

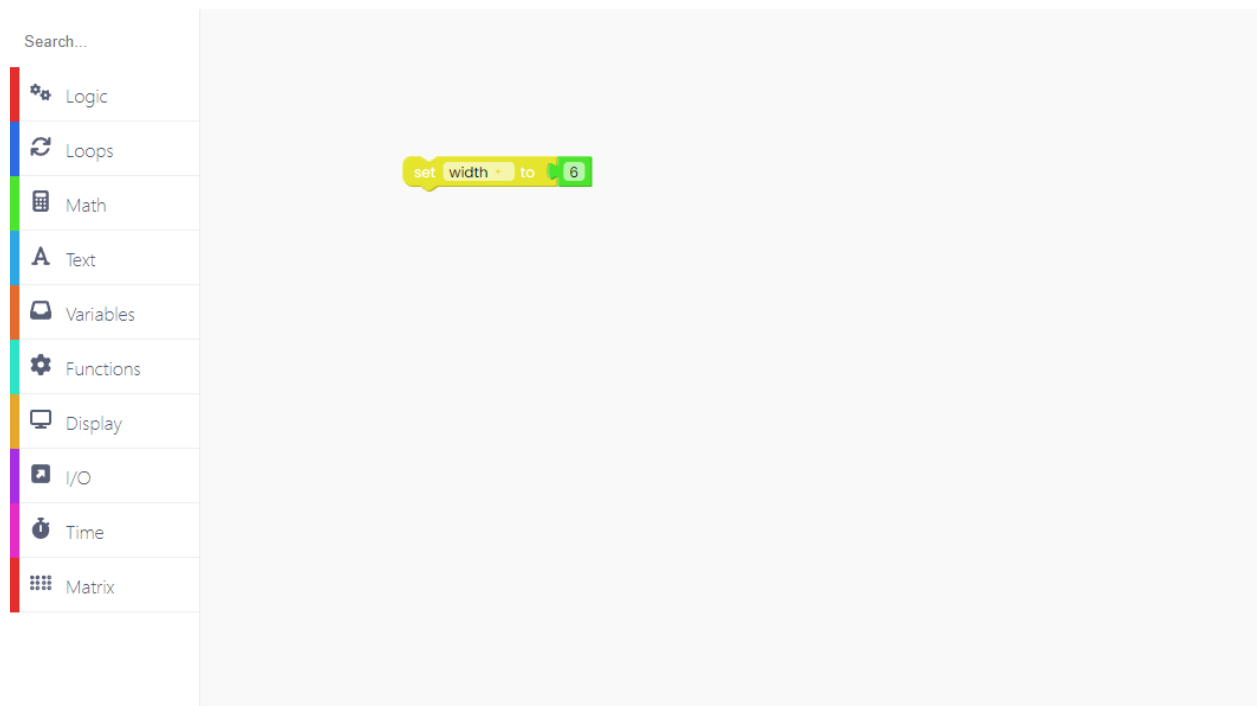
# Coding for beginners - how to code your Jay-D Encoders

As you can see, there are many LEDs on your Jay-D.

Let's have some fun with them.

The goal of this code will be to turn the encoders left and right to light up or switch off the LEDs in a row.

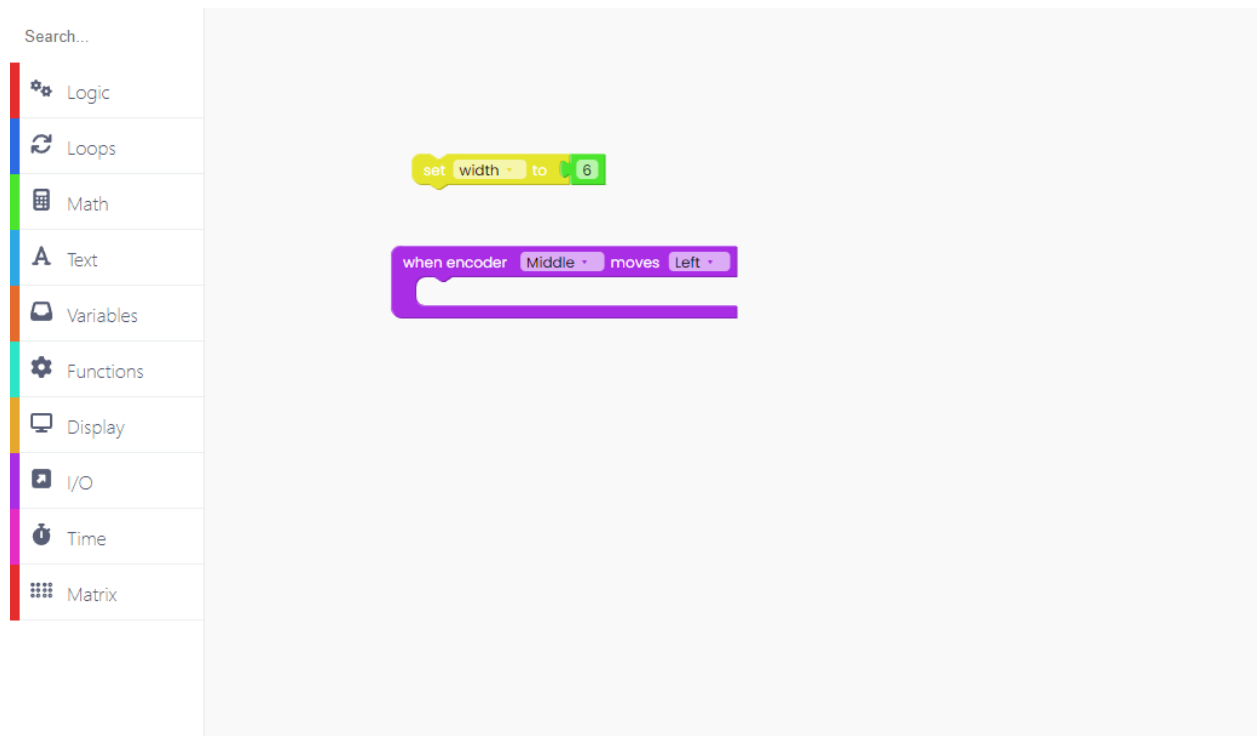
To begin, we must create the variable "width" and set it to 6.



Because we are coding the LEDs above the center slider (mid matrix), a value of 6 indicates that the variable is set to the middle.

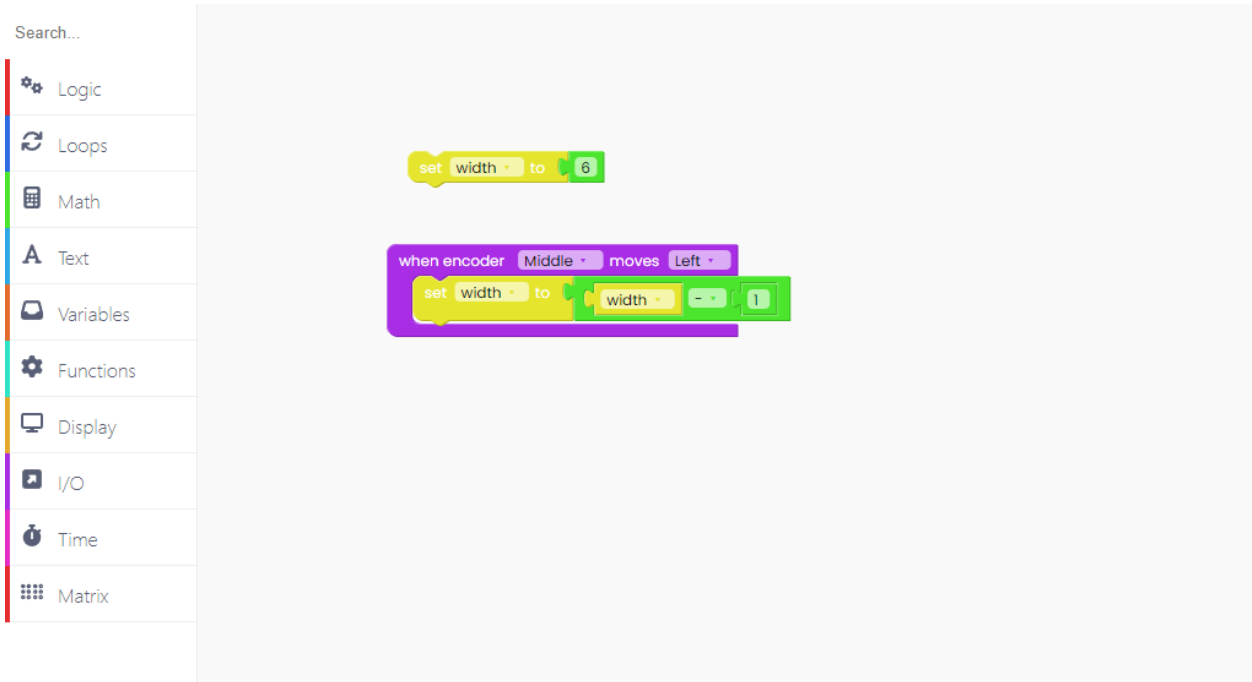
Now, let's code what happens when we move the encoders to the left and right.

Find this I/O block:

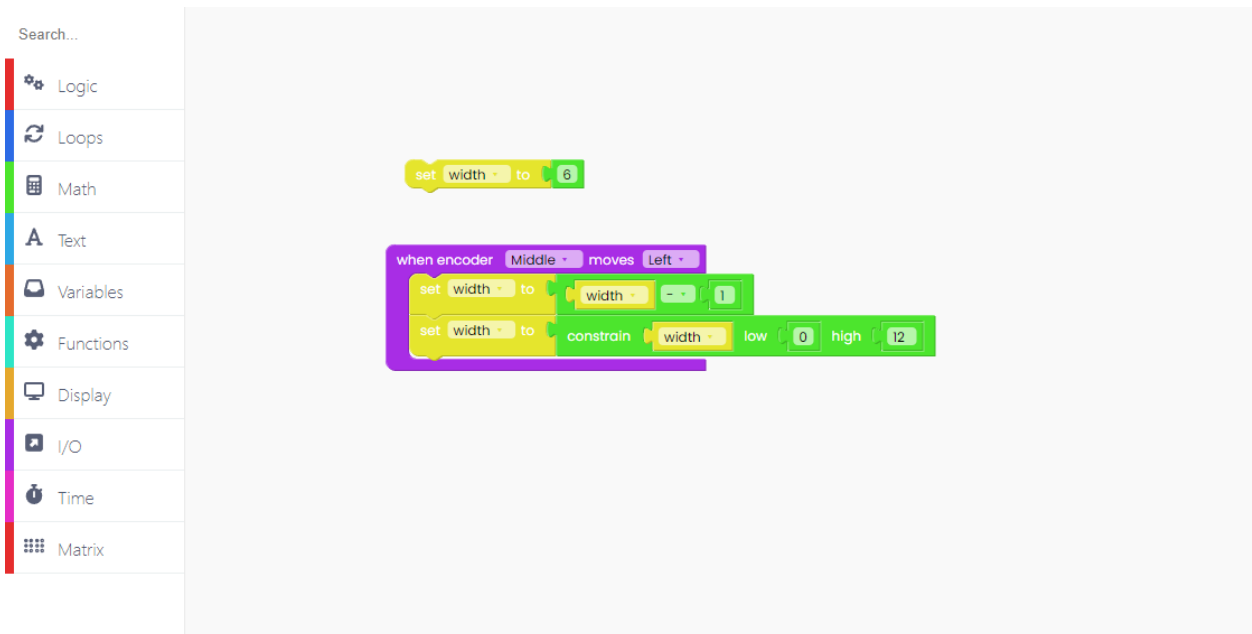


We want the LEDs to turn off one by one when we move the encoder to the left.

We'll need to subtract 1 from the variable "width" to accomplish this.

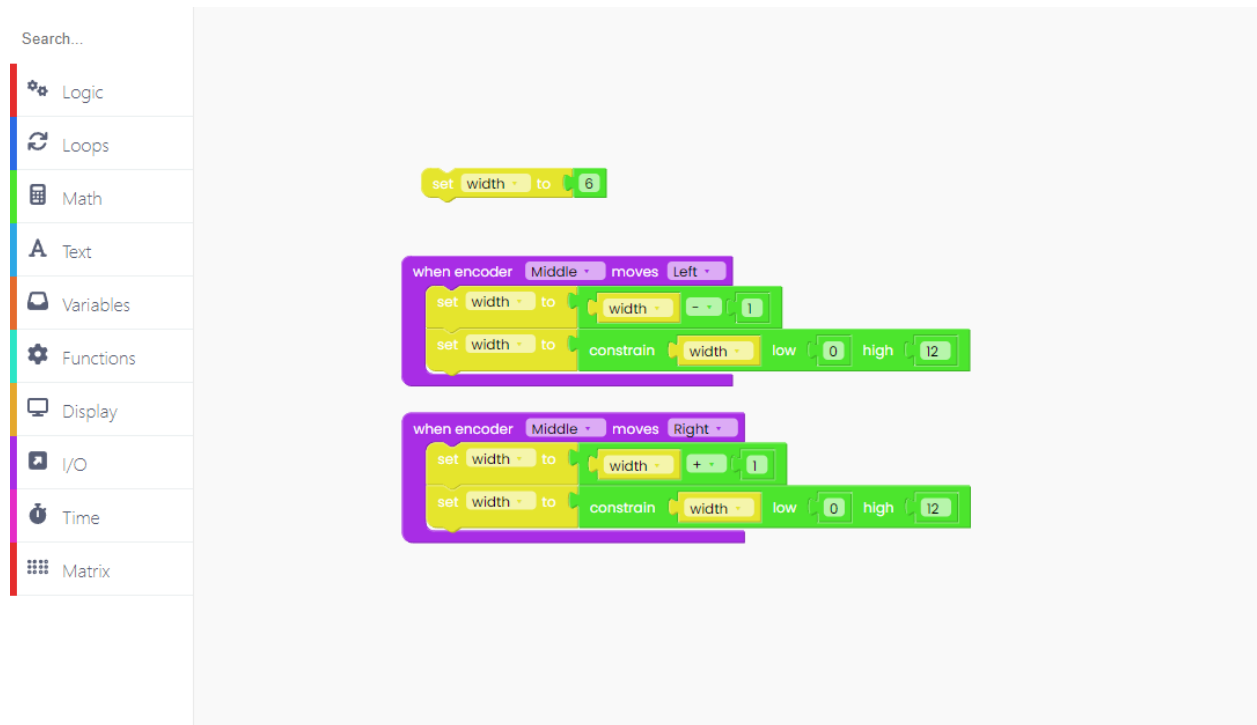


We must also ensure that the value range is between 0 and 12, as this is the number of available LEDs.



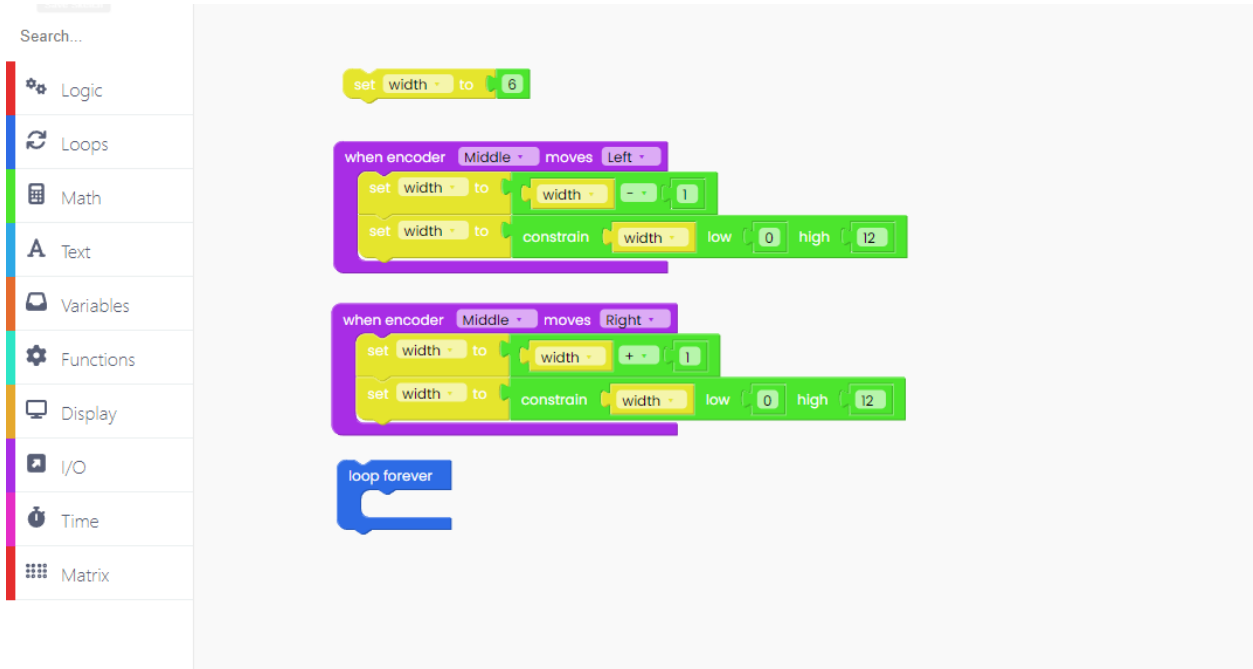
We can duplicate this entire block and change what will happen if we turn the encoder to the right.

Make sure you're adding 1 right now!



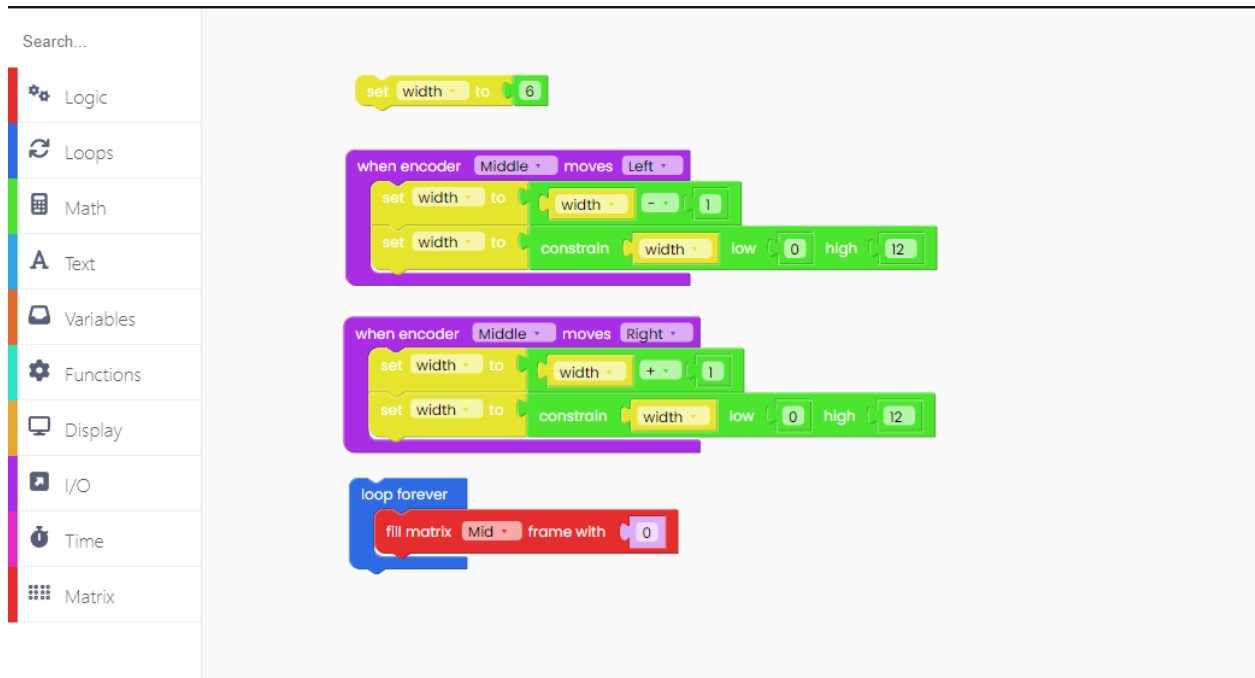
We'd like to draw anything on the LED matrix while we're messing around with the encoders.

Because that will be displayed all of the time, we must first locate the "loop forever" block from the "Loops" block part.



The first thing we want to do is turn off all of the LEDs.

This will be accomplished by locating the "fill matrix with" block in the "Matrix" block section.



Make sure to put the Mid matrix since that's the one we are working on right now.

It's time to draw a rectangle.

That block can also be found in the "Matrix" block section.

Make sure you use the correct matrix in the code.

Set the width value to the variable's value and the height value to 2.

Remember to switch on the LEDs by setting the intensity to 100.

Search...

Logic

Loops

Math

Text

Variables

Functions

Display

I/O

Time

Matrix

set width to 6

when encoder Middle moves Left

set width to width - 1

set width to constrain width low 0 high 12

when encoder Middle moves Right

set width to width + 1

set width to constrain width low 0 high 12

loop forever

fill matrix Mid frame with 0

in matrix Mid draw filled rectangle

width: width

height: 2

x: 0

y: 0

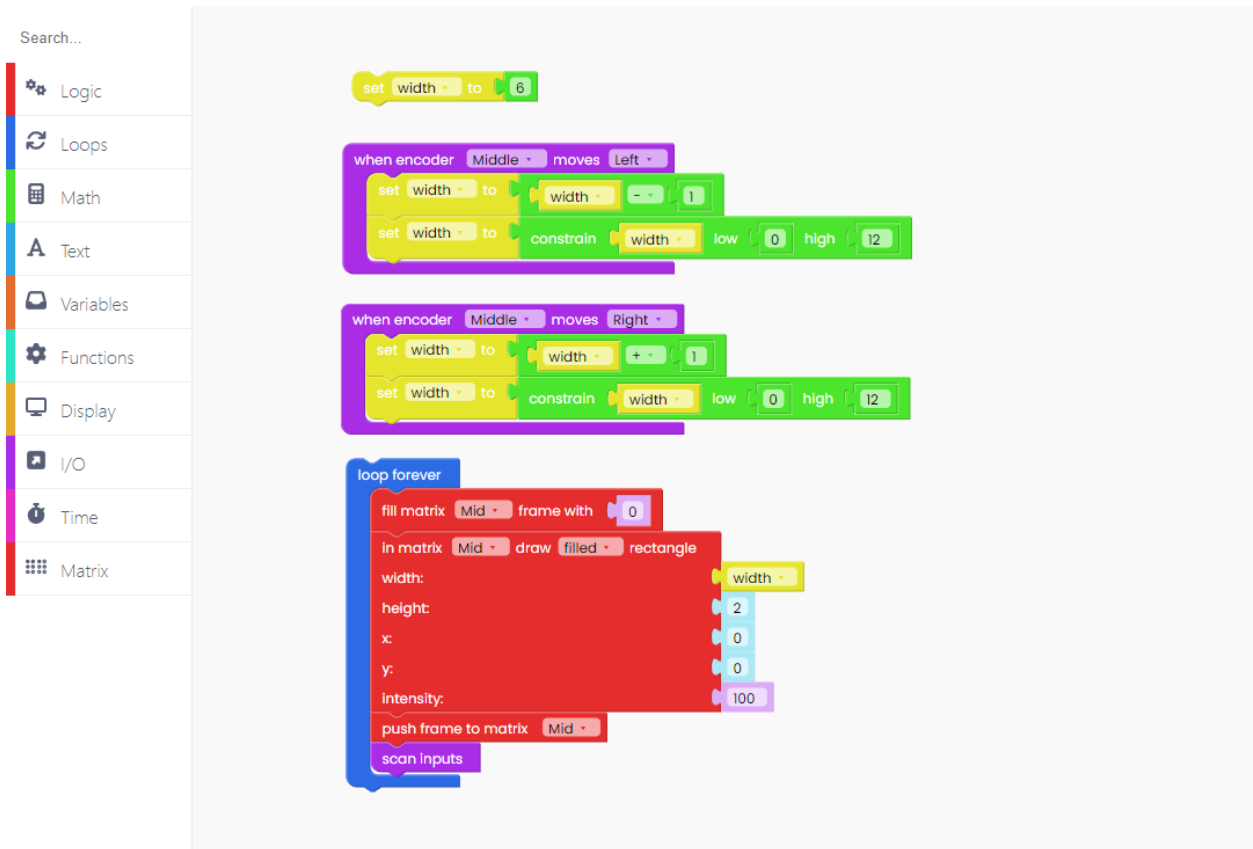
intensity: 100

To ensure that anything you create on the matrix appears on the screen, you must include the "push frame" block.



Another thing to remember is to always put the "scan inputs" block inside the "loop forever" block if you're coding anything with the inputs (encoders, pushbuttons, sliders, etc.).





Click on the Run button and start playing!

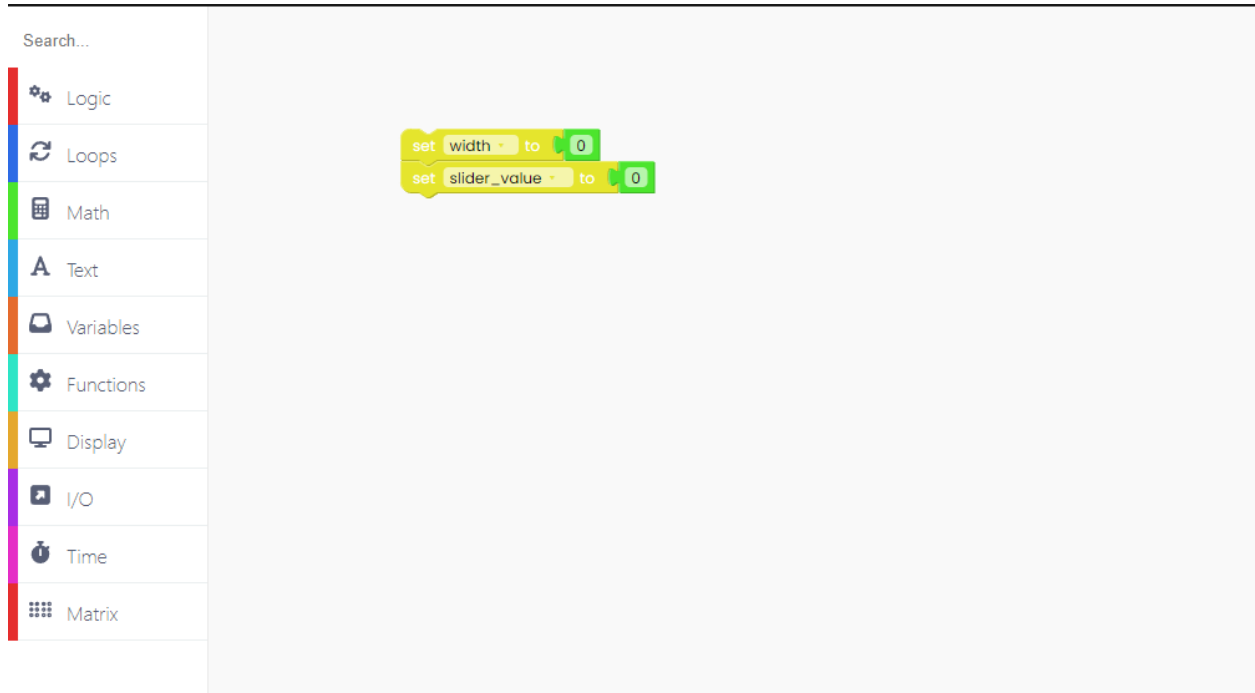
## Sliders

Sliders are another type of input that we can experiment with.

With this code, we want you to expand or narrow the rectangle you'll create on the matrix.

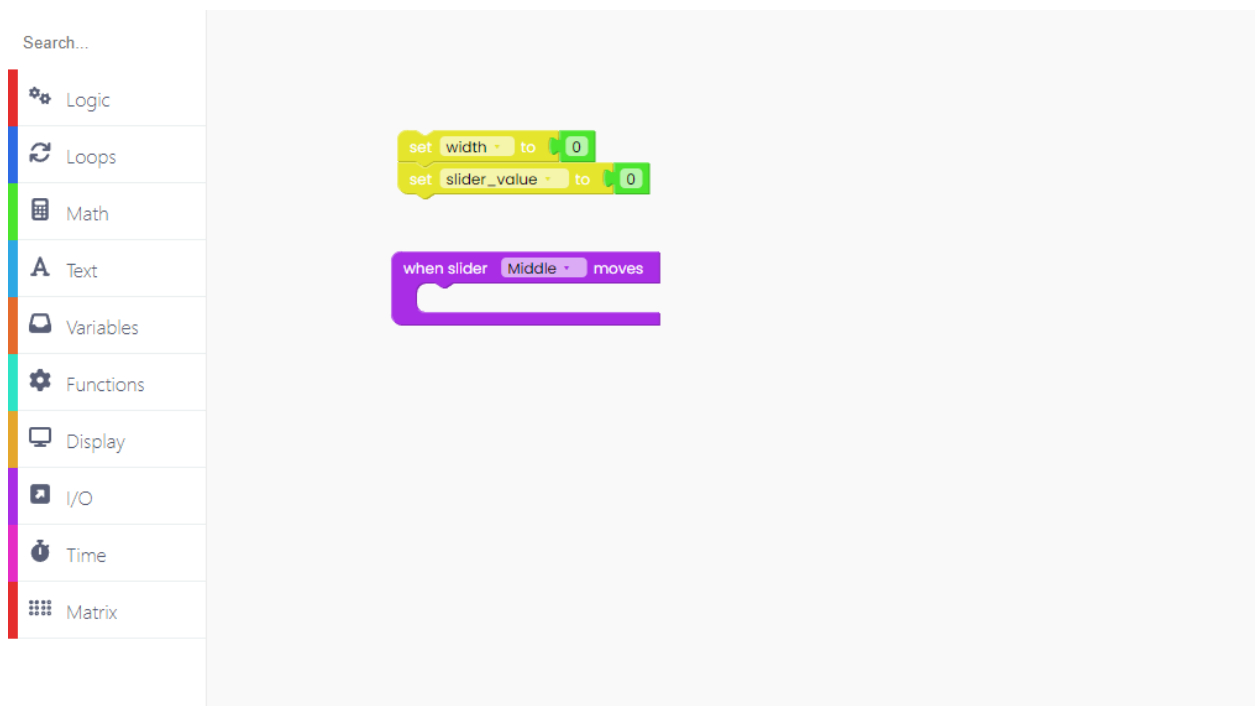
First, create two variables called "width" and "slider\_value."

Set both values to 0.



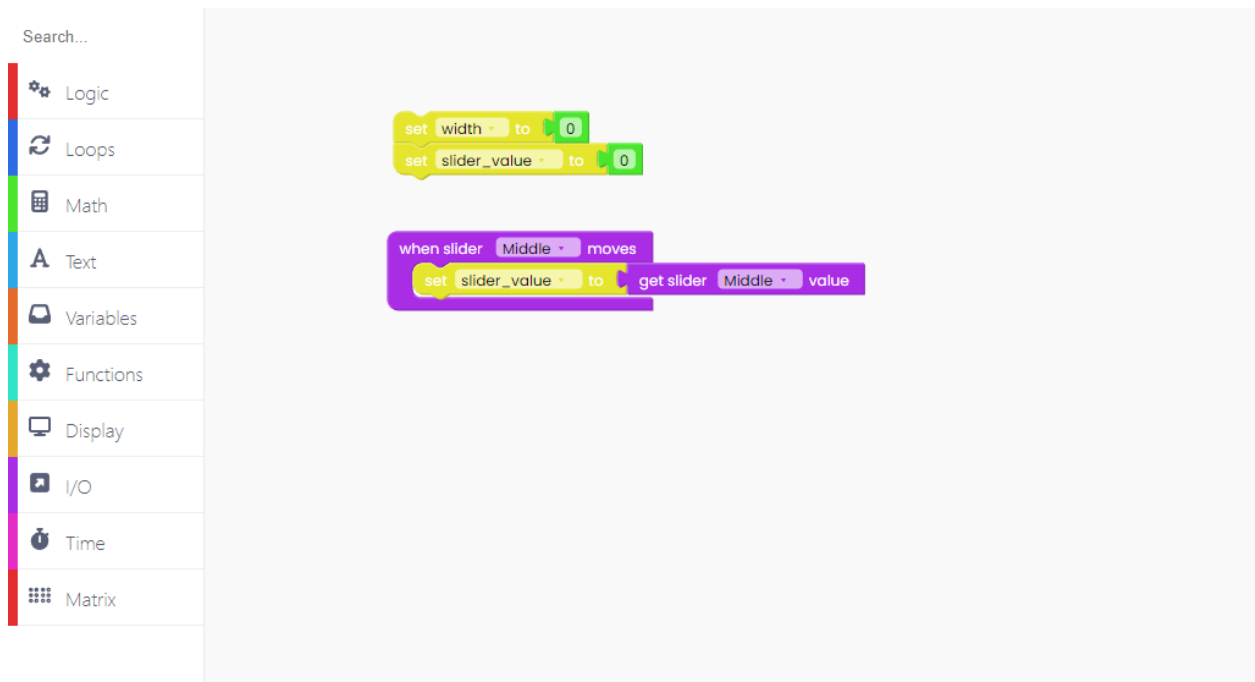
Now, let's code the changes we'll make with the slider.

For that, we'll need the I/O block, inside which we'll insert the code for the slider changes.



We want to save the slider's current position (which can range between 0 and 235) every

time we move it.

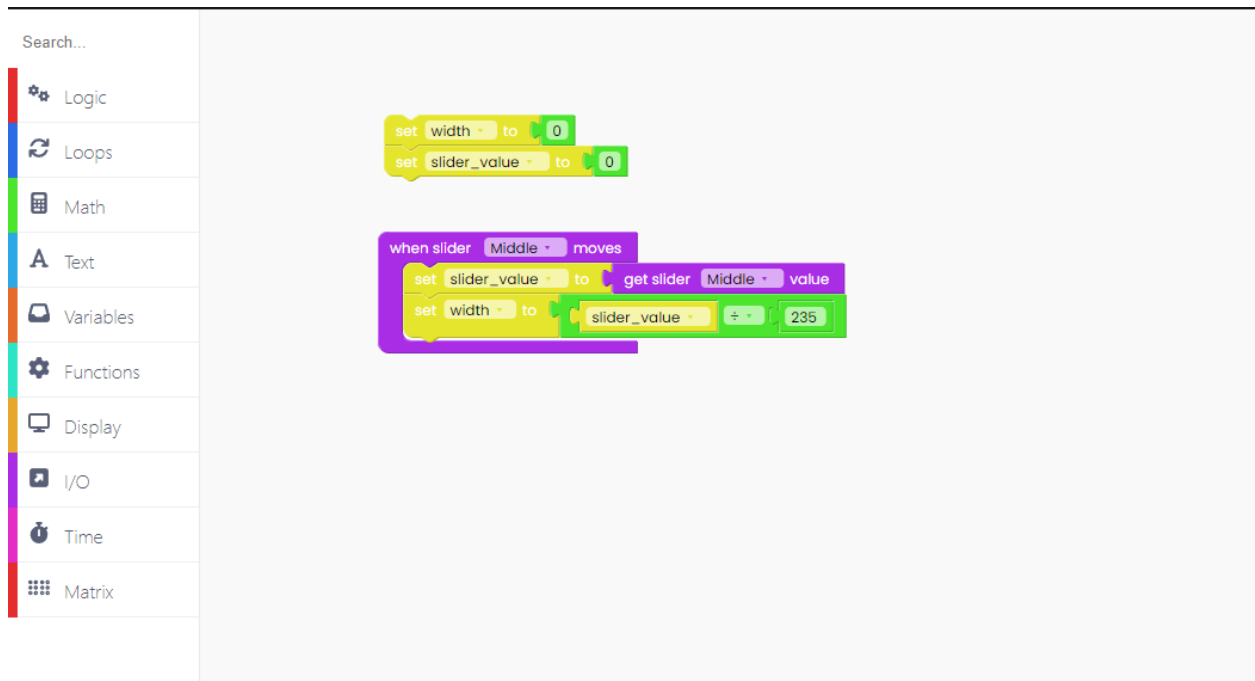


The image shows a Scratch code editor with a sidebar on the left containing categories: Logic, Loops, Math, Text, Variables, Functions, Display, I/O, Time, and Matrix. The main workspace contains the following code blocks:

```
set width to 0
set slider_value to 0

when slider Middle moves
  set slider_value to get slider Middle value
```

Then, in the form of ratio, we wish to store that value in the "width" variable.

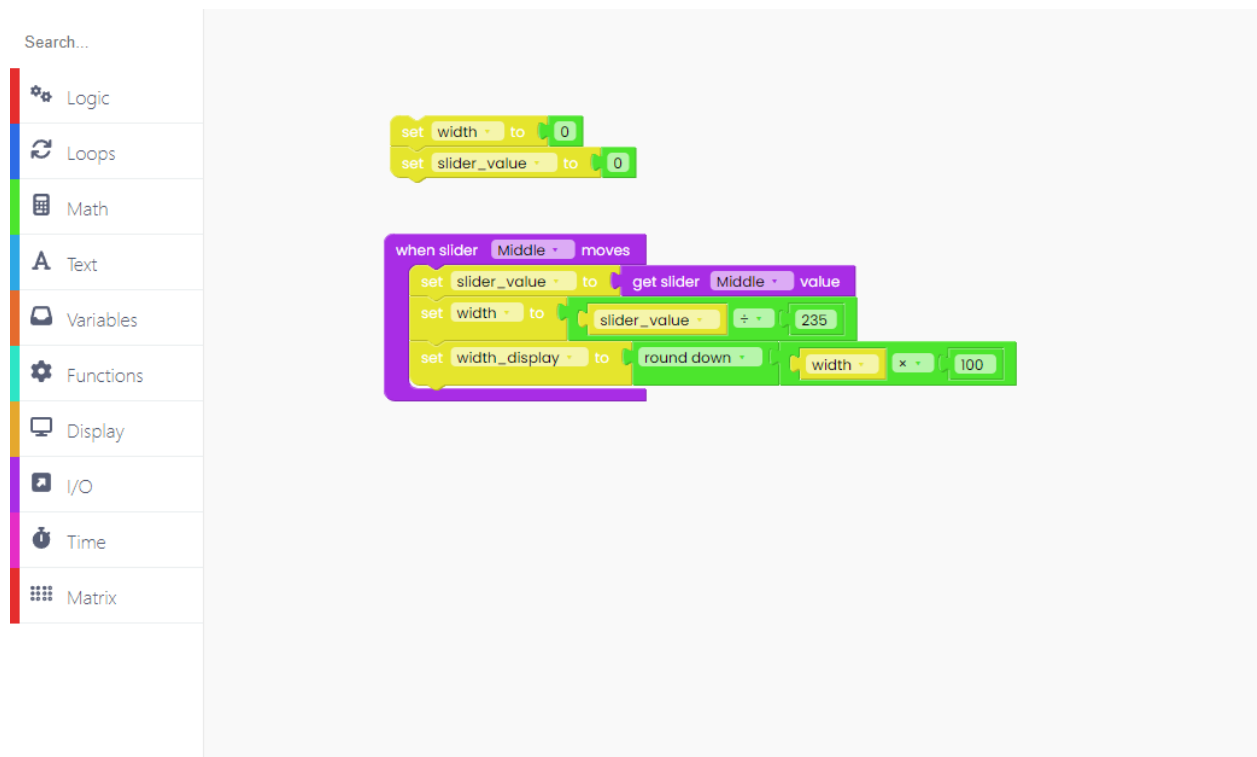


The image shows the same Scratch code editor as above, but with an additional block added to the 'when slider Middle moves' event:

```
set width to 0
set slider_value to 0

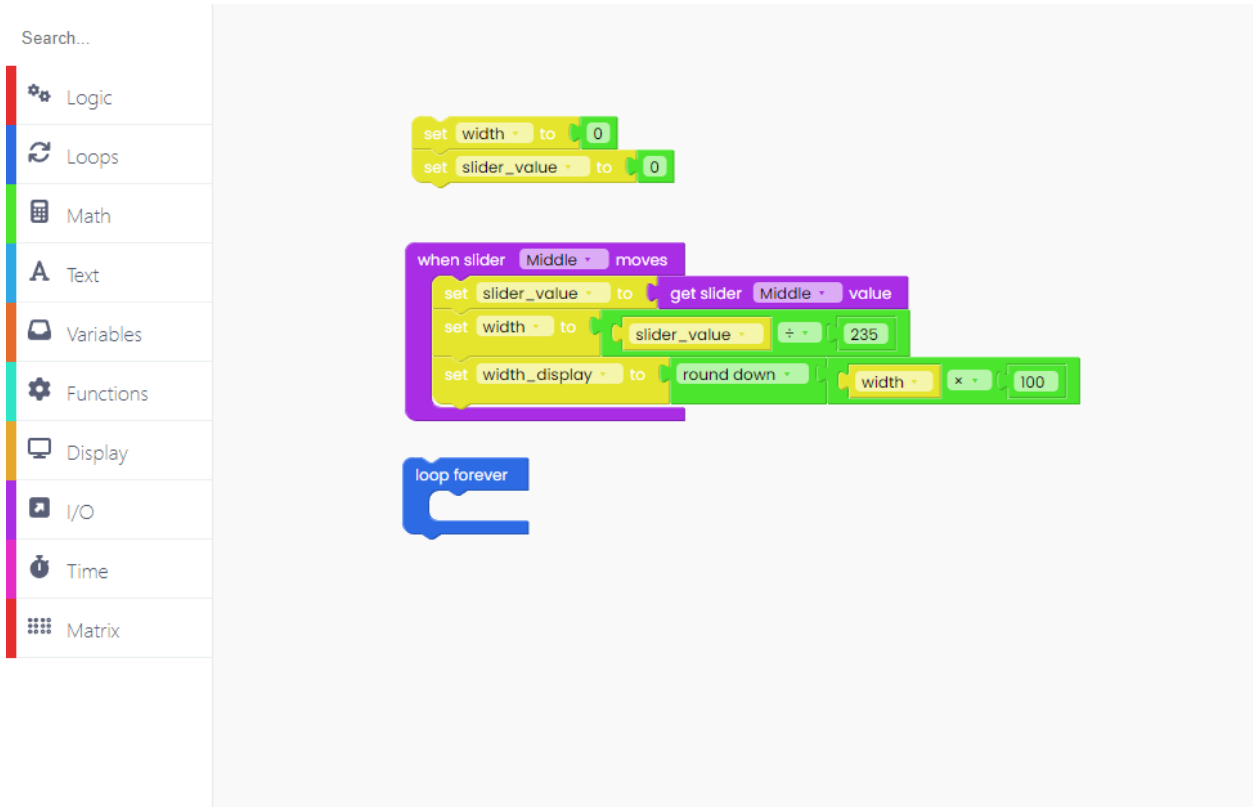
when slider Middle moves
  set slider_value to get slider Middle value
  set width to slider_value ÷ 235
```

Make a new variable called "width\_display". Store the above value in the variable "width\_display" as a percentage.



Now is the time to create what will be happening in loops on the matrix.

For starters, we need the "loop forever" block in the "Loops" block section.



We want for the display to turn black, and have the slider and width display values written on it.

For that, your code needs to look like this:

Search...

- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

```
set width to 0
set slider_value to 0

when slider Middle moves
  set slider_value to get slider Middle value
  set width to slider_value ÷ 235
  set width_display to round down width × 100

loop forever
  fill frame with Black
  write create text with " slider_value: " slider_value
  write create text with " width: " width_display "%"
  push frame
```

Make sure the value for "width" is written as a percentage.

Don't forget to include the "push frame" block at the end to guarantee that the code shows on the screen.

Now, let's make the main part on the big matrix.

The first step is to turn off all of the LEDs on that matrix.

You do this with the "fill matrix with 0" block. When the LED intensity is adjusted to 0, the LEDs turn off.

Search...

- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

The image shows a Scratch script with the following blocks:

- set width to 0
- set slider\_value to 0
- when slider Middle moves
  - set slider\_value to get slider Middle value
  - set width to slider\_value + 235
  - set width\_display to round down width \* 100
- loop forever
  - fill frame with Black
  - write create text with "slider\_value:" x: 0 y: 0 color: White slider\_value
  - write create text with "width:" x: 0 y: 10 color: White width\_display "%"
  - push frame
  - fill matrix Big frame with 0

Check if you put the correct matrix in the block (big one in this case).

The block for drawing the rectangle can be found in the "Matrix" block section.

The image shows a Scratch script with the following blocks:

- set width to 0
- set slider\_value to 0
- when slider Middle moves
  - set slider\_value to get slider Middle value
  - set width to slider\_value + 235
  - set width\_display to round down width x 100
- loop forever
  - fill frame with Black
  - write create text with "slider\_value:" x: 0 y: 0 color: White slider\_value
  - write create text with "width:" x: 0 y: 10 color: White width\_display %
  - push frame
  - fill matrix Big frame with 0
  - in matrix Big draw filled rectangle
    - width: round down width x 8
    - height: 9
    - x: 0
    - y: 0
    - intensity: 100

The rectangle's height is set to 9, and its width is relative to the number width x 8 (8 being the maximum width).

To turn on the LEDs, make sure the intensity is set to 100.

Don't forget to add the "push frame to matrix" to ensure the code really shows on the matrix.



Search...

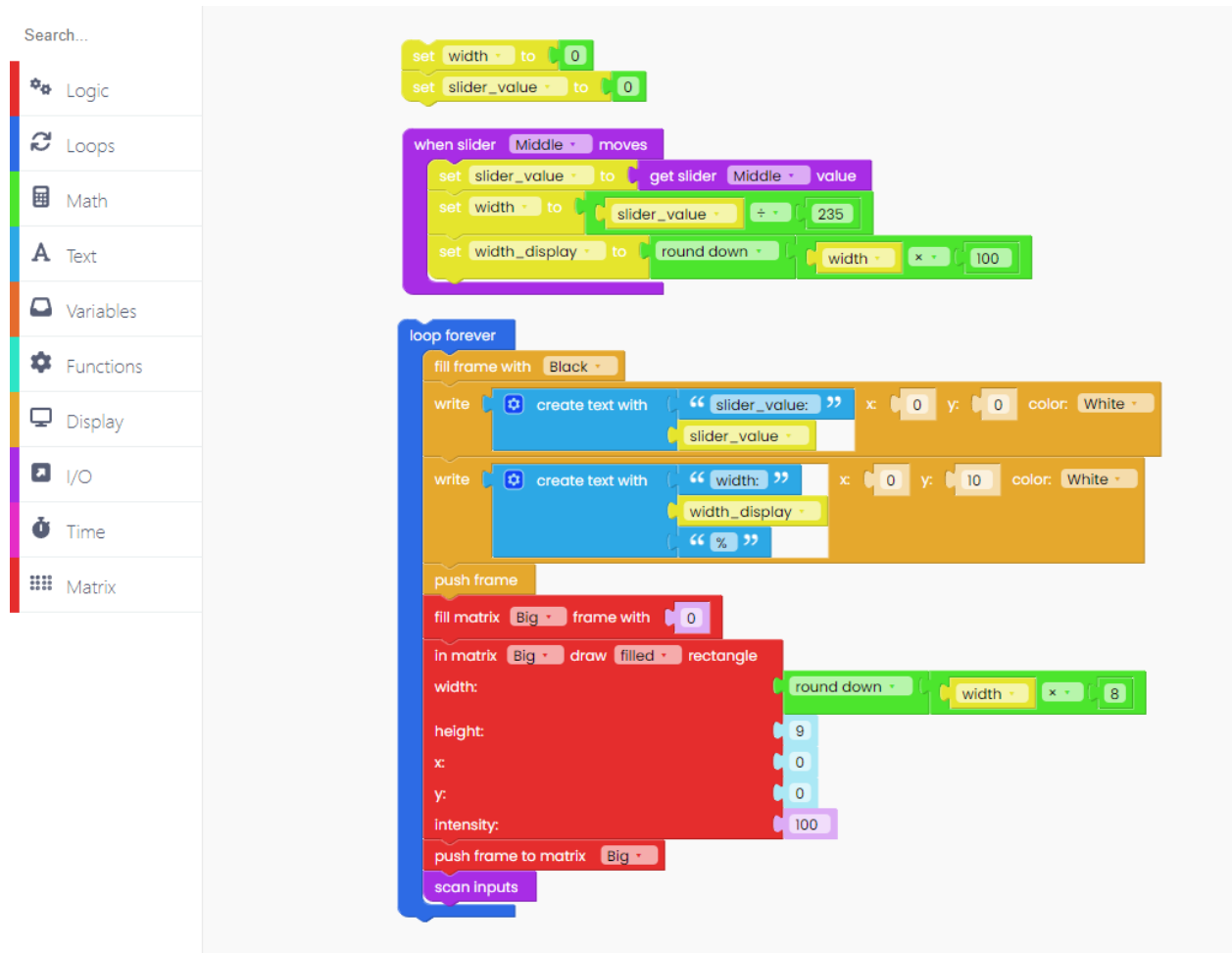
- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

```
set width to 0
set slider_value to 0

when slider Middle moves
  set slider_value to get slider Middle value
  set width to slider_value ÷ 235
  set width_display to round down width × 100

loop forever
  fill frame with Black
  write create text with " slider_value: " slider_value color: White
  write create text with " width: " width_display color: White
  push frame
  fill matrix Big frame with 0
  in matrix Big draw filled rectangle
  width: round down width × 8
  height: 9
  x: 0
  y: 0
  intensity: 100
  push frame to matrix Big
```

Another thing to keep in mind when using any of the inputs is to include the "scan inputs" block inside the "loop forever" block.



Click on the Run button, and start moving the slider.

Is the rectangle expanding?

## Simple timer

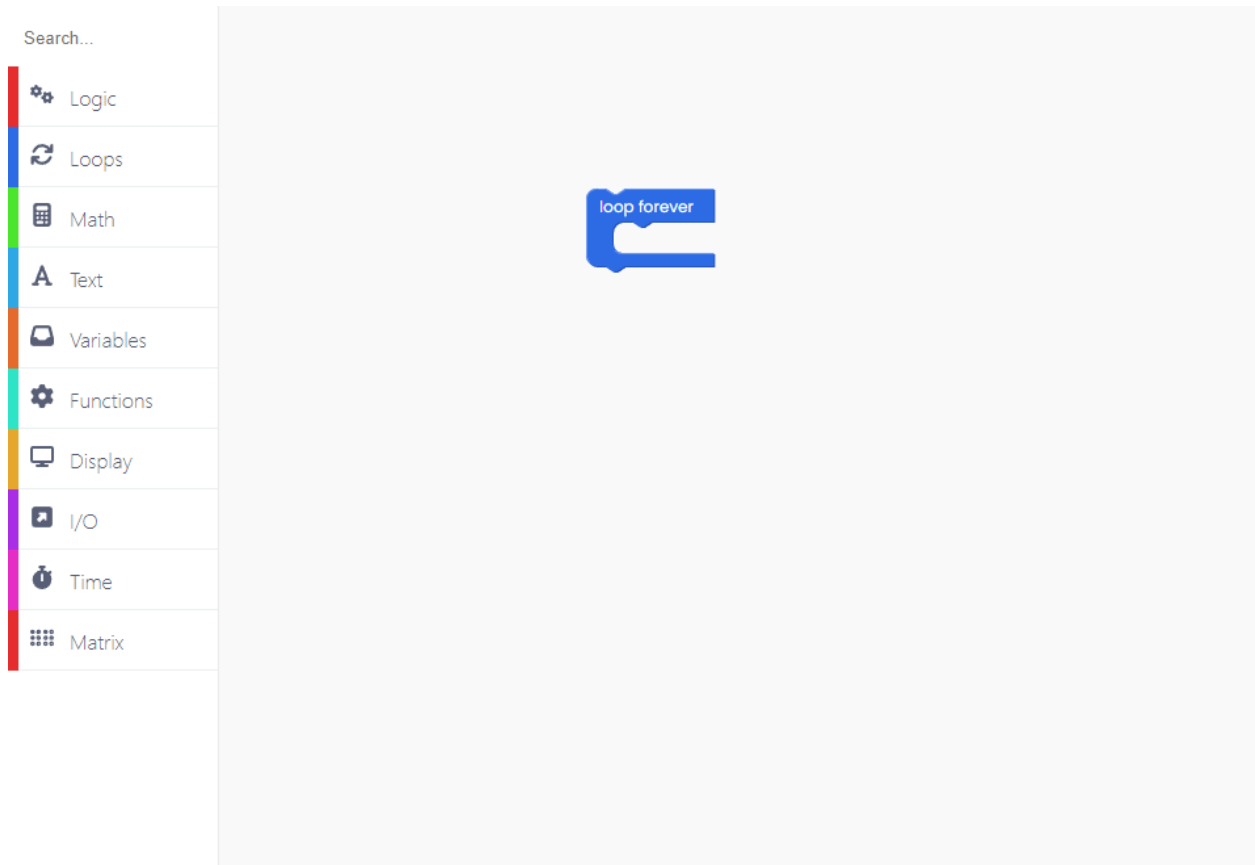
Now is the time to write simpler code.

This time, we'll learn how to switch the LEDs on the matrix on and off in loops with a one-second delay.

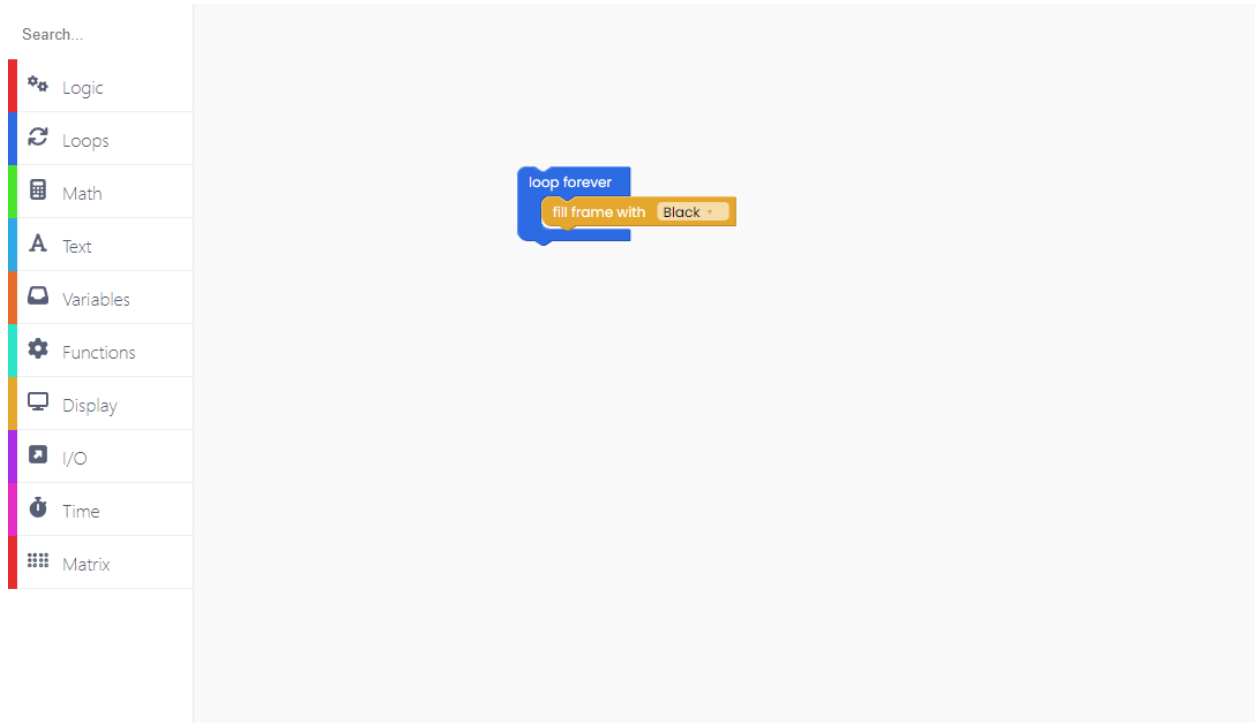
When the LEDs are turned on, the display turns white; when they are turned off, the display

turns black.

Because we want this to happen in the loops all the time, first drag and drop the "loop forever" block onto the drawing area:

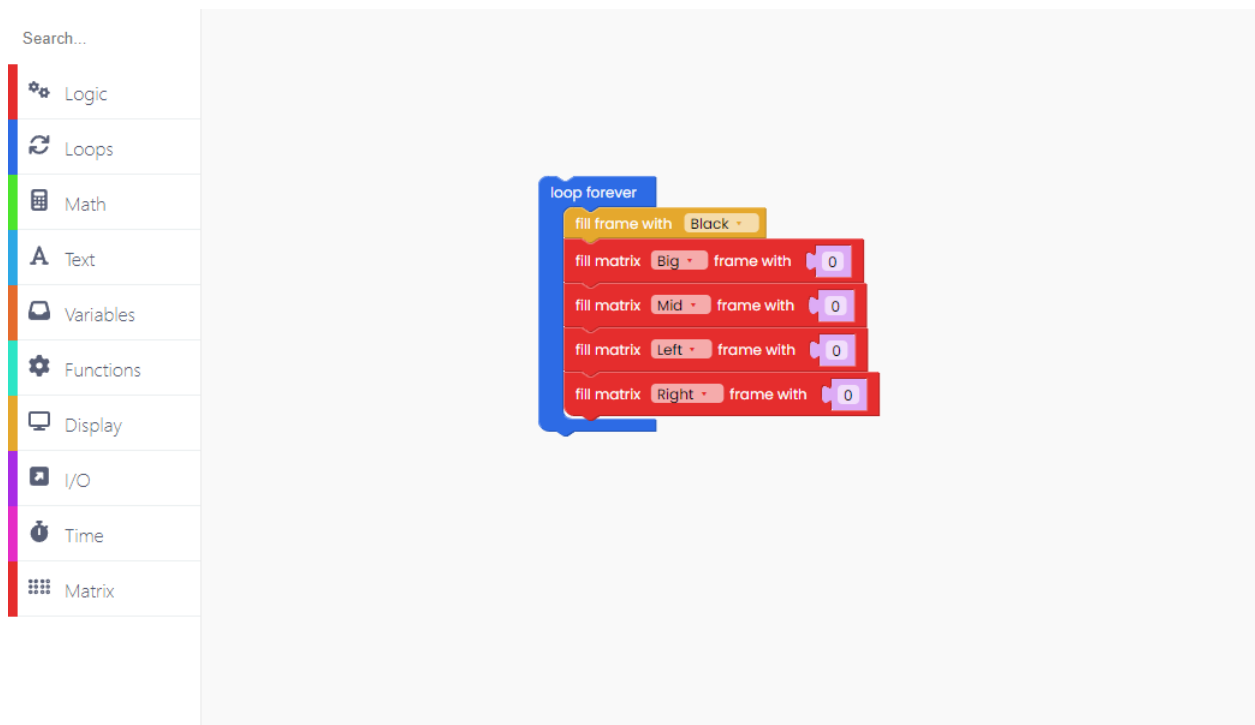


First, we want to make the display black, and the block for that is in the "Display" section.

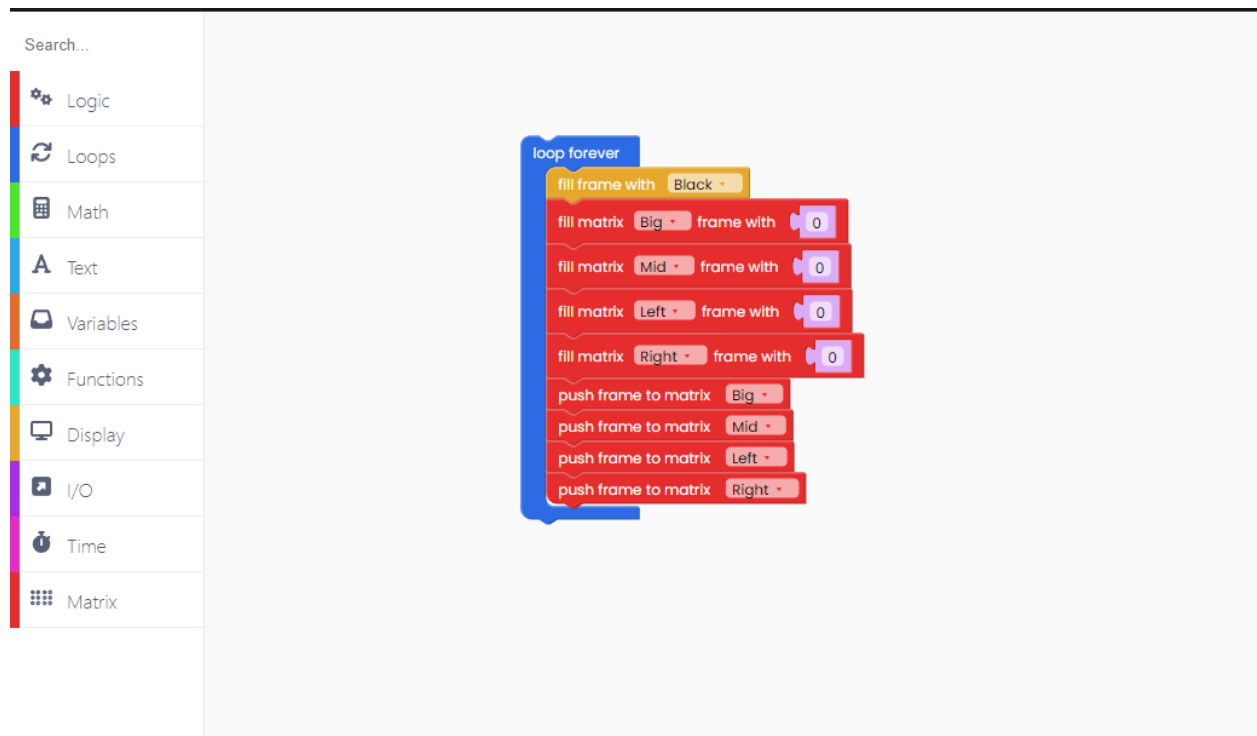


Put the blocks that will turn off the LEDs on each matrix below that.

If the intensity is set to 0, the LEDs are turned off.



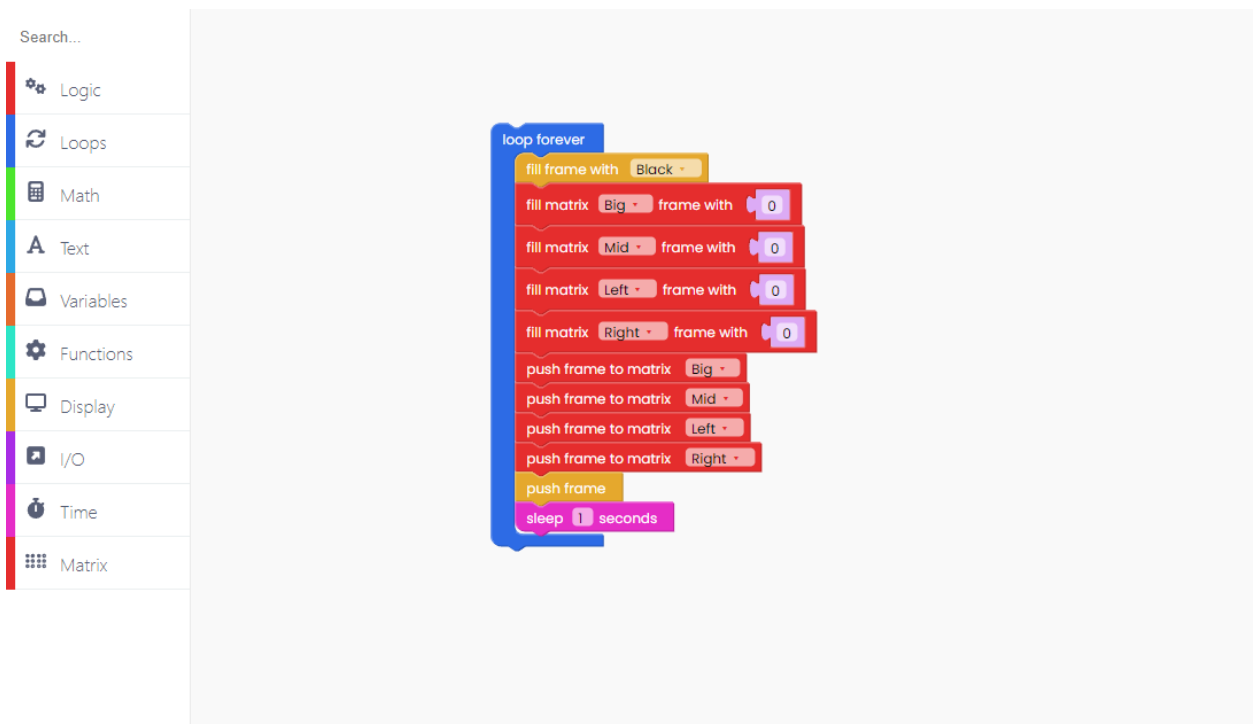
To ensure that the LEDs truly turn off, we must include "push frame to matrix" blocks for each matrix.



You must also include the "push frame" block from the "Display" block section to ensure that the display truly goes black.



Now, add a one-second delay between turning on and off the LEDs.



The next step is to make the display white and all of the LEDs turned on.

Search...

- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

```
loop forever
  fill frame with Black
  fill matrix Big frame with 0
  fill matrix Mid frame with 0
  fill matrix Left frame with 0
  fill matrix Right frame with 0
  push frame to matrix Big
  push frame to matrix Mid
  push frame to matrix Left
  push frame to matrix Right
  push frame
  fill frame with White
  fill matrix Big frame with 100
  fill matrix Mid frame with 100
  fill matrix Left frame with 100
  fill matrix Right frame with 100
```

Remember to include the "push frame to matrix" and "push frame" blocks to ensure that these changes are visible.

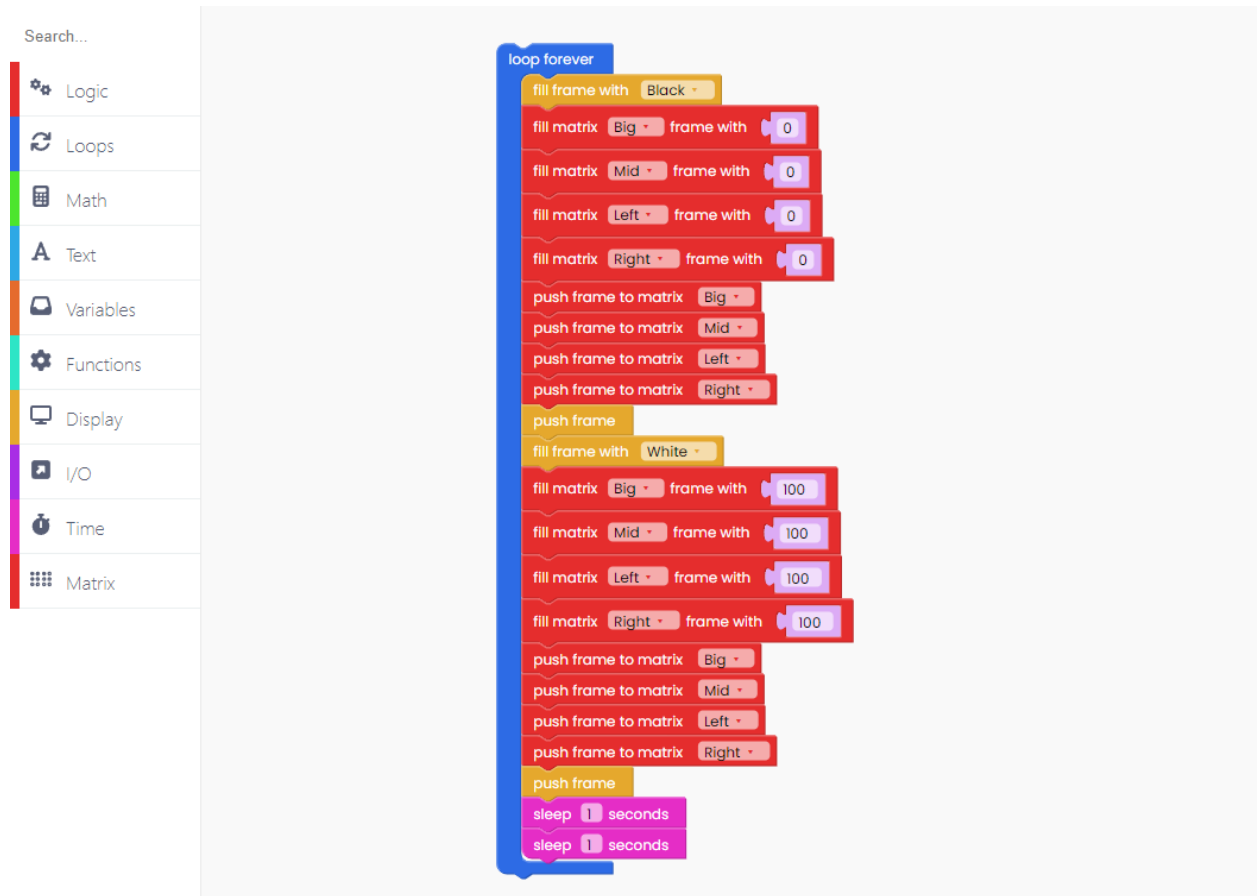
Search...

- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

```
loop forever
  fill frame with Black
  fill matrix Big frame with 0
  fill matrix Mid frame with 0
  fill matrix Left frame with 0
  fill matrix Right frame with 0
  push frame to matrix Big
  push frame to matrix Mid
  push frame to matrix Left
  push frame to matrix Right
  push frame
  fill frame with White
  fill matrix Big frame with 100
  fill matrix Mid frame with 100
  fill matrix Left frame with 100
  fill matrix Right frame with 100
  push frame to matrix Big
  push frame to matrix Mid
  push frame to matrix Left
  push frame to matrix Right
  push frame
```

Introduce another delay before the loop starts again.





Click on the Run button and check it out!

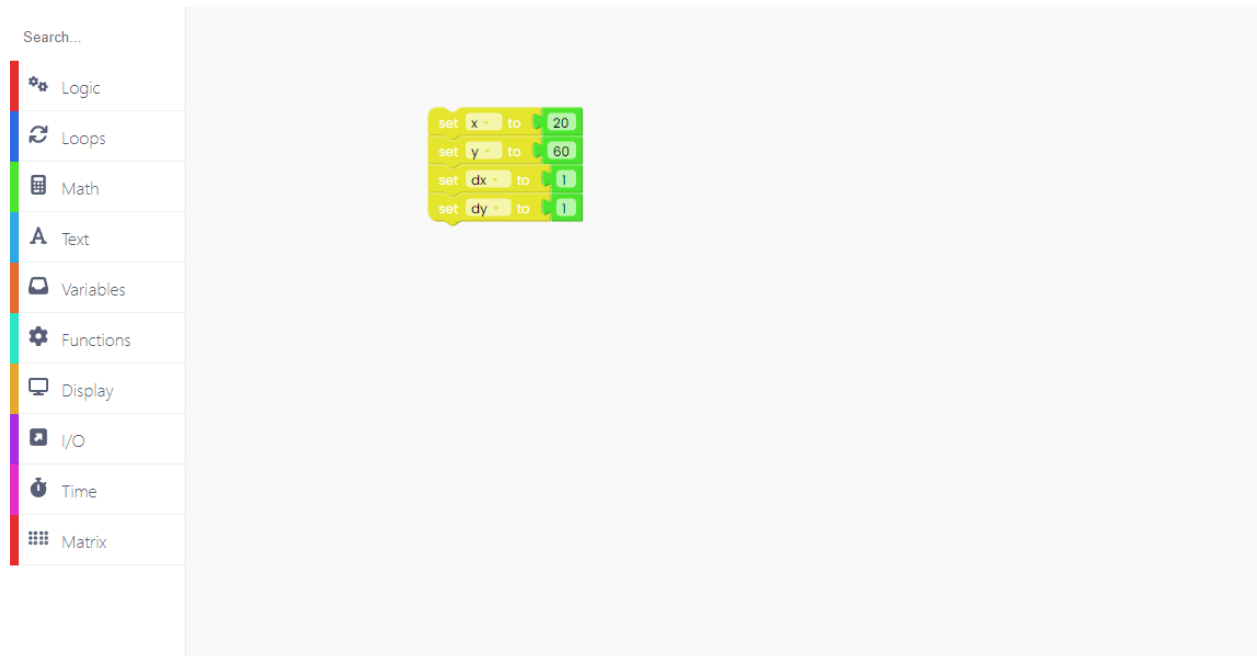
## Display

After an easy one, it's time for a slightly more difficult code.

In this example, we'll see how to make a green circle on the screen that bounces off the edges.

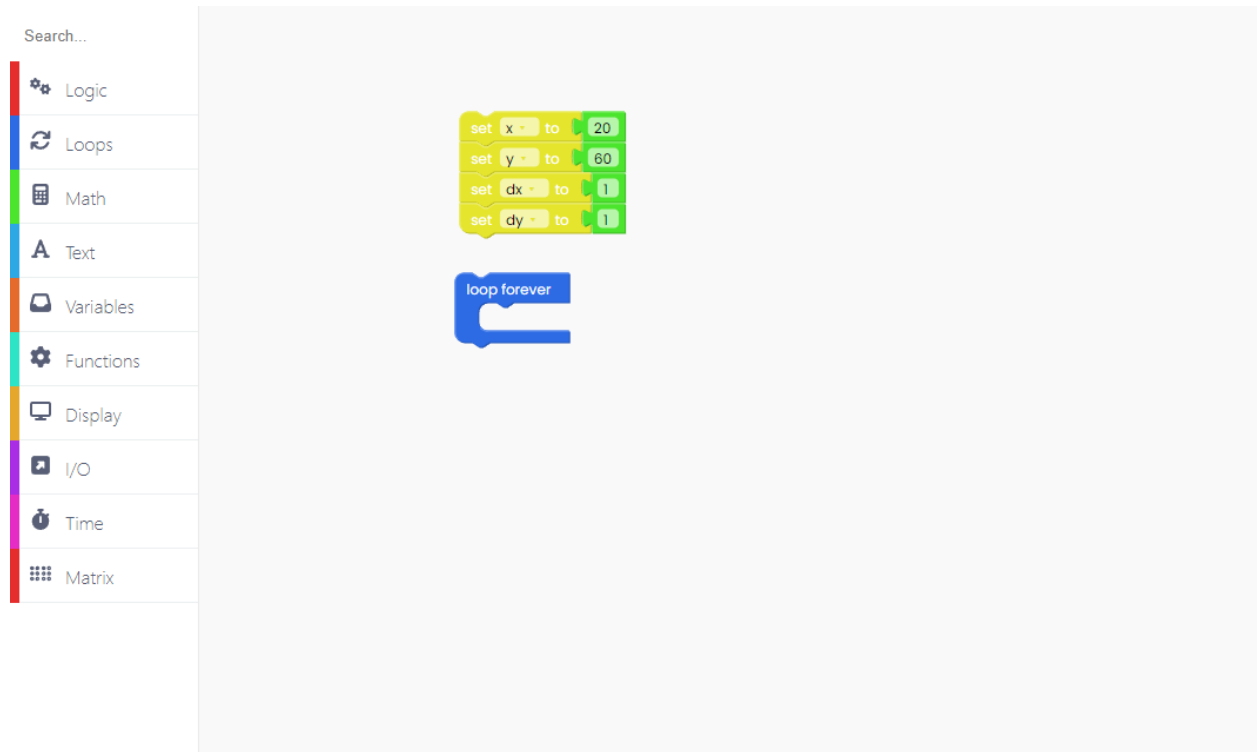
To begin, we must create four variables ( $x$ ,  $y$ ,  $dx$ , and  $dy$ ).

$X$  and  $Y$  represent the coordinates of the center of the circle, and  $DX$  and  $DY$  represent the amount by which the  $x$  and  $y$  coordinates move.



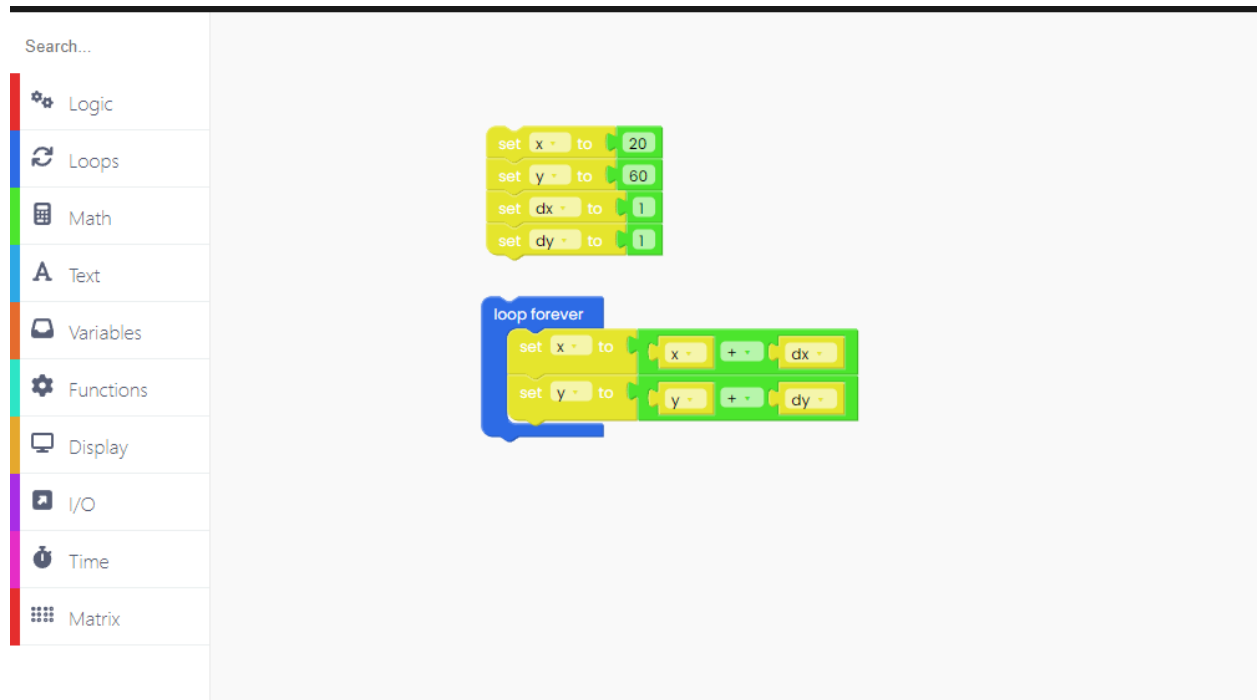
Now, let's introduce the "loop forever" block, which will include all of the code that will run continuously.

We'll set the 'x' and 'y' to 20 and 60, and they'll always change by one, therefore the 'dx' and 'dy' must be set to 1.



Every 20 ms, the 'x' and 'y' variables will change by one.

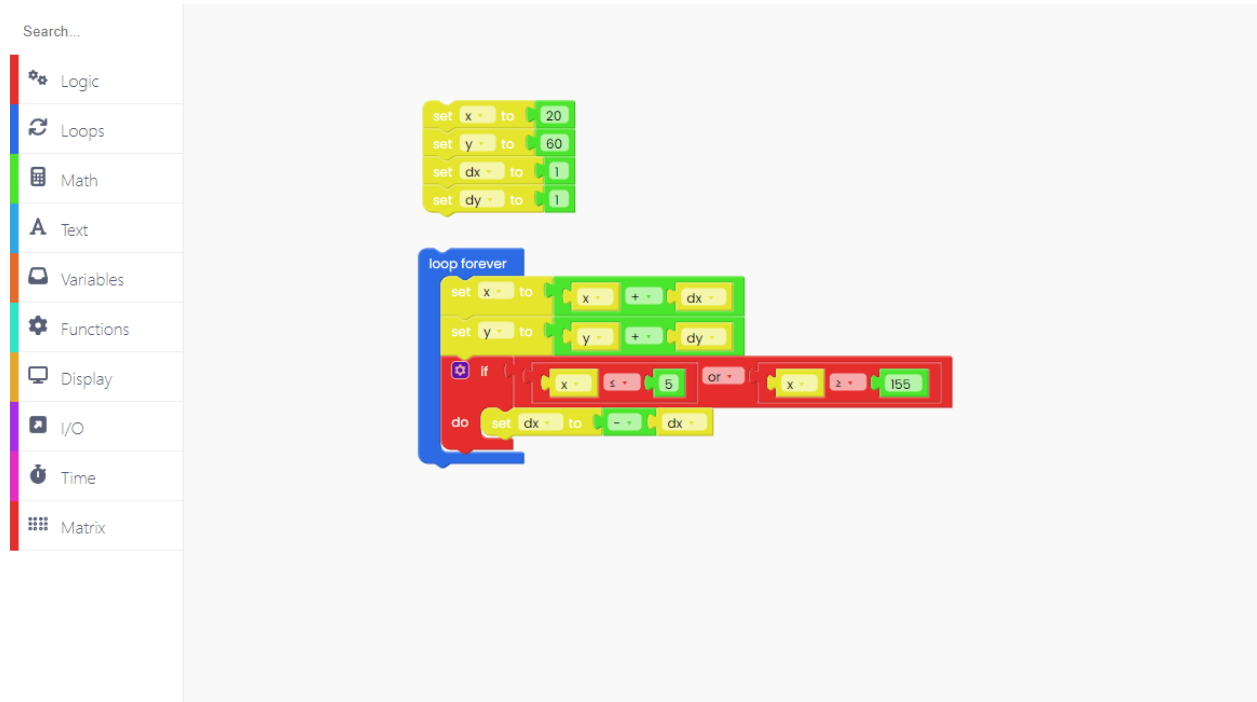
As a result, we must set the value of these variables to the sum of that variable's value and 1.



When the circle reaches the display's edges, it will begin to shrink in size.

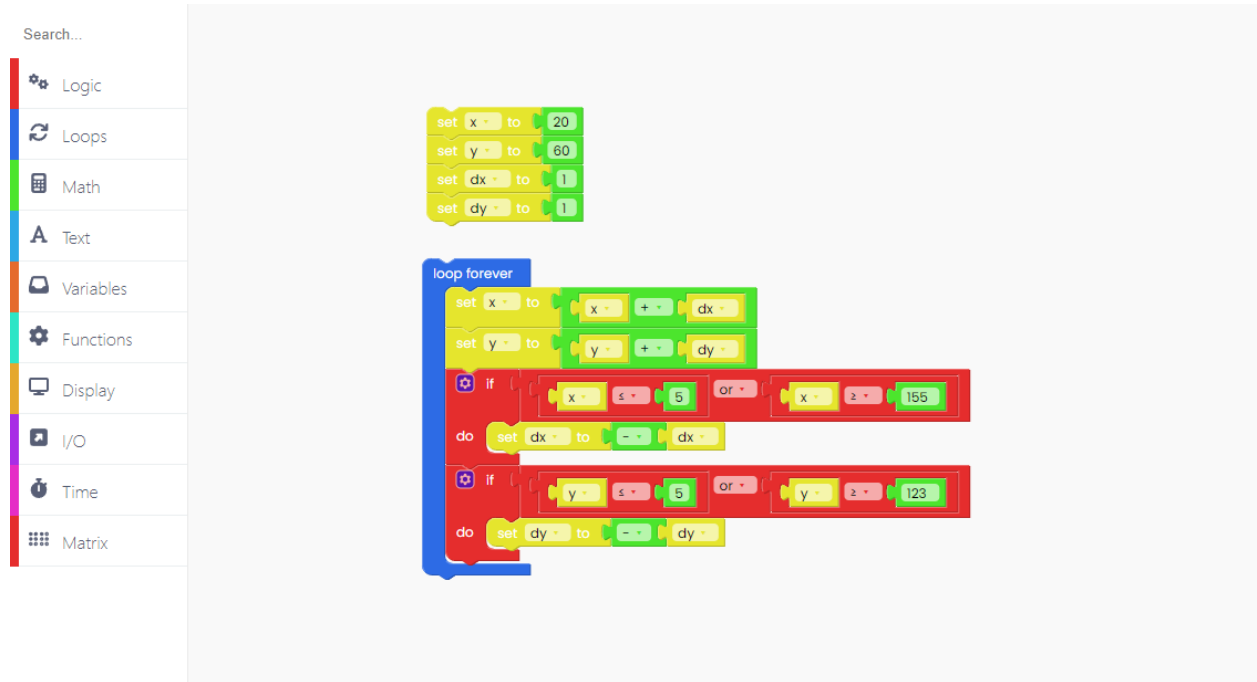
Because of that, we have to change its direction.

We do that with the "Logic" block.



So, whenever the circle reaches the edges, we chose to change the variable 'dx' to '-dx'.

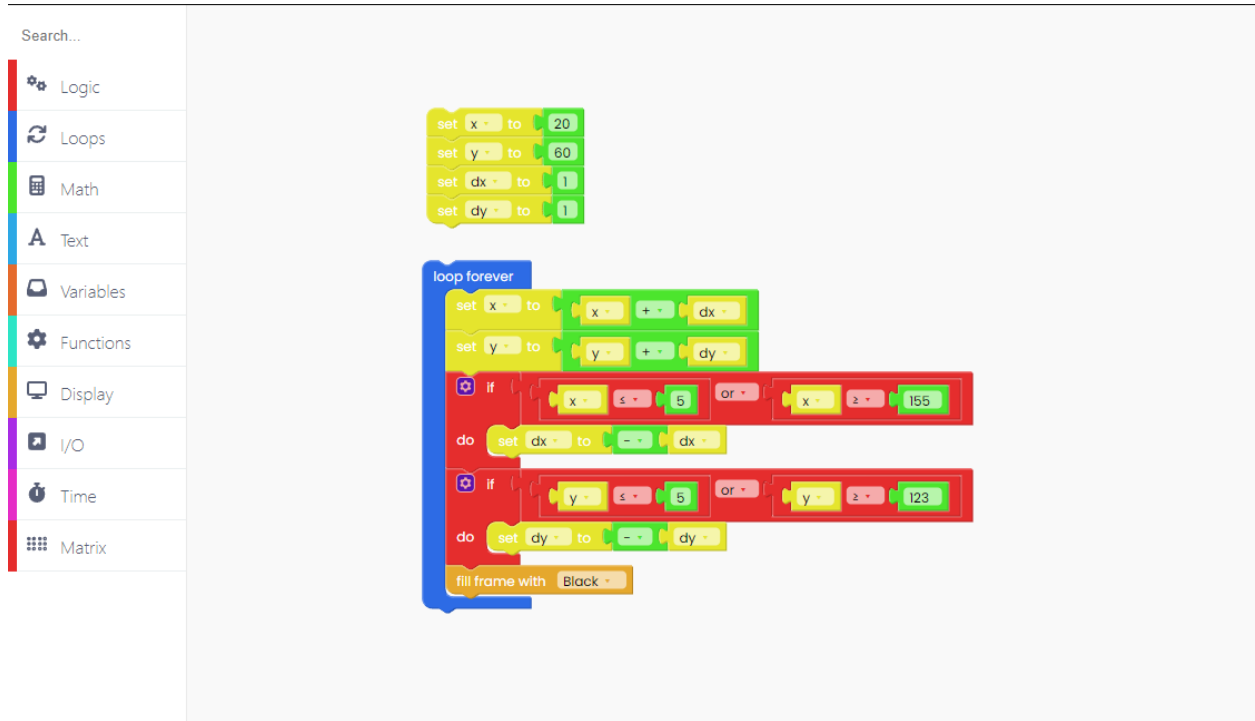
The same goes for the 'y' and 'dy' variables.



Great!

The only thing left to do is draw a circle.

To do that, first we have to fill the background of the display with black:



Draw the filled circle in color green.

The radius of our circle is set to 5, and the x and y coordinates are set to the initial values of these variables.

Search...

- Logic
- Loops
- Math
- Text
- Variables
- Functions
- Display
- I/O
- Time
- Matrix

```
set x to 20
set y to 60
set dx to 1
set dy to 1

loop forever
  set x to x + dx
  set y to y + dy
  if (x <= 5 or x >= 155)
    do set dx to -dx
  if (y <= 5 or y >= 123)
    do set dy to -dy
  fill frame with Black
  draw filled circle
  radius: 5
  x: x
  y: y
  color: Green
```

The image shows a Scratch code editor with a sidebar on the left containing categories: Logic, Loops, Math, Text, Variables, Functions, Display, I/O, Time, and Matrix. The main workspace contains a script starting with four 'set' blocks: 'set x to 20', 'set y to 60', 'set dx to 1', and 'set dy to 1'. This is followed by a 'loop forever' block containing: 'set x to x + dx', 'set y to y + dy', an 'if' block with condition '(x <= 5 or x >= 155)' and 'do' block 'set dx to -dx', another 'if' block with condition '(y <= 5 or y >= 123)' and 'do' block 'set dy to -dy', a 'fill frame with Black' block, and a 'draw filled circle' block with parameters: radius: 5, x: x, y: y, color: Green.

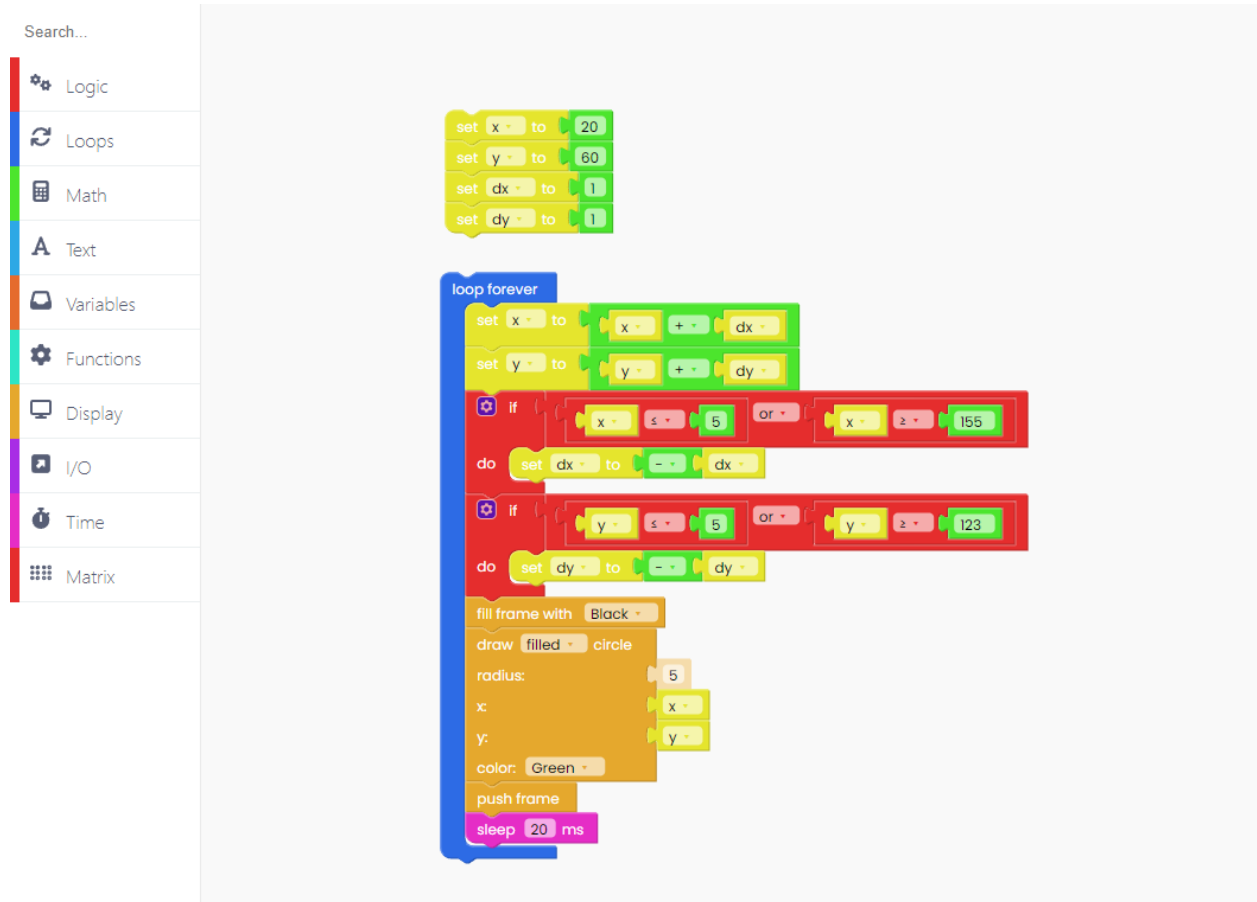
To ensure that this circle appears on the display, add the "push frame" block at the end.



The image shows a Scratch script for a bouncing ball simulation. The script is organized as follows:

- Initialization:** Four 'set' blocks are used to initialize variables: 'x' to 20, 'y' to 60, 'dx' to 1, and 'dy' to 1.
- Loop:** A 'loop forever' block contains the following steps:
  - Update 'x' to 'x + dx'.
  - Update 'y' to 'y + dy'.
  - Horizontal Bounce:** An 'if' block with the condition 'x <= 5 or x >= 155'. The 'do' block contains 'set dx to -dx'.
  - Vertical Bounce:** An 'if' block with the condition 'y <= 5 or y >= 123'. The 'do' block contains 'set dy to -dy'.
  - Drawing:** A sequence of blocks: 'fill frame with Black', 'draw filled circle' (with radius 5, x and y coordinates, and color Green), and 'push frame'.

And, finally, add the time block "sleep 20 ms".



Click on the Run button and check it out!

## Get creative

You've reached the end of our first Jay-D coding tutorial, congratulations!

We hope you're as excited as we are about Jay-D'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 Jay-D's lights, display, or remixed by sharing it on the CircuitMess community forum: <https://community.circuitmess.com/>

Ready for more advanced stuff? Once you master CircuitBlocks and are ready for more

advanced programming, try using PlatformIO for programming new functions:

Check out our guide for that here:

- [PlatformIO Jay-D programming guide](#)

If you need any help with your device, as always, reach out to us via [contact@circuitmess.com](mailto:contact@circuitmess.com) and we'll help as soon as we can.

Thank you and keep making!

