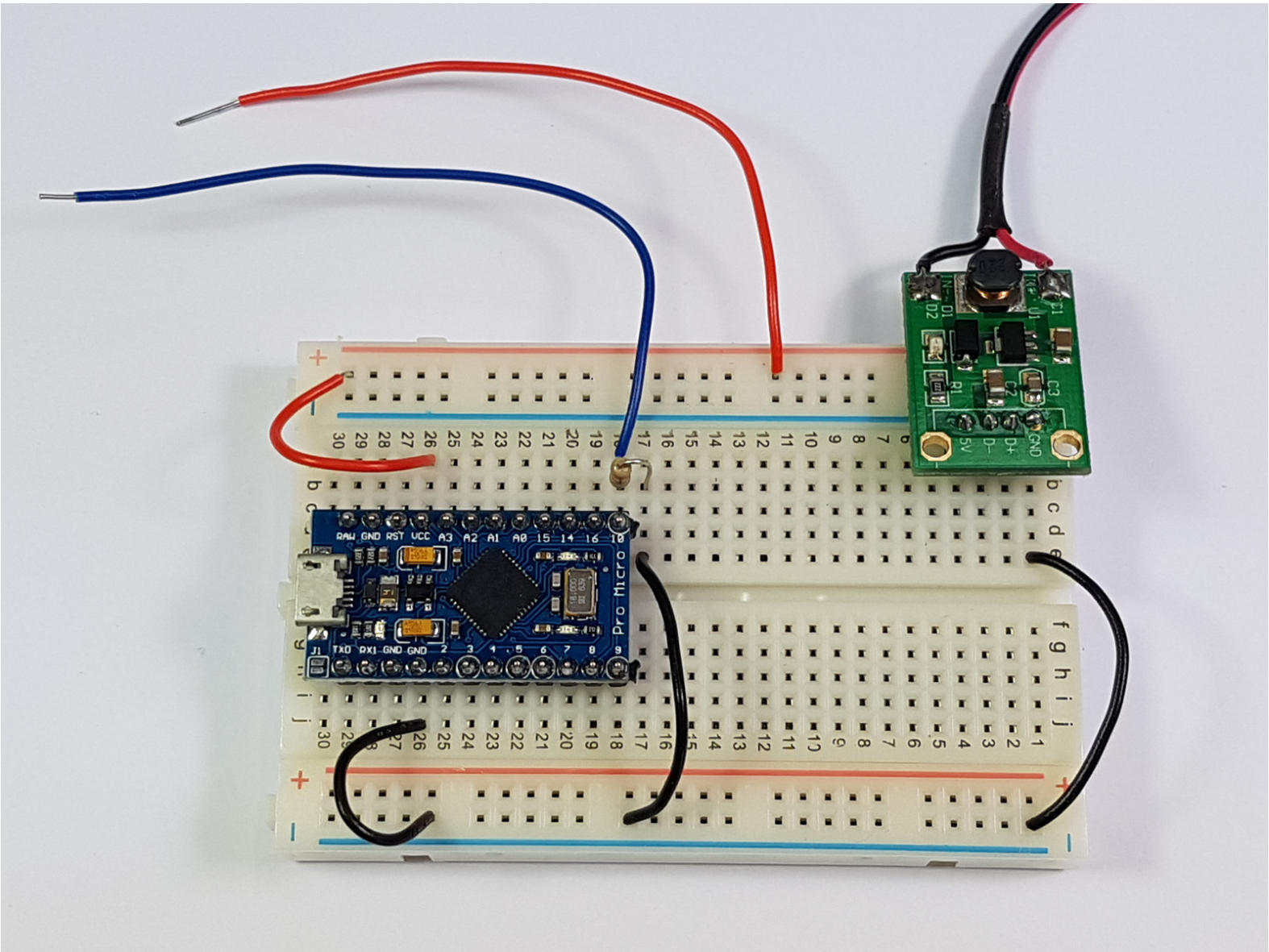
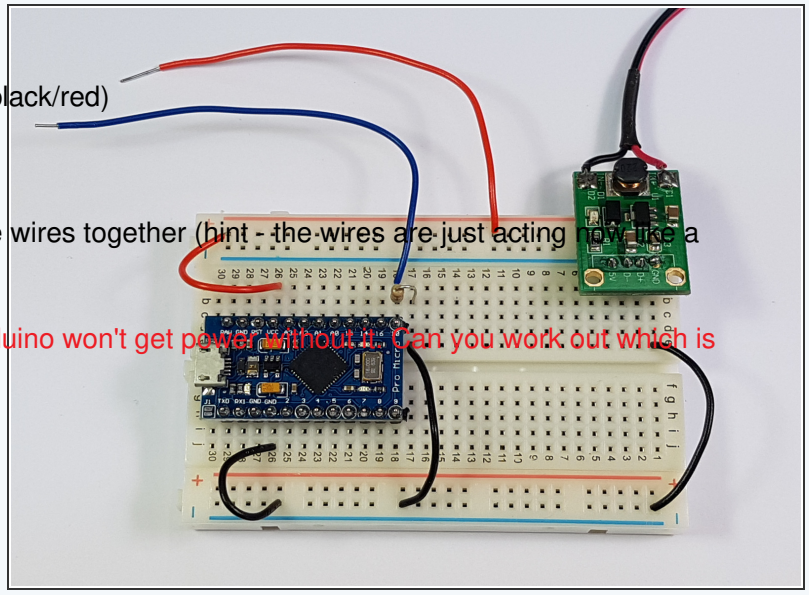
Inventor School Session 4 - Conductors and insulators



Step 1

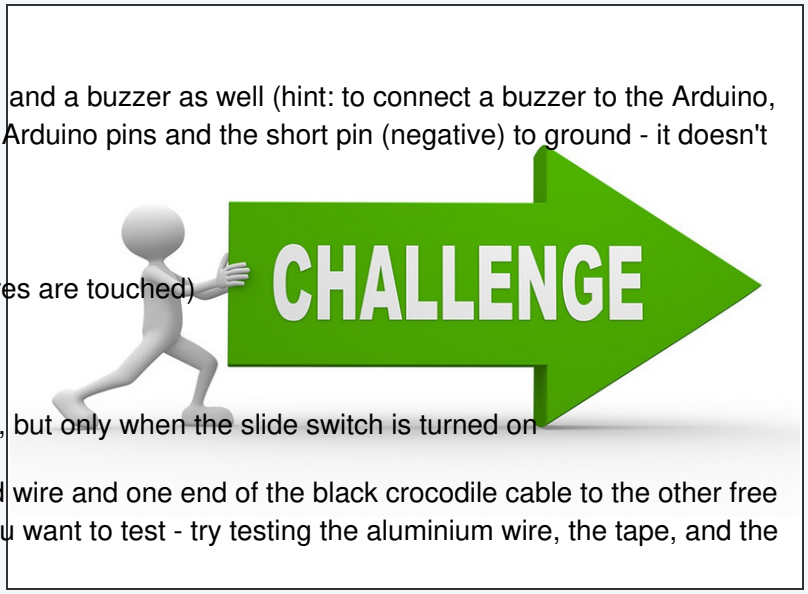
Step 1 — Continuity indicator



- Wire up the circuit shown, using a 1k resistor (brown/black/red)
- Add an LED and a resistor to one of the output pins
- Write a programme to turn on the LED if you touch the wires together (hint - the wires are just acting now like a switch, so you can detect this the same way)

⚠ We've actually left out one wire on this circuit - the Arduino won't get power without it. Can you work out which is missing and add it in?

Step 2 — Advanced continuity indicator



- Now add a bicolor LED into your circuit, a slide switch, and a buzzer as well (hint: to connect a buzzer to the Arduino, just connect the long pin (positive) to one of the spare Arduino pins and the short pin (negative) to ground - it doesn't need a resistor.
- Write a programme to:
  - Turn the green LED on if there is continuity (the wires are touched)
  - Turn the red LED on if there is not
  - Turn the buzzer on as well when there is continuity, but only when the slide switch is turned on
- Clip one end of your red crocodile cable to the free red wire and one end of the black crocodile cable to the other free wire. Now you can clip the other ends to something you want to test - try testing the aluminium wire, the tape, and the black plastic bag material.

Step 3

Step 3

— Making a pressure switch

- We're going to make a secret pressure switch - you can slip this under a door mat to detect when someone walks over it
- Watch the video to see how to make the switch, and follow along to make your own!

Step 4

— Testing the pressure switch

- Your pressure switch should work like any of the other switches - use jumper wires and alligator clips to connect it to your circuit.
- Your challenge is to sound a buzzer **and** flash a light when the pressure switch is stepped on!

Step 5

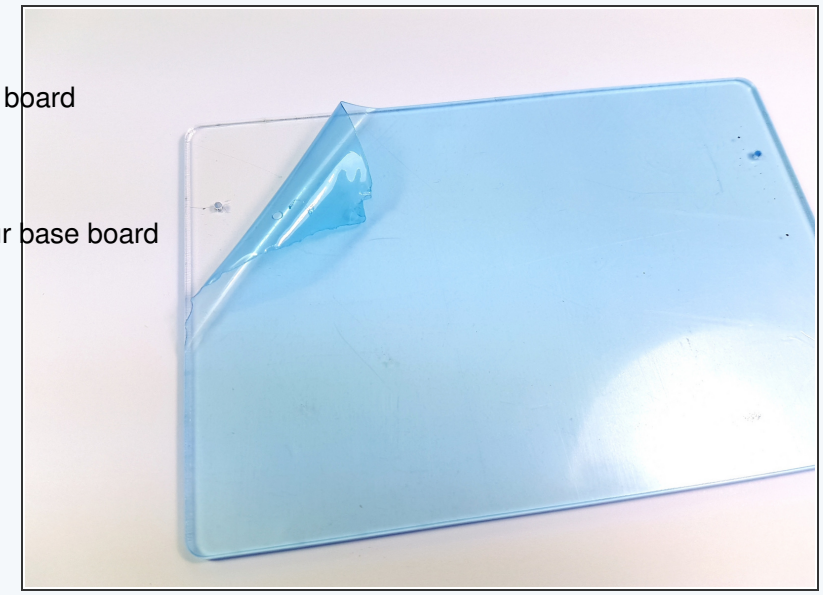
Step 5

— Preparing your base board

- Peel off the protective layer of plastic from your base board

Your base board may be a different colour!

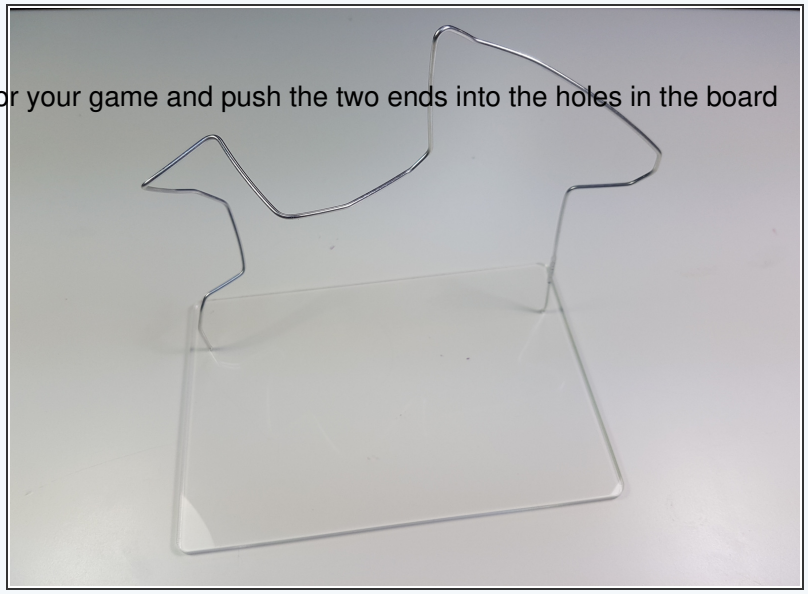
There may be protective plastic on **both** sides of your base board



Step 6

— Make your path

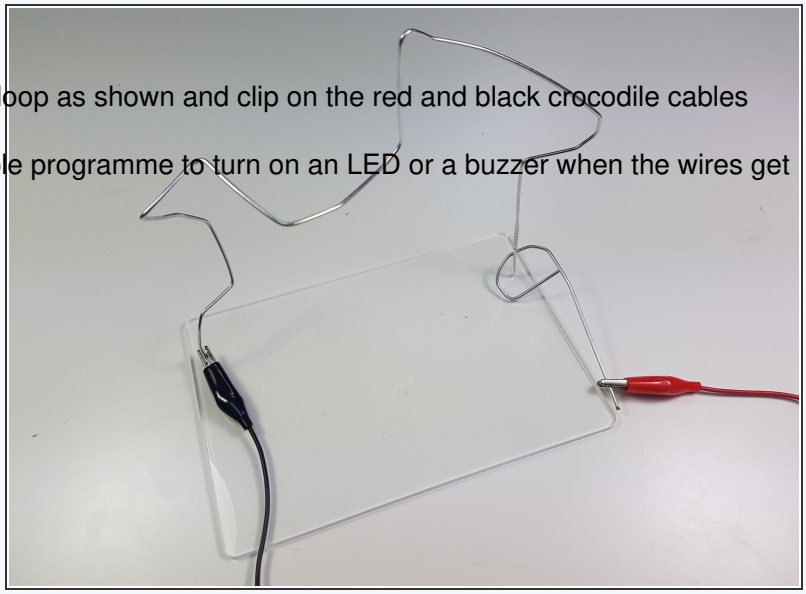
- Bend your long piece of aluminium wire into a shape for your game and push the two ends into the holes in the board



Step 7

Step 7 — Connecting the circuit

- Now bend the end of your smaller piece of wire into a loop as shown and clip on the red and black crocodile cables
- Now connect it to your Arduino board and write a simple programme to turn on an LED or a buzzer when the wires get connected

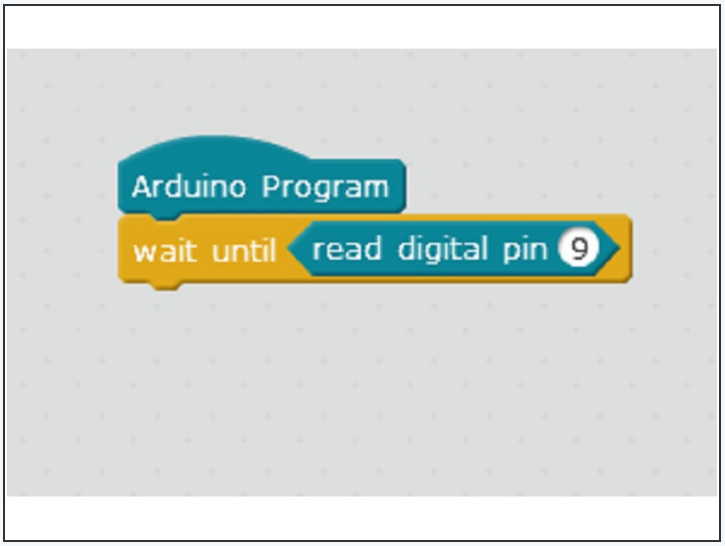


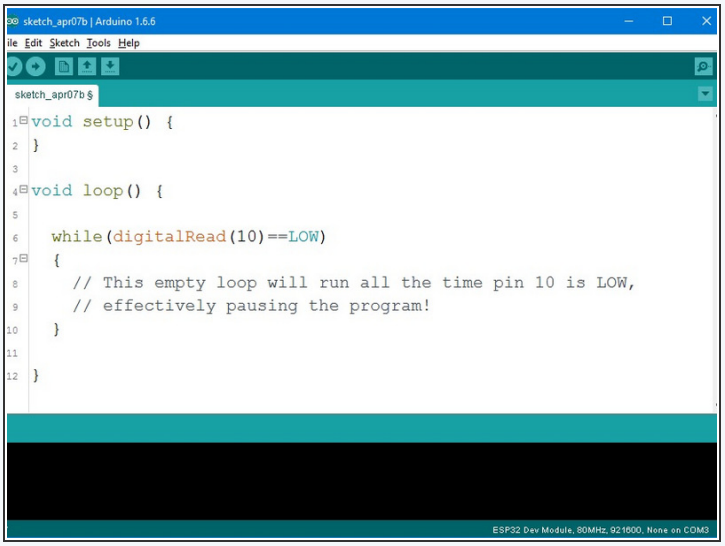
Step 8 — Advanced steady hand game

- Let's change our steady hand game just a little ...
- When you connect the wires touch, you should sound a buzzer for a fixed time (e.g. 1 second) so that it's easier to hear when the game is over.

Step 9

Step 9 — Three Strikes and You're Out

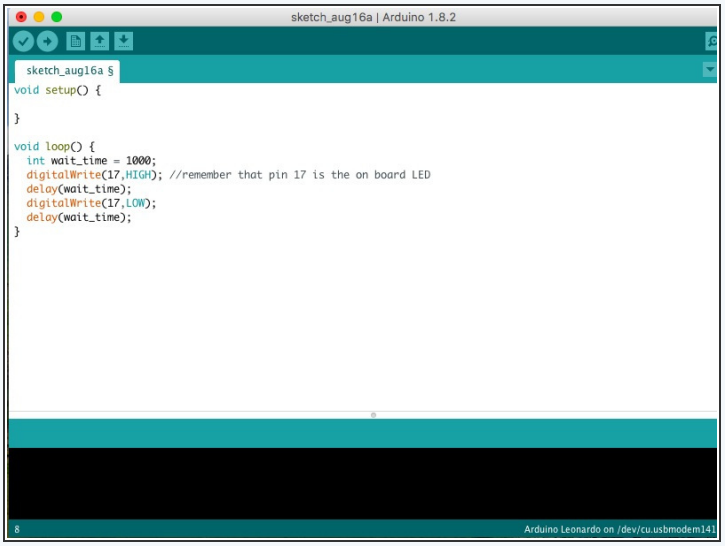


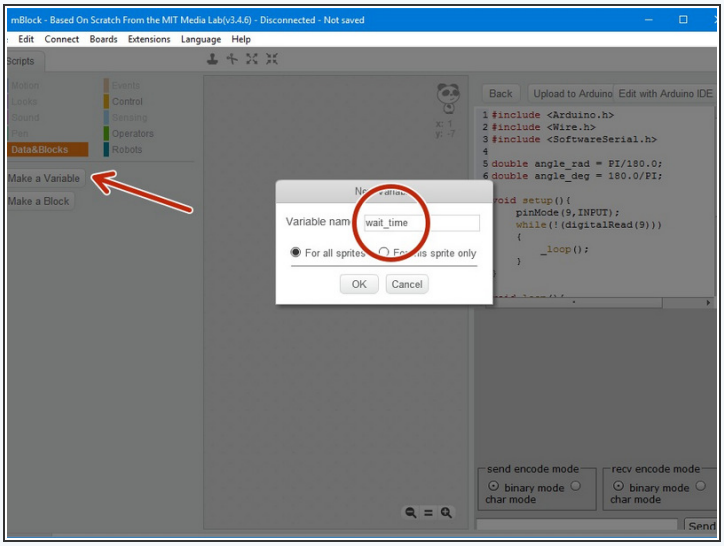


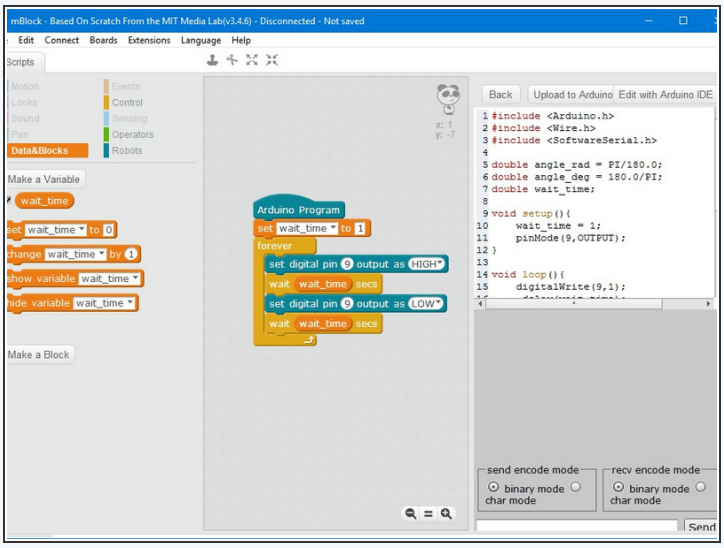
- Ideally we want to have a number of tries before its **'game over'.**
- Using the **'wait until'** block (or an **empty while loop** for the Arduino users - see the picture for an example), extend your steady hand code so that the player has to hit the wire **3 times** before the buzzer sounds for a long time.
- Each time they do hit the wire, make sure the buzzer **still sounds quickly** so they know they have made a mistake!
- Hint: you will need **3** wait until blocks (or **3** while loops), each of which gets the program to **wait** until the wire is touched.

Step 10

— Variables

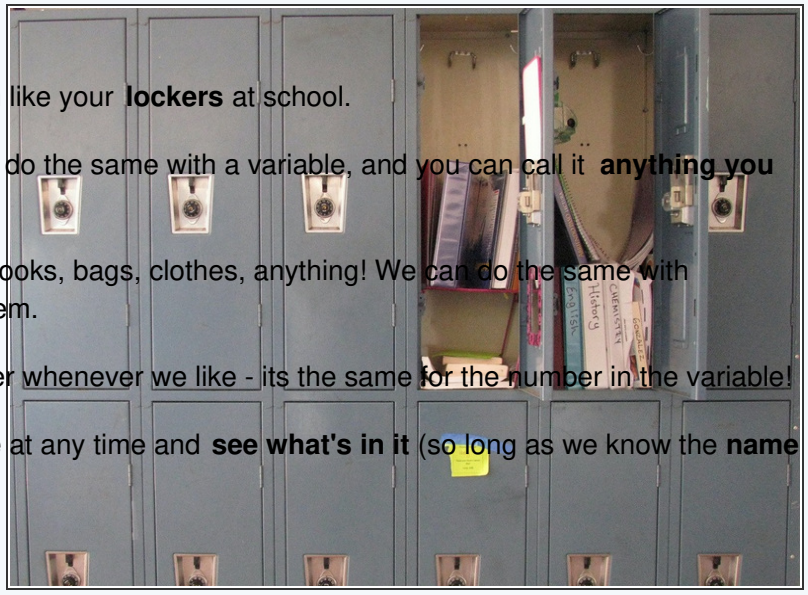






- Hopefully your code for several tries worked, but what if you wanted the game to have **20 tries?**
- That would take an **awful lot of code!**
- Luckily, we can use a programming tool called **variables** to make the program much **simpler**, and easier to change for larger numbers of tries.
- **Build** the sample program in the picture, which uses a **variable** - can you guess what your circuit will do?
- **MBlock users** - you will need to create a new variable first in the **Data&Blocks** menu.

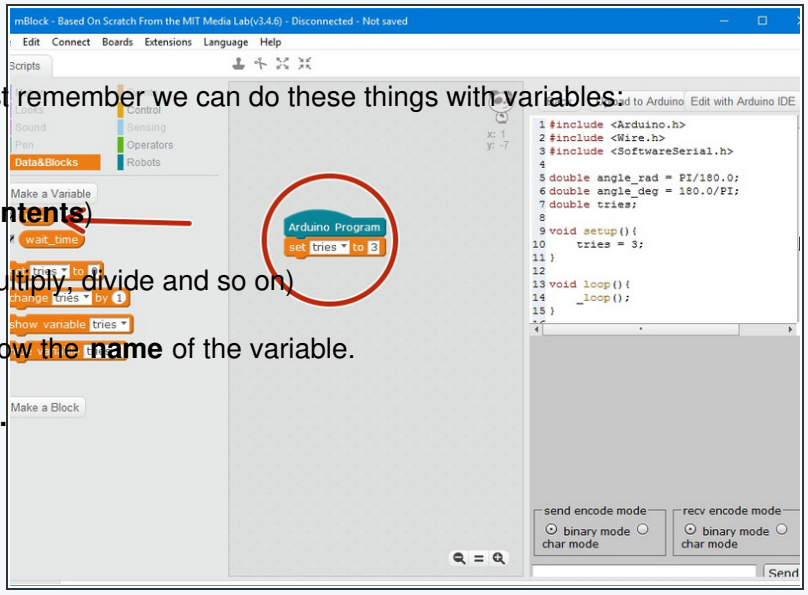
— What are Variables?



- A good way to understand variables is to think of them like your **lockers** at school.
- To use a locker, you need to put your **name** on it - we do the same with a variable, and you can call it **anything you want!**
- We can then put whatever we like inside the locker - books, bags, clothes, anything! We can do the same with variables, but for now we'll just put **numbers** inside them.
- We can **add, remove and change things** in the locker whenever we like - its the same for the number in the variable!
- Most usefully, we can go back to the locker or variable at any time and **see what's in it** (so long as we know the **name** of the locker or variable!).

Step 12

— Using Variables (M)

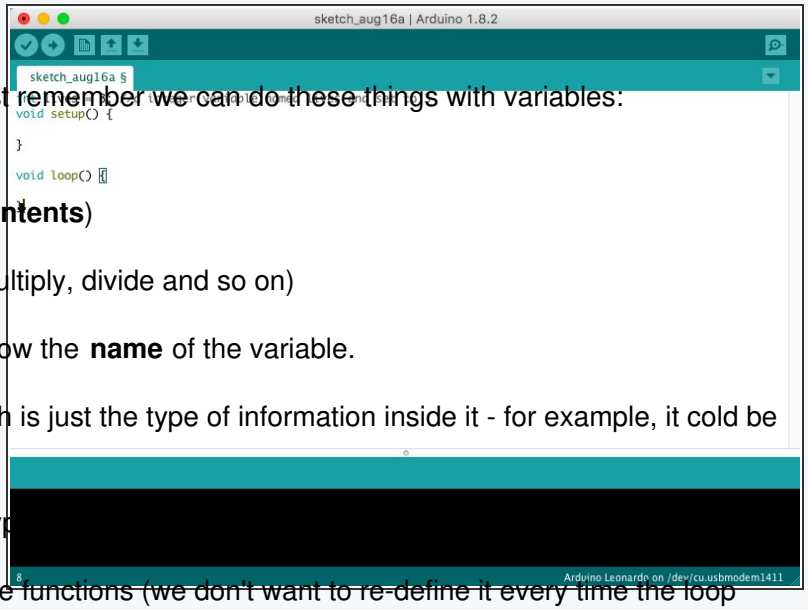


- If you don't quite understand, don't worry - for now, just remember we can do these things with variables:
  - **Call** them anything we like (variable **name**)
  - **Store** any number we like inside them (variable **contents**)
  - **Change** the contents at any time (add, subtract, multiply, divide and so on)
  - **Access** the contents at any time, so long as we know the **name** of the variable.
- Make a new variable called **tries**, and set it equal to **3.**

Step 13

Step 13

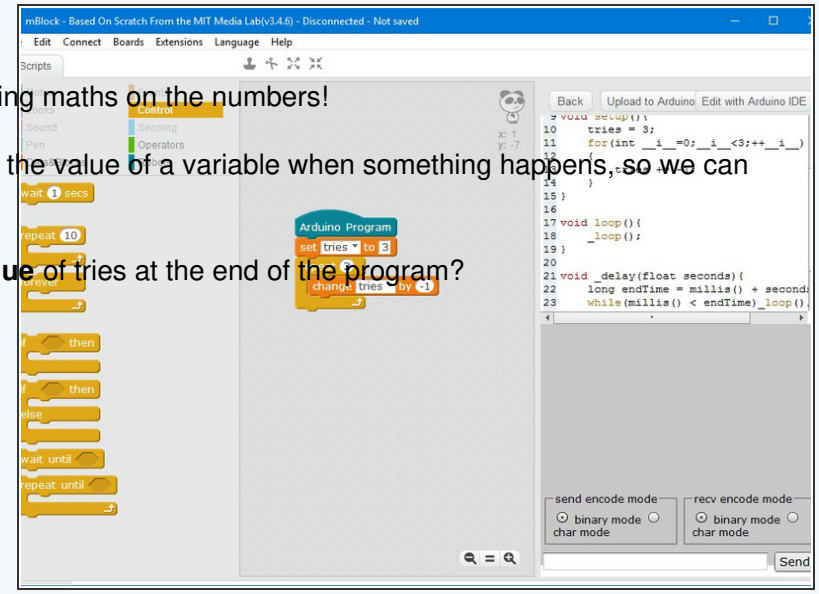
— Using Variables (A)



- If you don't quite understand, don't worry - for now, just remember we can do these things with variables:
  - **Call** them anything we like (variable **name**)
  - **Store** any number we like inside them (variable **contents**)
  - **Change** the contents at any time (add, subtract, multiply, divide and so on)
  - **Access** the contents at any time, so long as we know the **name** of the variable.
- We also need to specify the **type** of the variable, which is just the type of information inside it - for example, it cold be a **whole number**, a **fraction**, or even **text!**
- For now, we will just use the **"int"** (short for integer) typ
- Make an **int** variable called **tries** or **lives** outside of the functions (we don't want to re-define it every time the loop runs!) and set it to **3**. This makes it a global variable and you can use it in all of your functions.

Step 14

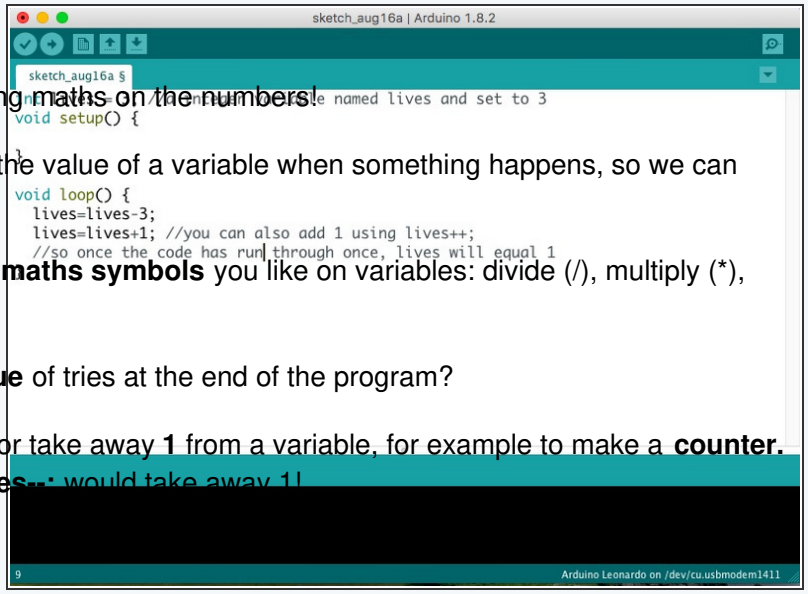
— Maths with variables (M)



- We can **change** variables as the program runs by doing maths on the numbers!
- On of the most useful things is to **increase/decrease** the value of a variable when something happens, so we can **count things.**
- **Build** the program in the picture - what will be the **value** of tries at the end of the program?

Step 15

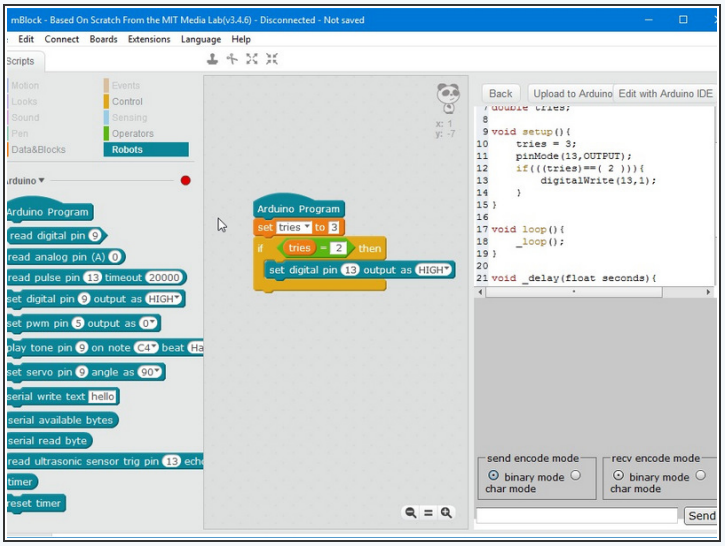
Step 15 — Maths with Variables (A)

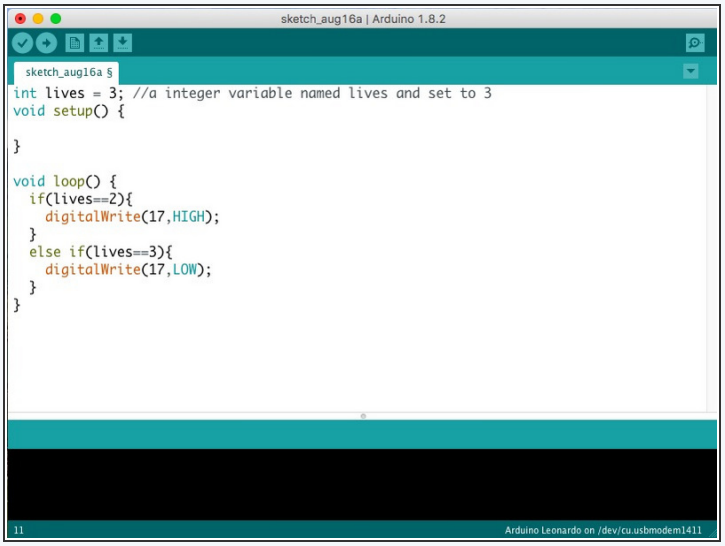


- We can **change** variables as the program runs by doing maths on the numbers!
- On of the most useful things is to **increase/decrease** the value of a variable when something happens, so we can **count things.**
- In the Arduino environment, you can use **any normal maths symbols** you like on variables: divide (/), multiply (*), add(+) and subtract(-)!
- **Build** the program in the picture - what will be the **value** of tries at the end of the program?

A quick **speed tip** - a very common thing to do is add or take away **1** from a variable, for example to make a **counter.** There is a shorthand for this - **tries++;** adds 1, and **tries--;** would take away 1!

Step 16 — IF statements and Variables





- You can also use variables in **IF statements!**
- Make our **test program** in the picture - will the LED turn on or not?

Step 17

Step 17 — Steady Hand Game with Variables

- Using everything you've learned about variables, can you make a steady hand game using a **variable to control the number of lives?**
- You will need to:
  - Set a **variable** equal to the number of lives
  - If the player hits the wire, **reduce** the number of lives by 1
  - **Check** if the player has run out of lives - if they have, do a **long buzz** (game over) and then **reset** the number of lives so they can **play again.**