases with the clockwise direction, meaning that the LED in the **upper left corner** is **led[7].**

LEDs also have a little bit different color library than the screen. It is called **CRGB** and it has many more color combinations. You can practically set your LEDs to any color you can imagine.

All these colors can be selected from the drop-down menu, so no need to keep them memorized.

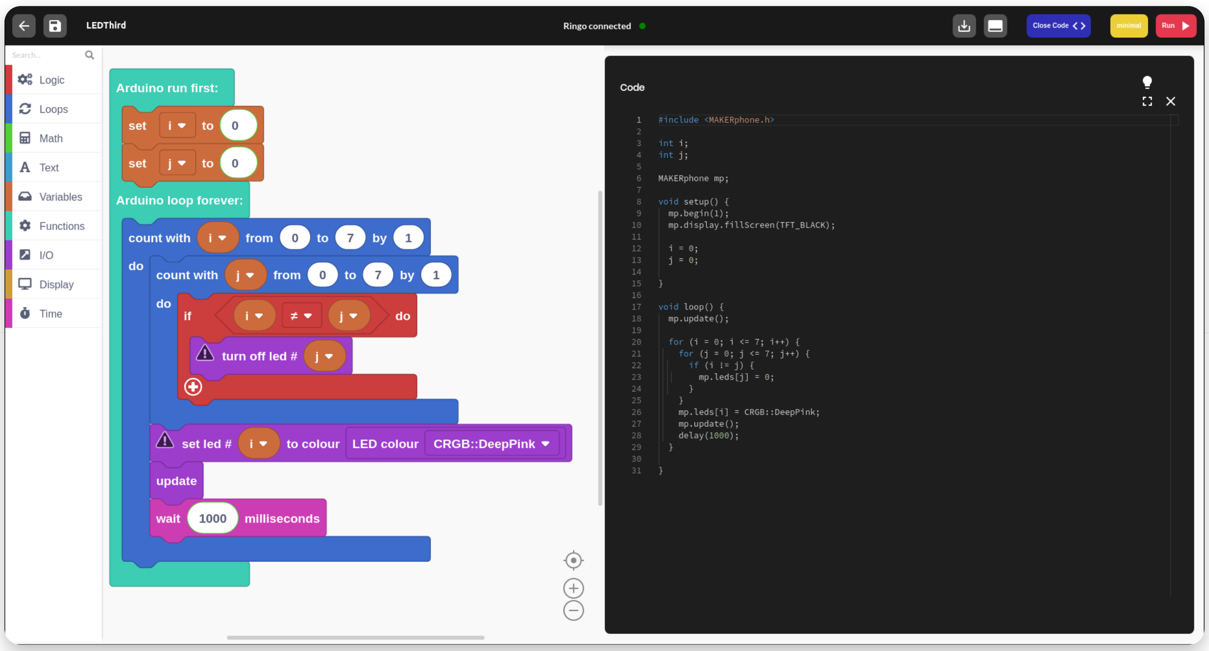
Here is an example.


LEDSecond
Ringo connected
Close Code
Run
Logic
Loops
Math
Text
Variables
Functions
I/O
Display
Time
Arduino run first:
Arduino loop forever:
set led # 0 to colour LED colour CRGB::BlueViolet
set led # 1 to colour LED colour CRGB::DarkViolet
set led # 2 to colour LED colour CRGB::Sienna
set led # 3 to colour LED colour CRGB::Gainsboro
set led # 4 to colour LED colour CRGB::RoyalBlue
set led # 5 to colour LED colour CRGB::SlateGray
set led # 6 to colour LED colour CRGB::Yellow
set led # 7 to colour LED colour CRGB::Thistle
Code

```
#include <MAKERphone.h>

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  mp.leds[0] = CRGB::BlueViolet;
  mp.leds[1] = CRGB::DarkViolet;
  mp.leds[2] = CRGB::Sienna;
  mp.leds[3] = CRGB::Gainsboro;
  mp.leds[4] = CRGB::RoyalBlue;
  mp.leds[5] = CRGB::SlateGray;
  mp.leds[6] = CRGB::Yellow;
  mp.leds[7] = CRGB::Thistle;

}
```

## LED cycle

Here is one example of how to set the LEDs in motion. It is really simple yet effective and it's not the only way you can do this.



```
#include <MAKERphone.h>

int i;
int j;

MAKERphone mp;

void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  i = 0;
  j = 0;

}

void loop() {
  mp.update();

  for (i = 0; i <= 7; i++) {
    for (j = 0; j <= 7; j++) {
      if (i != j) {
        mp.leds[j] = 0;
      }
    }
    mp.leds[i] = CRGB::DeepPink;
    mp.update();
    delay(1000);
  }

}
```

Having multiple for loops inside one another can often be the way of creating some cool programs. In this example, for loop is actually going through 64 cycles before it ends.

Here it allows you to turn off all the LEDs except that one you want to have turned on.

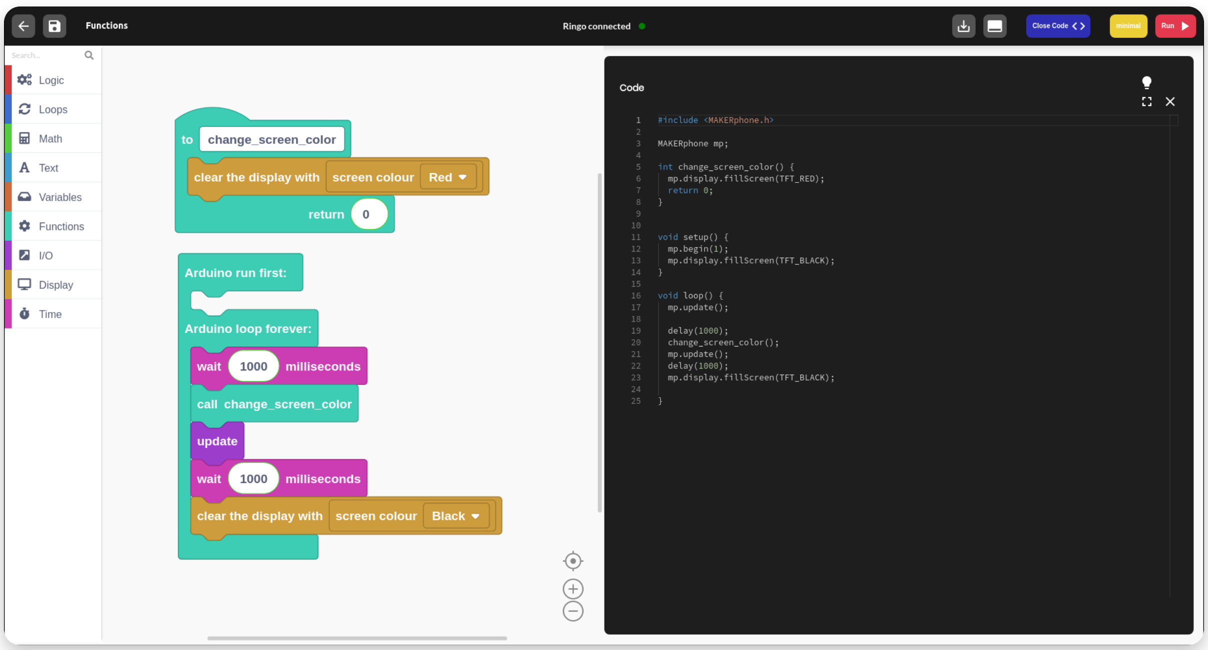
Increasing or decreasing **delay()** function will change the speed of alternating LEDs.

# Functions

Now that you've been introduced to every component it is a good time to meet some functions, which are going to be crucial for making our first app.

Functions are basically sections of the code that receive and return some values and can be used in many ways.

They are crucial if you want to keep your code clean and fast. Also, they allow for some quick upgrades without having to redo half of the code.



```
#include <MAKERphone.h>

MAKERphone mp;

int change_screen_color() {
  mp.display.fillScreen(TFT_RED);
  return 0;
}


void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);
}

void loop() {
  mp.update();

  delay(1000);
  change_screen_color();
  mp.update();
  delay(1000);
  mp.display.fillScreen(TFT_BLACK);

}
```

### How do functions work?

You can create a new function in the Functions section where you're setting its name, type, and variables.

Once called, the program will automatically **jump** to the code in the function and will run that before continuing with the rest of the program.

Whatever function returns can be caught in another variable or in a comparison.

For example, you can call a function and depending on what it returns, decide whether to enter the loop or not.

## Creating my fist app

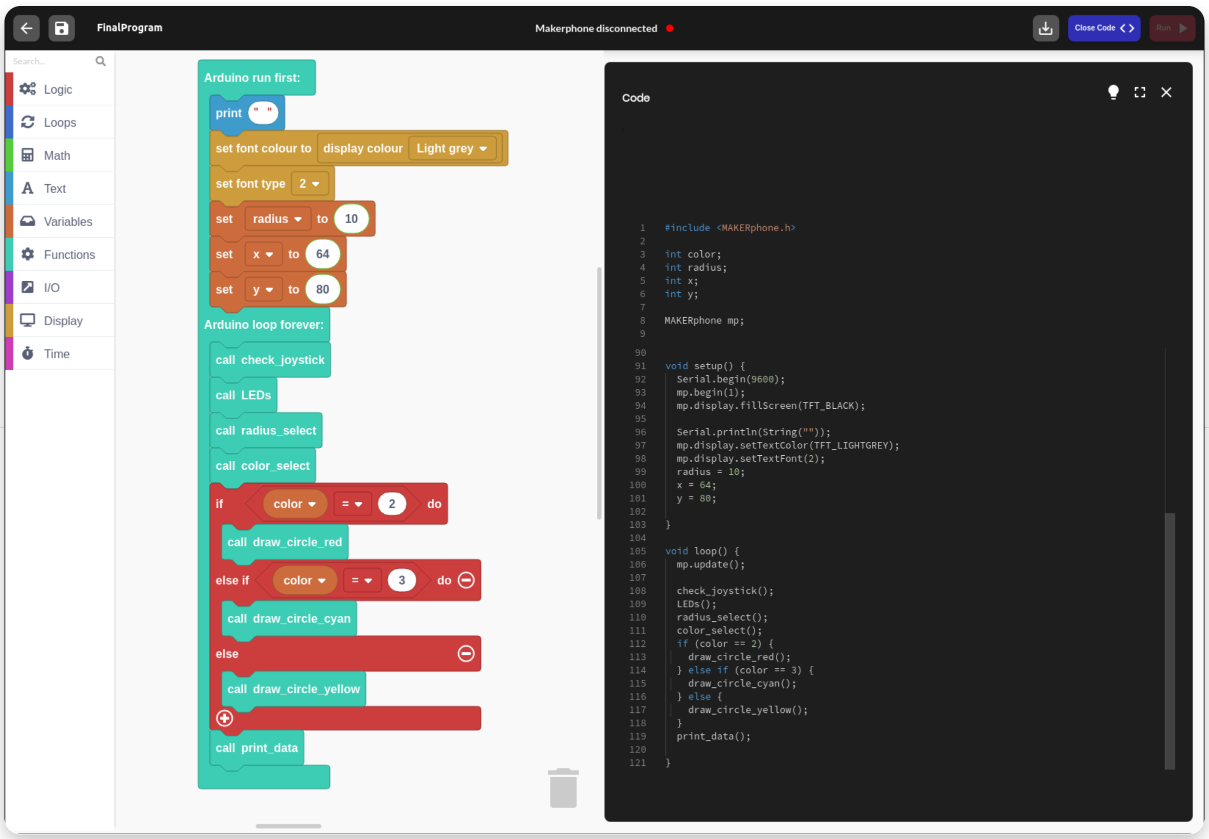
### Where do I start?

Creating a whole app is not an easy task. There are a lot of little details that need to be considered in order to make the app work as good as possible.When creating your own app, always set the goals at the beginning, so 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 on in the code development.

The app that we're going to create is one that is going to showcase pretty much every feature we've gone through so far.

It will look like a simple video game, kind of like Snake, where you'll be able to move a circle and change its properties.

## Main loop



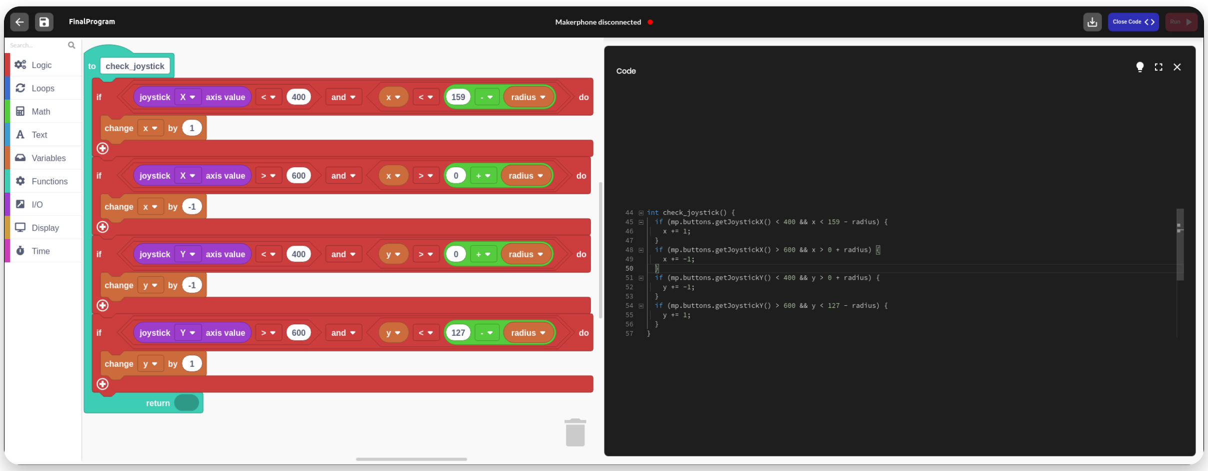
```
#include <MAKERphone.h>

int color;
int radius;
int x;
int y;

MAKERphone mp;


void setup() {
  Serial.begin(9600);
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  Serial.println(String(""));
  mp.display.setTextColor(TFT_LIGHTGREY);
  mp.display.setTextFont(2);
  radius = 10;
  x = 64;
  y = 80;

}

void loop() {
  mp.update();

  check_joystick();
  LEDs();
  radius_select();
  color_select();
  if (color == 2) {
    draw_circle_red();
  } else if (color == 3) {
    draw_circle_cyan();
  } else {
    draw_circle_yellow();
  }
  print_data();

}
```

This is what the main part of the program looks like. It doesn't say much since it has several functions that run the show in separate blocks.

We're going to go through each of those functions in order to see what are they exactly doing.

You can notice the clever use of **if-else** here - a different function will be called depending on the value of the global variable color.

## Using joystick


FinalProgram
Makerphone disconnected
Close Code
Logic
Loops
Math
Text
Variables
Functions
I/O
Display
Time
to check_joystick
if joystick X axis value < 400 and x < 159 - radius do
change x by 1
if joystick X axis value > 600 and x > 0 + radius do
change x by -1
if joystick Y axis value < 400 and y > 0 + radius do
change y by -1
if joystick Y axis value > 600 and y < 127 - radius do
change y by 1
return
Code

```
int check_joystick() {
  if (mp.buttons.getJoystickX() < 400 && x < 159 - radius) {
    x += 1;
  }
  if (mp.buttons.getJoystickX() > 600 && x > 0 + radius) {
    x += -1;
  }
  if (mp.buttons.getJoystickY() < 400 && y > 0 + radius) {
    y += -1;
  }
  if (mp.buttons.getJoystickY() > 600 && y < 127 - radius) {
    y += 1;
  }
}
```

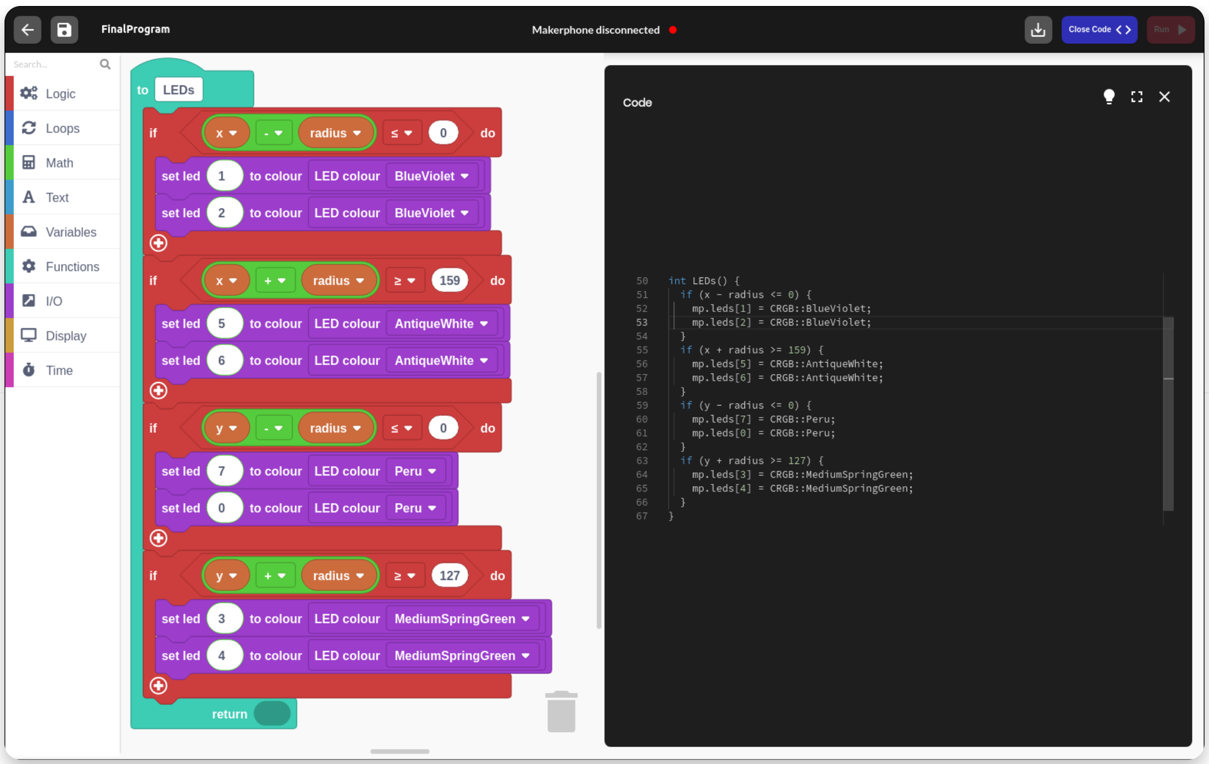
At the beginning of each loop, the program checks whether the joystick has been moved or not.

If it has been moved in a certain direction, it will change the location variables of our character.

Placing multiple ifs instead of **if-else** we can get the **diagonal** movement, since one if doesn't exclude another.

**The sensitivity of joystick can be changed by changing the values of comparison to the X and Y values.**

## LEDs



```
int LEDs() {
  if (x - radius <= 0) {
    mp.leds[1] = CRGB::BlueViolet;
    mp.leds[2] = CRGB::BlueViolet;
  }
  if (x + radius >= 159) {
    mp.leds[5] = CRGB::AntiqueWhite;
    mp.leds[6] = CRGB::AntiqueWhite;
  }
  if (y - radius <= 0) {
    mp.leds[7] = CRGB::Peru;
    mp.leds[0] = CRGB::Peru;
  }
  if (y + radius >= 127) {
    mp.leds[3] = CRGB::MediumSpringGreen;
    mp.leds[4] = CRGB::MediumSpringGreen;
  }
}
```

Here we're using the mighty power of our LEDs to create some really cool wall effects.

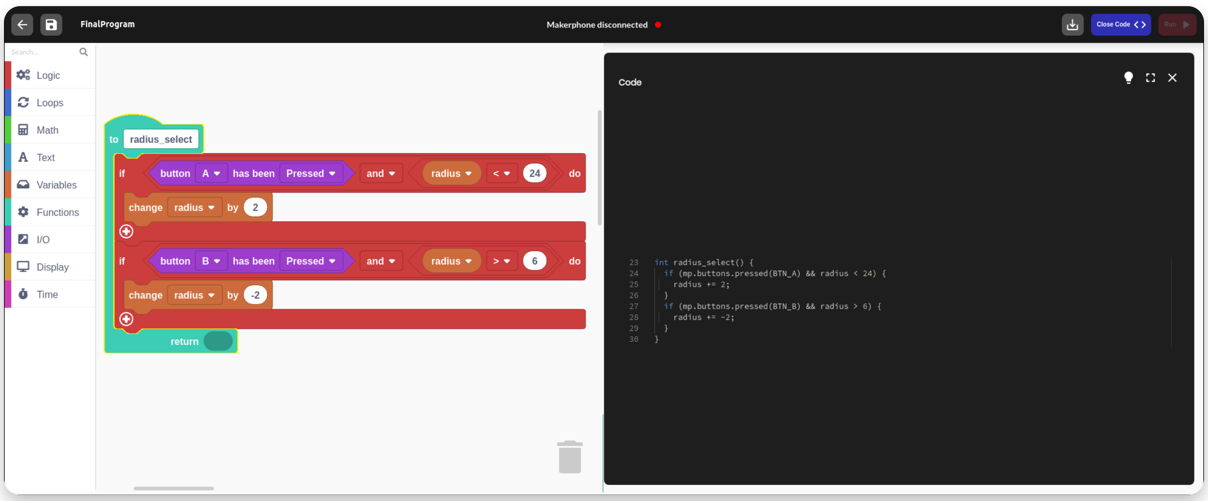
Whenever the ball touches the wall, the LEDs at the back of the phone will light up and thus create an indication of the collision.

Notice that since both **X and Y** values are actually the center of the ball, we must keep in mind the radius size of our ball so that we know when the collision actually appears.

Since the length of the screen is **160** pixels and is taking up the values between **0 and 159**, we are comparing it to the values of **0 and 159**.

Same works with the height, where the most important values are **0 and 127**.

## Object size



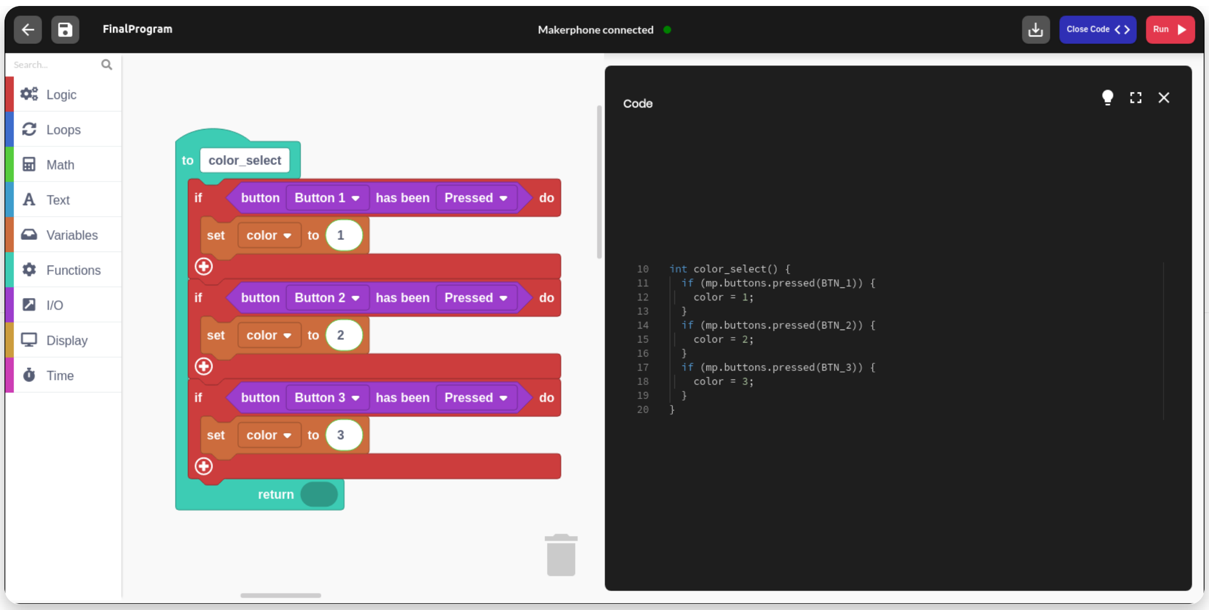
```
int radius_select() {
  if (mp.buttons.pressed(BTN_A) && radius < 24) {
    radius += 2;
  }
  if (mp.buttons.pressed(BTN_B) && radius > 6) {
    radius += -2;
  }
}
```

Here we can choose our radius size by pressing buttons A and B.

**Button A will increase its size by two and button B will decrease it.**

There are also checks at specific values so we can stop our circle from getting too big or too small.

## Colors

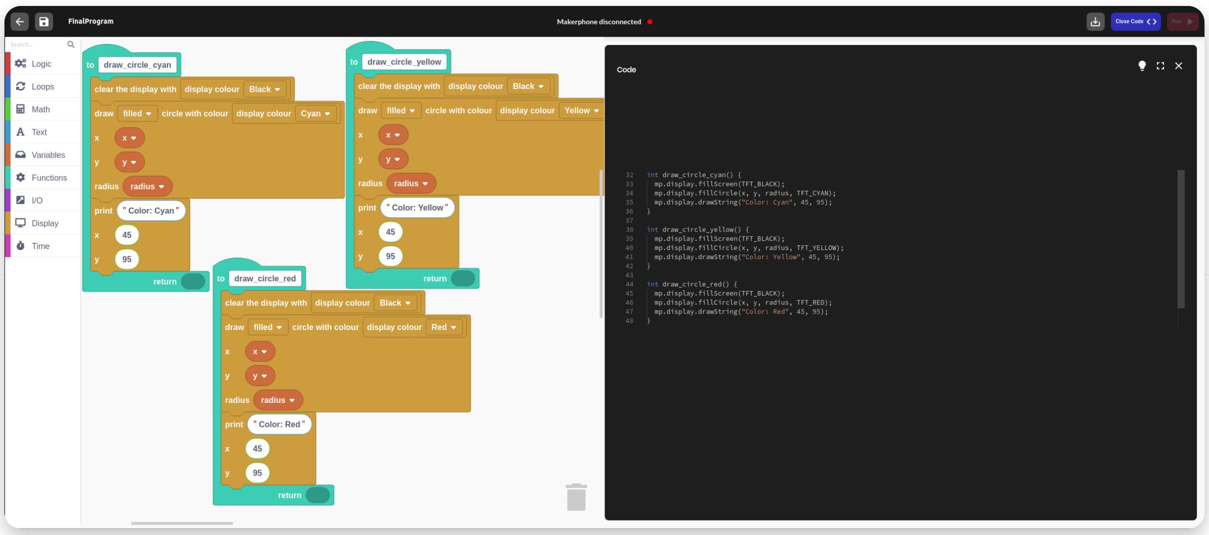


```
int color_select() {
  if (mp.buttons.pressed(BTN_1)) {
    color = 1;
  }
  if (mp.buttons.pressed(BTN_2)) {
    color = 2;
  }
  if (mp.buttons.pressed(BTN_3)) {
    color = 3;
  }
}
```

We've created the variable color so we can easier change circle colors.

This is probably the easiest way to do it, but it can also be done in other ways.

Pressing on of buttons **1, 2 or 3,** this variable changes. We will use it in another function.



```
int draw_circle_cyan() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_CYAN);
  mp.display.drawString("Color: Cyan", 45, 95);
}

int draw_circle_yellow() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_YELLOW);
  mp.display.drawString("Color: Yellow", 45, 95);
}

int draw_circle_red() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_RED);
  mp.display.drawString("Color: Red", 45, 95);
}
```

This is the part of the code where we come across the if-else part.

Depending on the color variable that we've previously changed, we are entering one of the three functions.

They all work in the same way but have a tiny little difference that makes them special.

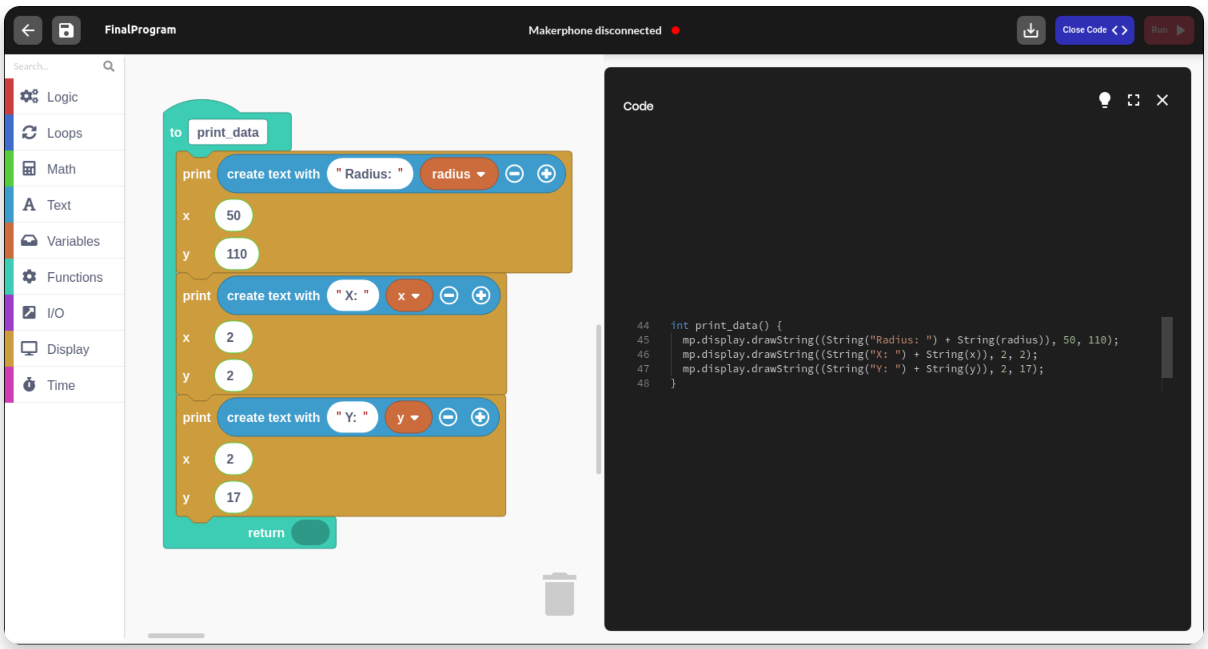
Depending on which function is called, the color of the circle will be different.

Clearing the screen with the black color allows the circle to move swiftly and look like an animation.

Also, the function prints out the color of the circle at the bottom of the screen.

We've already set the font type and font size at the beginning of the program so there is no need in doing that now.

## Print data



```
int print_data() {
  mp.display.drawString((String("Radius: ") + String(radius)), 50, 110);
  mp.display.drawString((String("X: ") + String(x)), 2, 2);
  mp.display.drawString((String("Y: ") + String(y)), 2, 17);
}
```

And finally, we're printing out some additional data on the screen.

At the bottom of the screen, right below the color code, there will be the current radius size shown.

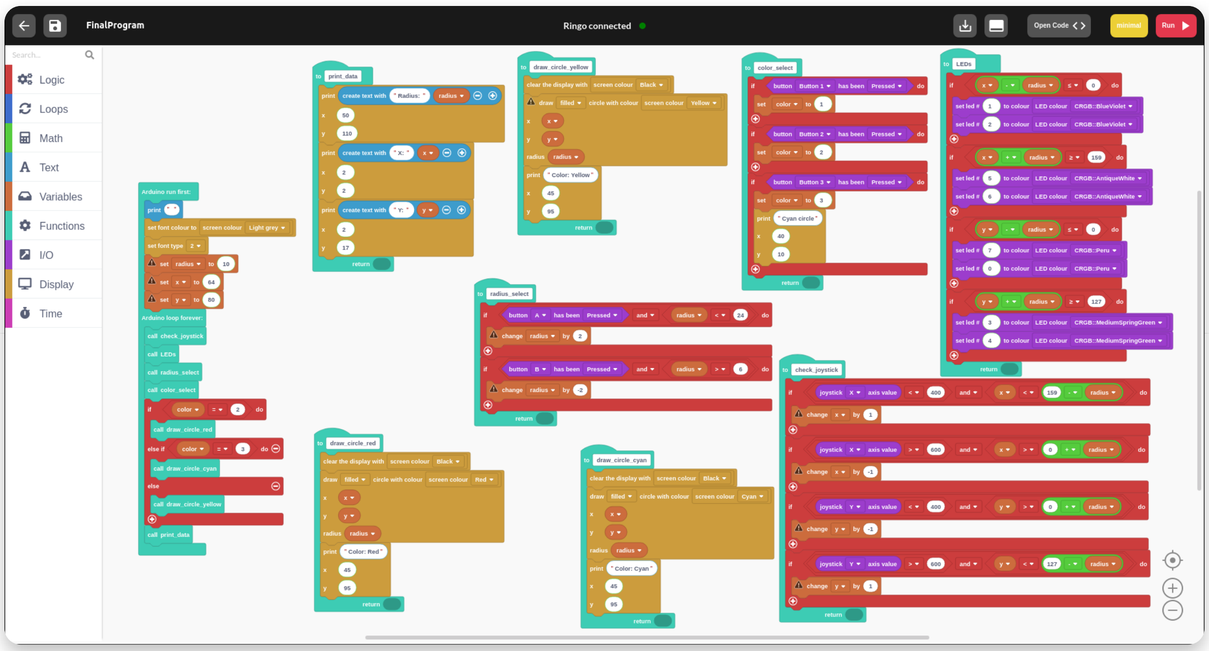
Also, the **coordinates** of the circle center will be shown in the **upper left corner**.

This will perfectly track the circle along with its size.

Since this function was called after the draw_circle function, this text will always be visible even when the circle is at the same location on the screen.

If you want it to be otherwise, just call the draw_circle function after this one.

**The next picture represents the entire program.**

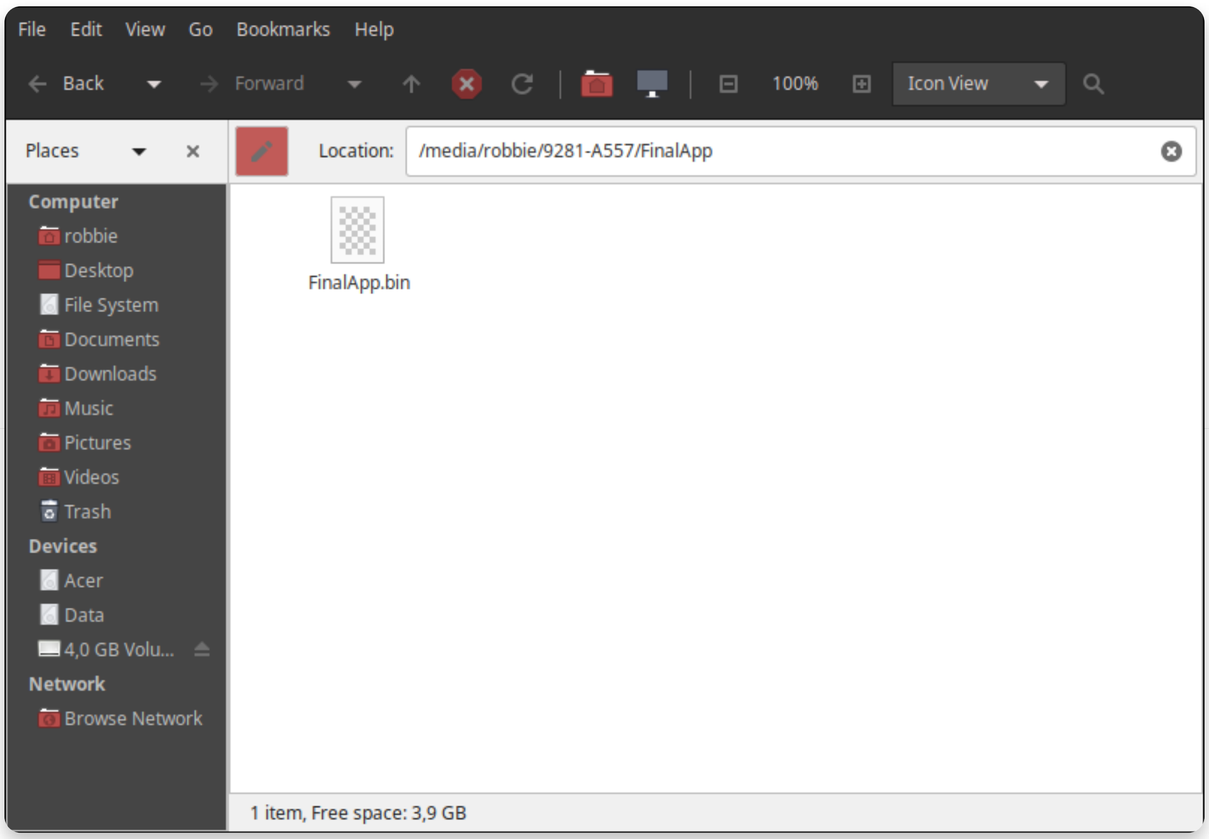


ARDUINO

```
#include <MAKERphone.h>

int x;
int y;
int radius;
int color;

MAKERphone mp;

int print_data() {
  mp.display.drawString((String("Radius: ") + String(radius)), 50, 110);
  mp.display.drawString((String("X: ") + String(x)), 2, 2);
  mp.display.drawString((String("Y: ") + String(y)), 2, 17);
}

int draw_circle_yellow() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_YELLOW);
  mp.display.drawString("Color: Yellow", 45, 95);
}

int color_select() {
  if (mp.buttons.pressed(BTN_1)) {
    color = 1;
  }
  if (mp.buttons.pressed(BTN_2)) {
    color = 2;
  }
  if (mp.buttons.pressed(BTN_3)) {
    color = 3;
    mp.display.drawString("Cyan circle", 40, 10);
  }
}

int LEDs() {
  if (x - radius <= 0) {
    mp.leds[1] = CRGB::BlueViolet;
    mp.leds[2] = CRGB::BlueViolet;
  }
  if (x + radius >= 159) {
    mp.leds[5] = CRGB::AntiqueWhite;
    mp.leds[6] = CRGB::AntiqueWhite;
  }
  if (y - radius <= 0) {
    mp.leds[7] = CRGB::Peru;
    mp.leds[0] = CRGB::Peru;
  }
  if (y + radius >= 127) {
    mp.leds[3] = CRGB::MediumSpringGreen;
    mp.leds[4] = CRGB::MediumSpringGreen;
  }
}

int radius_select() {
  if (mp.buttons.pressed(BTN_A) && radius < 24) {
    radius += 2;
  }
  if (mp.buttons.pressed(BTN_B) && radius > 6) {
    radius += -2;
  }
}

int check_joystick() {
  if (mp.buttons.getJoystickX() < 400 && x < 159 - radius) {
    x += 1;
  }
  if (mp.buttons.getJoystickX() > 600 && x > 0 + radius) {
    x += -1;
  }
  if (mp.buttons.getJoystickY() < 400 && y > 0 + radius) {
    y += -1;
  }
  if (mp.buttons.getJoystickY() > 600 && y < 127 - radius) {
    y += 1;
  }
}

int draw_circle_red() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_RED);
  mp.display.drawString("Color: Red", 45, 95);
```

```
}

int draw_circle_cyan() {
  mp.display.fillScreen(TFT_BLACK);
  mp.display.fillCircle(x, y, radius, TFT_CYAN);
  mp.display.drawString("Color: Cyan", 45, 95);
}


void setup() {
  mp.begin(1);
  mp.display.fillScreen(TFT_BLACK);

  mp.display.setTextColor(TFT_LIGHTGREY);
  mp.display.setTextFont(2);
  radius = 10;
  x = 64;
  y = 80;

}

void loop() {
  mp.update();

  check_joystick();
  LEDs();
  radius_select();
  color_select();
  if (color == 2) {
    draw_circle_red();
  } else if (color == 3) {
    draw_circle_cyan();
  } else {
    draw_circle_yellow();
  }
  print_data();

}
```

## Conclusion



Location: /media/robbie/9281-A557/FinalApp

FinalApp.bin

1 item, Free space: 3,9 GB

When we finally write the entire thing, we can export it as an app and voila!

**In chapter 3 there is a detailed tutorial on how to do this, so go check it out!**

There is a new app on your main menu ready to go!

## Uploading apps and games

### Direct Upload

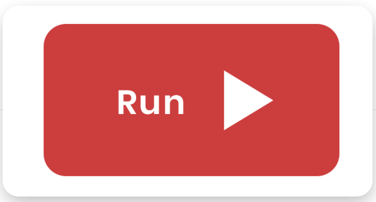
Now that you know everything about what CircuitBlocks can do and how it works, let's head to some programming.

There are two main ways you can upload your creations to Ringo – **upload it directly to the phone and running it as .bin.**

Let's touch on these two ways so you can see what works best for you.

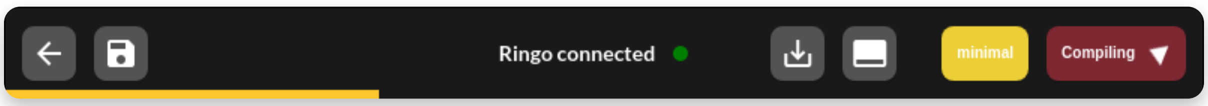
Uploading the program directly to the Ringo's memory is simple and effective!

It is done by just **pressing the "Run" button in the top right corner of the screen.**



The program will compile and upload to your phone, which will take about a couple of minutes.

There is also a yellow bar right below the header that indicates the progress of compiling and uploading.



Yellow bar represents compilation and upload progress

**"Run" command will not only upload everything's that's on the board to the phone but will also erase the default firmware, so you won't be able to go back to using the phone unless you restore the default firmware!**

However, that is really easy, since you just go back to the main menu of CircuitBlocks and press the 'Restore Ringo Firmware' button.

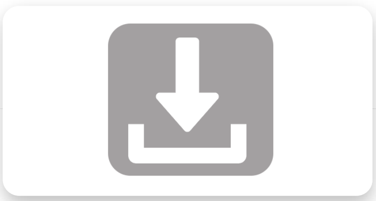
This method should be used when testing the program and just want a quick check of functions or when you only want this app to be available on your phone!

**ex. You made a video game and you want to organize a contest with your friends to check out who gets the highest score – this way you can make Ringo game only device!**

### Making a .bin file

When you make sure your program works just like you want it to, **it's time to make it part of the Ringo OS.**

By doing this method, you're basically putting it on the main menu screen so you can use it whenever you want to!



Though it won't load as fast as the other regular apps, it works more like the games you have on your menu. The games first load for about 10-20 seconds and then are loaded to your phone. When you want to return, just press the 'Home' button and select 'Home' icon to go straight back to the lockscreen.

**Pressing the 'Export to .bin' button will open a new window so you can choose the exact location you want to export the .bin.**

**The whole process takes about 30 seconds.**

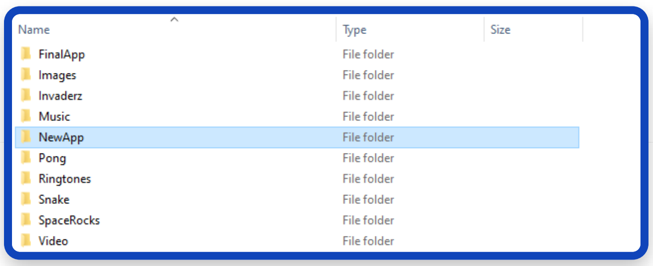
**Now we need to transfer it to Ringo.**

For this step, you're going to use the SD card, which is located at the bottom of the brain board.

Push it in and a spring mechanism will pop it out.

Now take a little USB-like device that you've received in your Ringo box – it's SD to USB adapter and it allows you to work with your SD card even if you don't have that exact port on your computer.

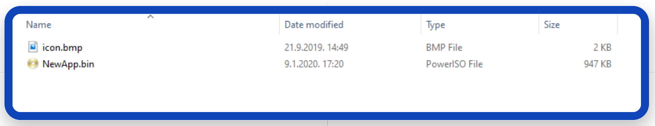
When you plug it in there should be some default folders on your SD card.



| Name | Type | Size |
|---|---|---|
| FinalApp | File folder | |
| Images | File folder | |
| Invaderz | File folder | |
| Music | File folder | |
| NewApp | File folder | |
| Pong | File folder | |
| Ringtones | File folder | |
| Snake | File folder | |
| SpaceRocks | File folder | |
| Video | File folder | |

Create a new folder and name it however you want (in this example it's named 'NewApp'). Now copy the .bin file inside.

You can see that in another game folder there is also a .bmp file named 'icon'. It is, as you might guess, an icon that will be shown on the Ringo main menu.



| Name | Date modified | Type | Size |
|---|---|---|---|
| icon.bmp | 21.9.2019. 14:49 | BMP File | 2 KB |
| NewApp.bin | 9.1.2020. 17:20 | PowerISO File | 947 KB |

**NOTE: Make sure to name the .bin file the same as the folder.Ex. If the folder is named 'NewApp', the game file should be named 'NewApp.bin'.**

**If you wish to create your own app icon, make sure it's a 24×26 dimension and a 24-bit Bitmap (.bmp).**

**Name it 'icon' and put it in the same folder.**

If you don't do that, the app will still be visible in the main menu but will contain no icon.

Now that you know how to upload your apps and games to the phone, it's time to start creating!