AUSTRALIAN MATHS TRUST

Robotics

Workbook

digIT

critical thinking, creative solutions

SOLVE PROBLEMS. CREATE THE FUTURE.

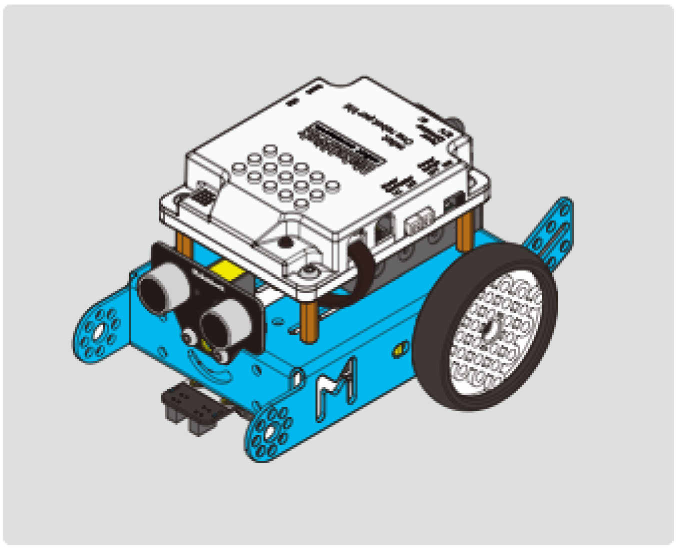
Robotics workbook

Lesson Overview:

- Introduction to Programming MakerBots
    - Difference between mBlock and Scratch
    - MakerBots what are they?
        - Sensors
        - Motors
        - The brain of the Bot
    - Programming the MakerBot
        - mBlock
            - how to upload a simple piece of code (move forward for 10 seconds)
- Designing Behaviour
    - What is a behaviour
    - Physical demonstration
        - Think/pair/share – navigate a person around a room using pseudo code.
- Functional Robotics
    - How far is the object?
    - Is the line on the left or the right?
    - Avoid the object
    - Putting that together
- Challenge: Quick Challenge
- Advanced Workshop: Major Challenge

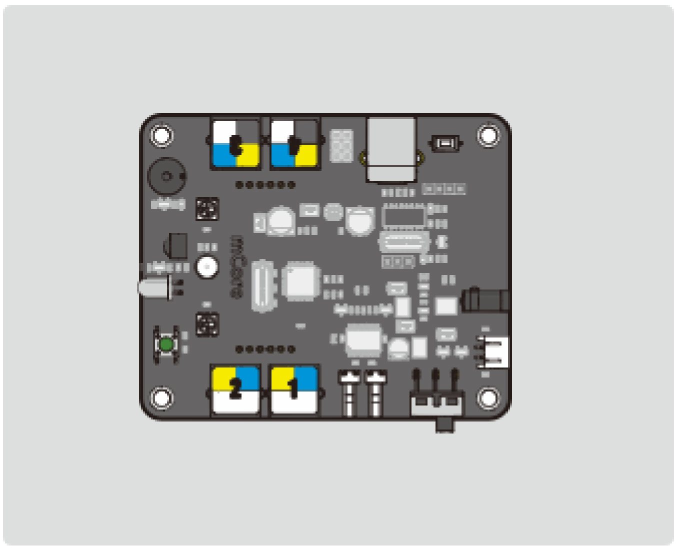
Introduction to Programming MakerBots

The different parts of our mBots:



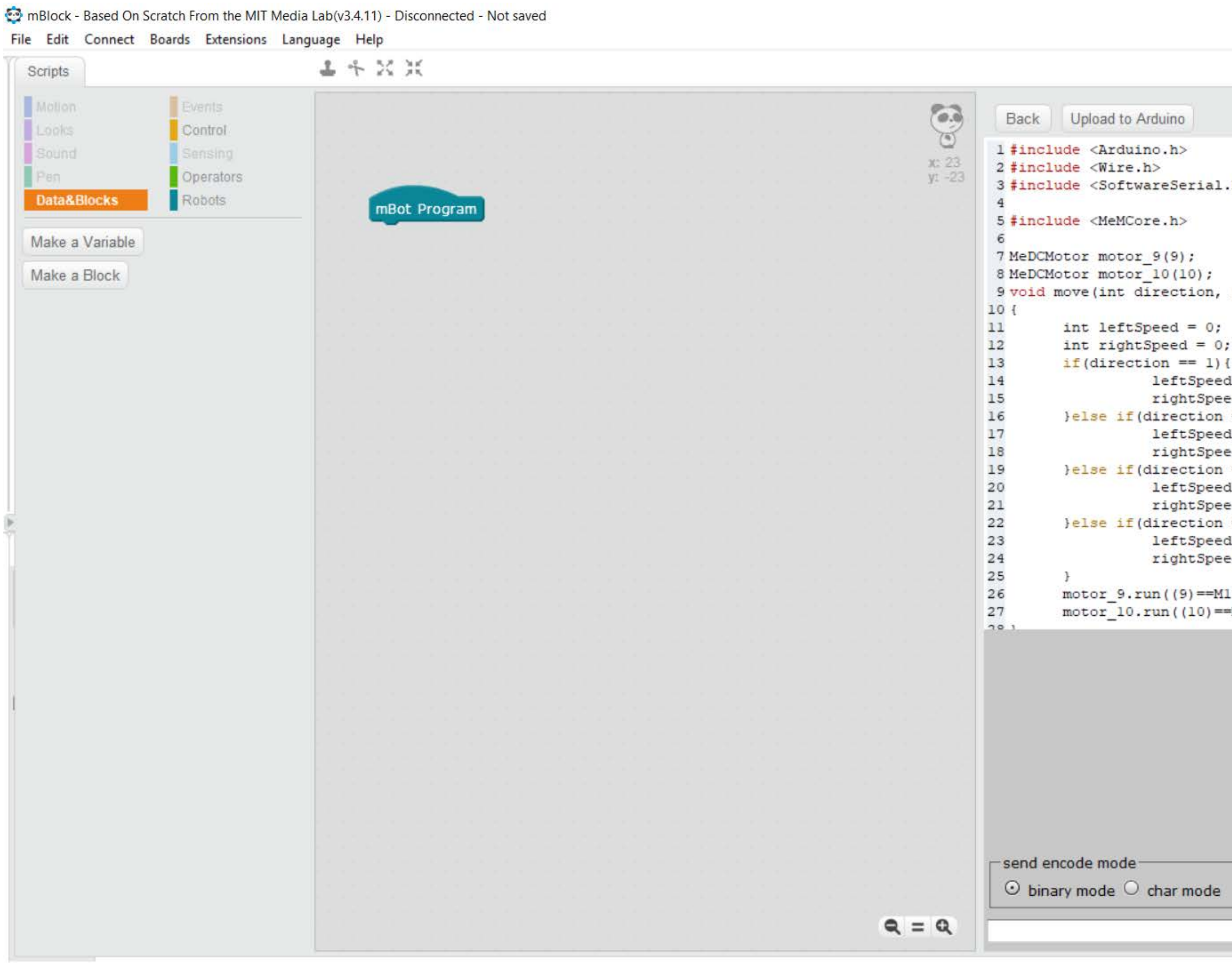
The mBot!

The brain of an mBot is a microcontroller. A microcontroller is a small computer that has no operating system.



Microcontroller

We load instructions from our Integrated Development Environment, mBlock to control different components.



```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.

#include <MeMCore.h>

MeDCMotor motor_9(9);
MeDCMotor motor_10(10);
void move(int direction,
{
        int leftSpeed = 0;
        int rightSpeed = 0;
        if(direction == 1){
                leftSpeed
                rightSpee
        }else if(direction
                leftSpeed
                rightSpee
        }else if(direction
                leftSpeed
                rightSpee
        }else if(direction
                leftSpeed
                rightSpee
        }
        motor_9.run((9)==M1
        motor_10.run((10)==
```

The mBlock IDE

You’ll notice that the mBlock IDE is pretty similar to Scratch. In fact, when you first boot into it you’ll see most of it looks exactly like scratch. The major difference is the Robot section that contains a listof scripts that control parts of the robot.

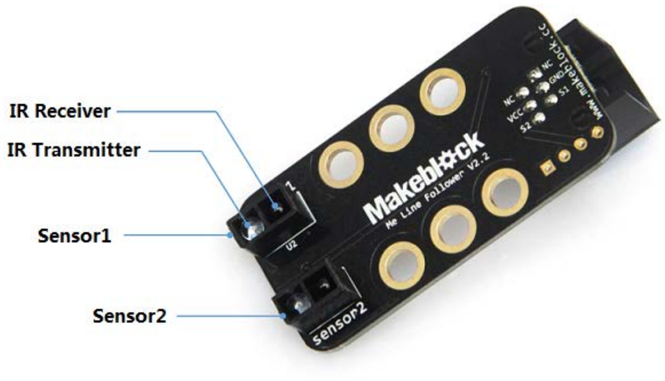
Sensors and components:

Ultrasonic distance sensor:

On the front of our robot is the Ultrasound distance sensor. It sends out sound at an ultrahigh frequency and then listens to how long it takes to come back. It's like a sonar ping from those WWII submarine movies.

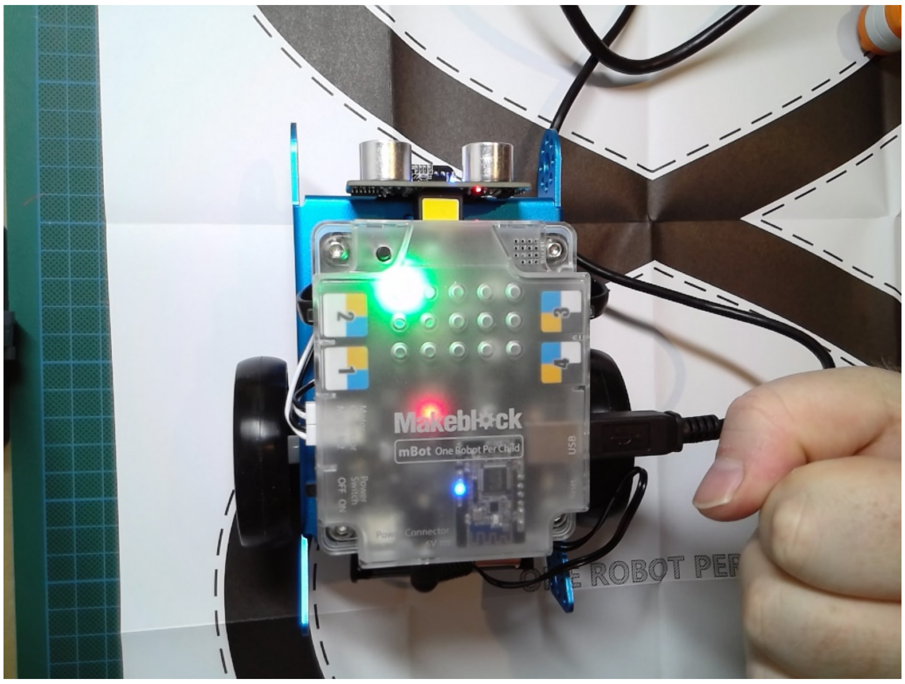
Line Sensor:

On the bottom of the mBot you'll see the line sensor. If you look closely you'll see the line sensor is split into two halves and each half has two components. The top component shoots Infrared light and the bottom component checks to see how "bright" the light is. Black absorbs more Infrared light, so if the line passes over a black object it will report black. However, you can also use this to find cliffs that you don't want to fall off too. This is because the intensity of light bouncing off of a distant object is less bright than from a close object so it looks "black".

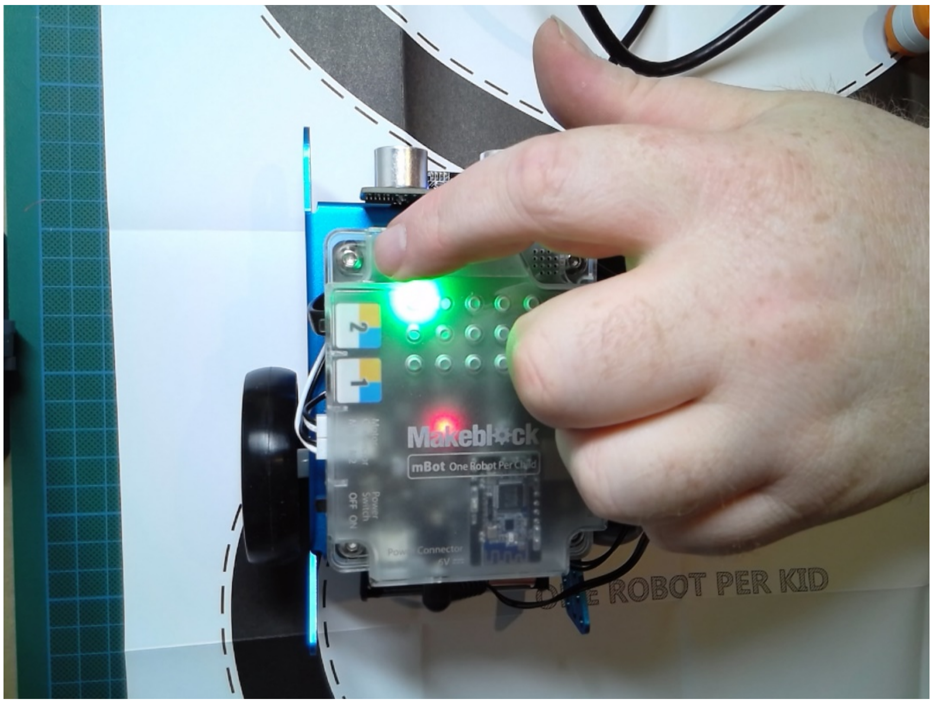


Push button:

There is a little push button on top of the mBot. We are going to use this to activate the mBot when we want it to go.



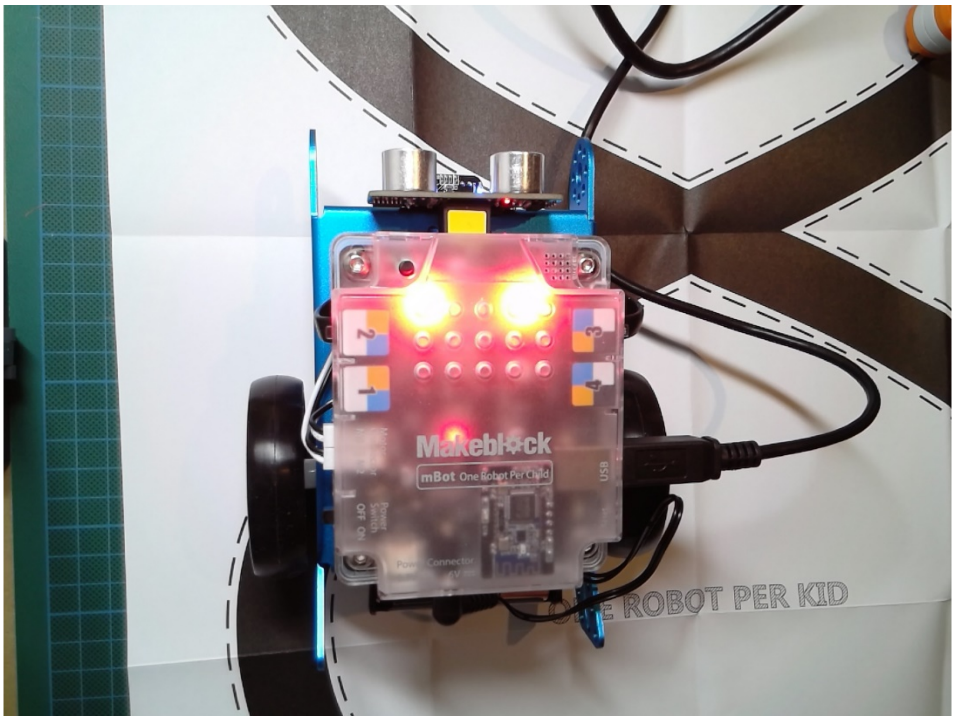
The button is that black dot right above the green light.



I'm covering it with my finger.

Built in lights:

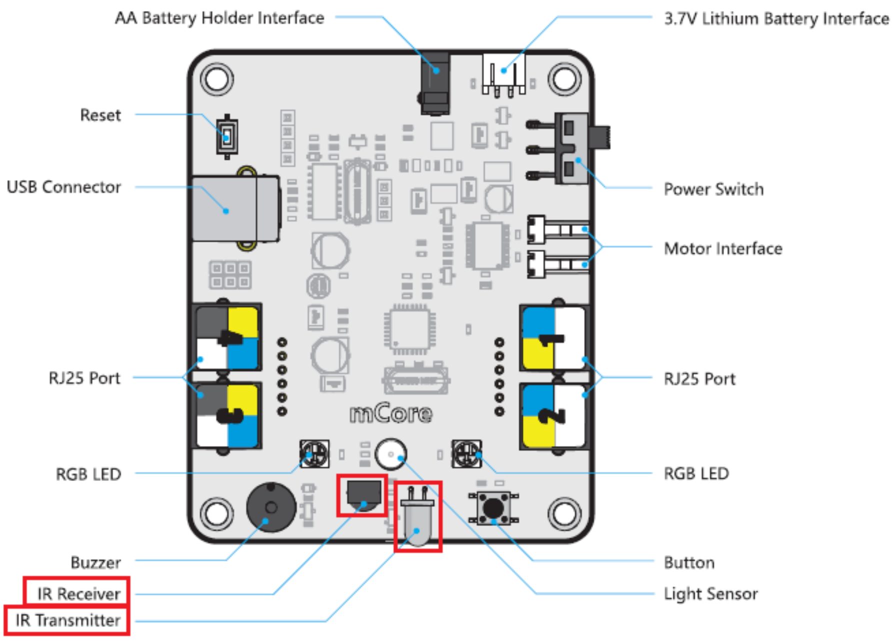
There are two RGB lights on top of the mBot.



It almost looks like the lights are eyes!

Built in annoying sound generator:

Yes, it's there. Please use sparingly.



Difference between mBlock and Scratch

So, a question you might be having is, “why a block based language and not Arduino C++?” Good question!

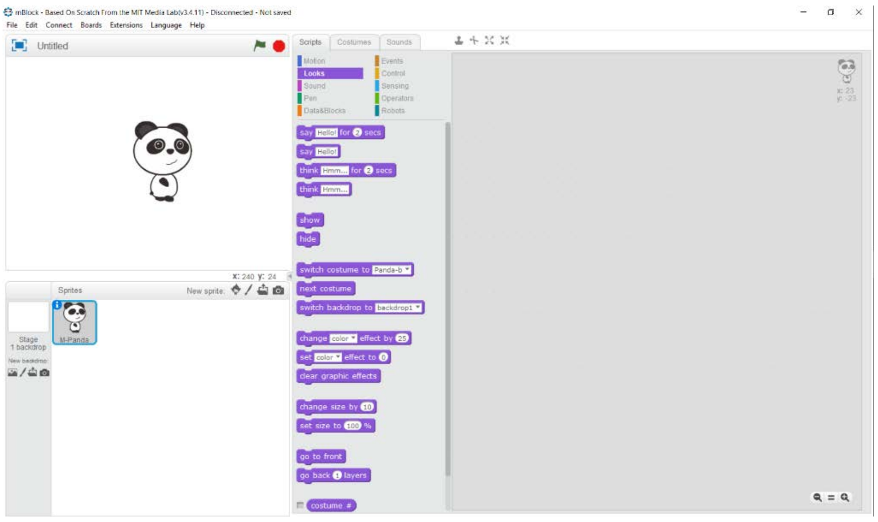
The answer comes down to ‘bang for our buck’. By focusing on block based languages we can get to the fun stuff quicker rather than later. However, with mBlock we can also see the Arduino C++ code being generated. In fact, for any pros in the room you can even code straight to the system using the Arduino IDE.

Let’s move to a practical demonstration of the system. Let’s make it so we can change the colour of the top lights as objects get closer to the mBot.

Let’s start:

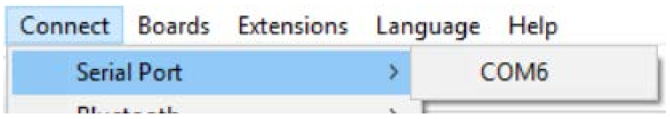
Open up mBlocks.

You might be surprised to find that it looks a lot like scratch now. That’s okay, we’ll be changing it in just a minute for work with robots.

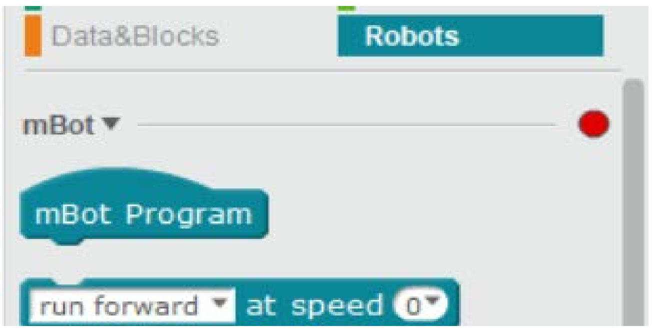


Connect your mBot to your computer using the USB connection and turn your mBot on.

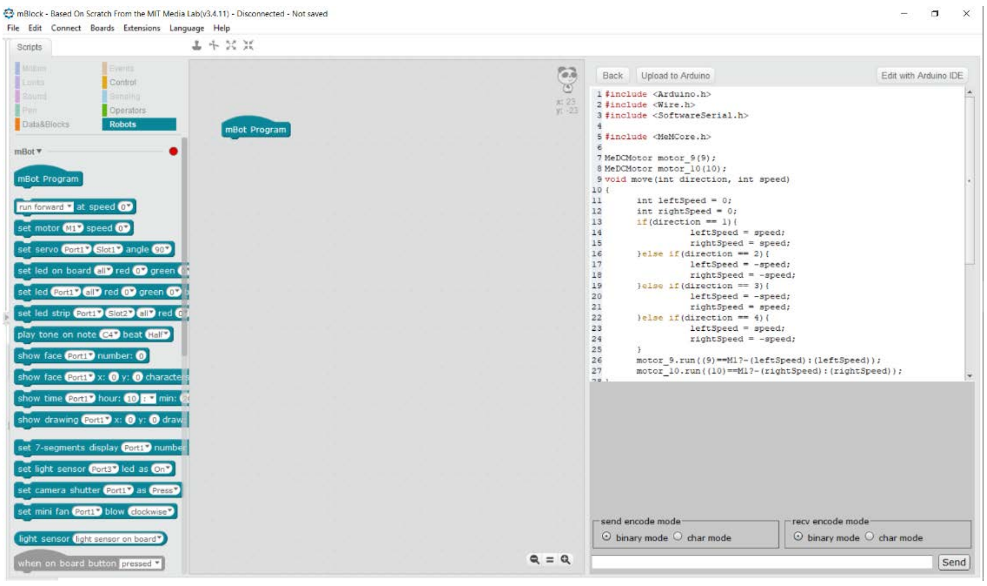
Look along the menu items at the top of the IDE. You’ll see one that says “Connect” press it. Then, press Serial Port, and then whatever communication port your computer selects. My computer selected COM 6 but yours will likely be different.



Let’s start by getting our IDE into the right spot.
Click on the “Robots” in the script section.



Now, select the mBot Program hat and drag it into the main screen.



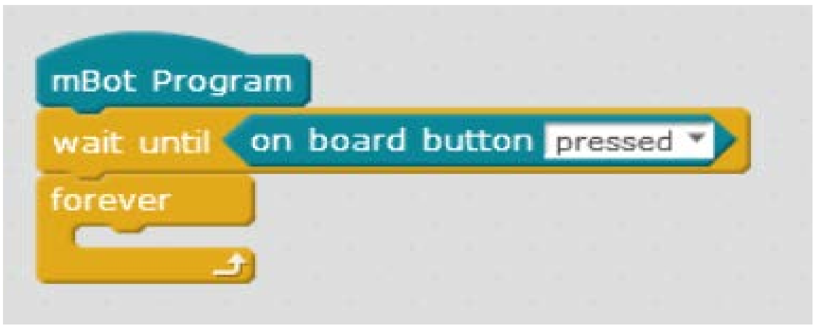
```
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>

#include <MeMCore.h>

MeDCMotor motor_9(9);
MeDCMotor motor_10(10);
void move(int direction, int speed)
{
      int leftSpeed = 0;
      int rightSpeed = 0;
      if(direction == 1){
            leftSpeed = speed;
            rightSpeed = speed;
      }else if(direction == 2){
            leftSpeed = -speed;
            rightSpeed = -speed;
      }else if(direction == 3){
            leftSpeed = -speed;
            rightSpeed = speed;
      }else if(direction == 4){
            leftSpeed = speed;
            rightSpeed = -speed;
      }
      motor_9.run((9)==M1?-(leftSpeed):(leftSpeed));
      motor_10.run((10)==M1?-(rightSpeed):(rightSpeed));
```

This is the view we will be looking at a lot. The code on the right is the actual Arduino code that mBlocks creates.

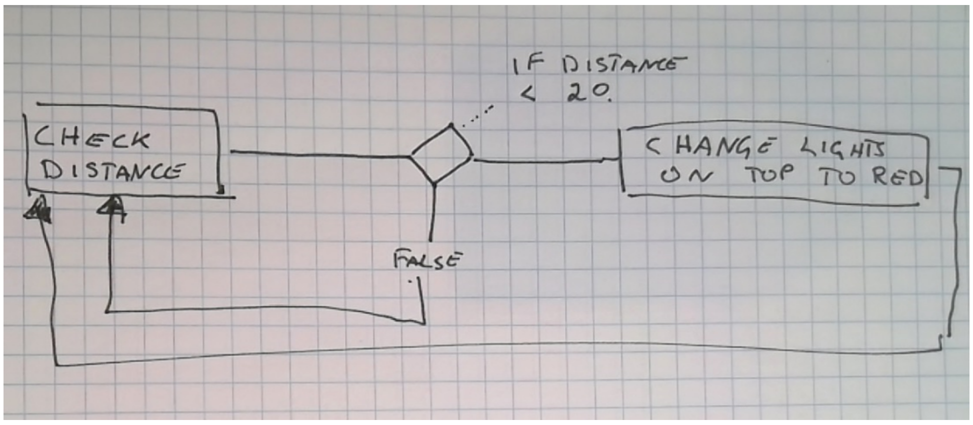
Now, let's get our base template code ready to go. We're going to be using it a few times.



Don't run it yet, but let's just look at it for a second. What does it do? Well, basically what it does is when the code is uploaded to the mBot it will start running. This is awkward if the robot just runs off when you aren't ready, so we wait for the button to start.

Let's make a simple program that changes the mBot's lights if something gets too close. Let's say IF something gets less than 20cm THEN turn red LEDs on.

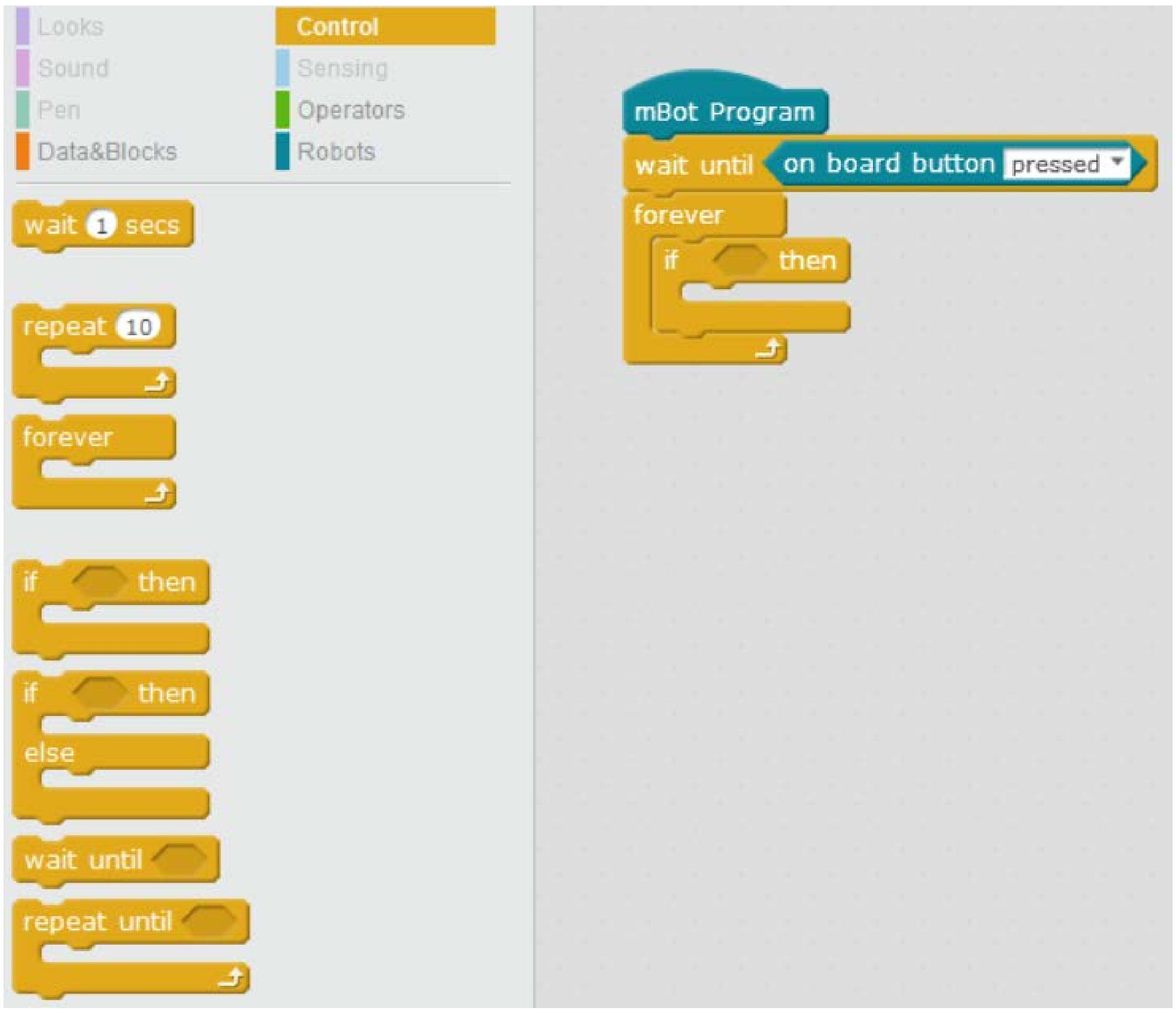
What does that logic look like?



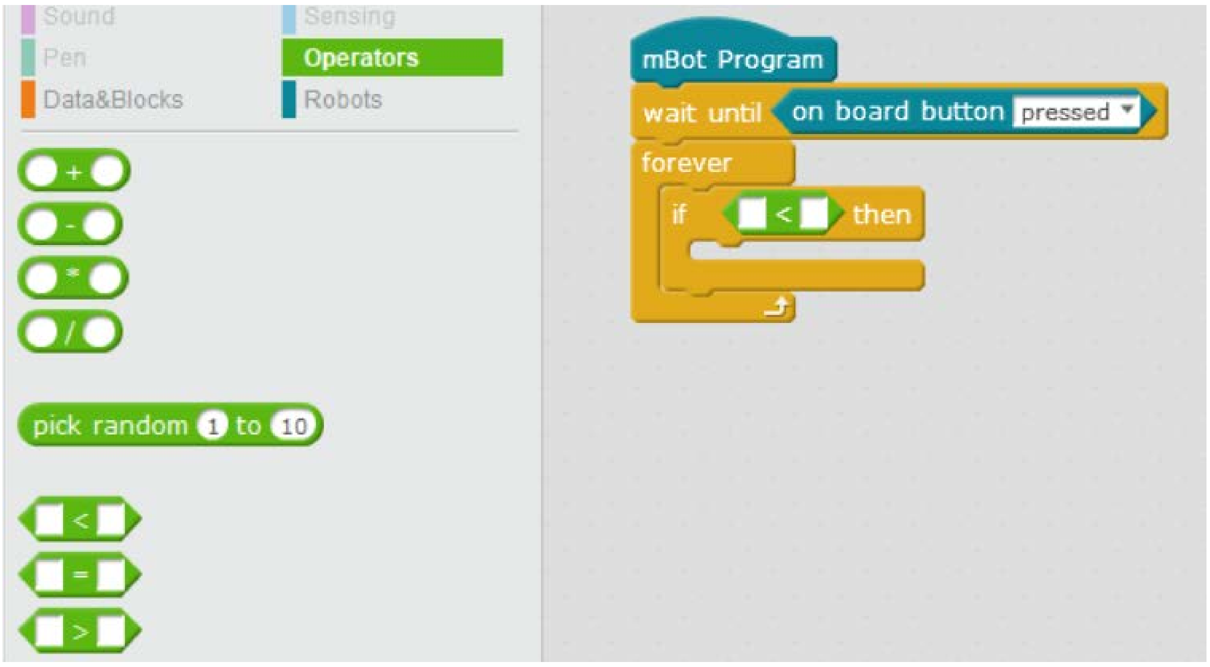
We are going to do some command that checks the distance and if that distance is less than 20 we are going to change the light on top to red. Otherwise we won't do anything.

Right, let's do it.

The first thing that comes to mind is that the value of the ultrasound sensor and if there is a distance less than 20cm is a CONDITION. So, we need to use a conditional, which are normally found in CONTROL, and it's going to be an IF statement.

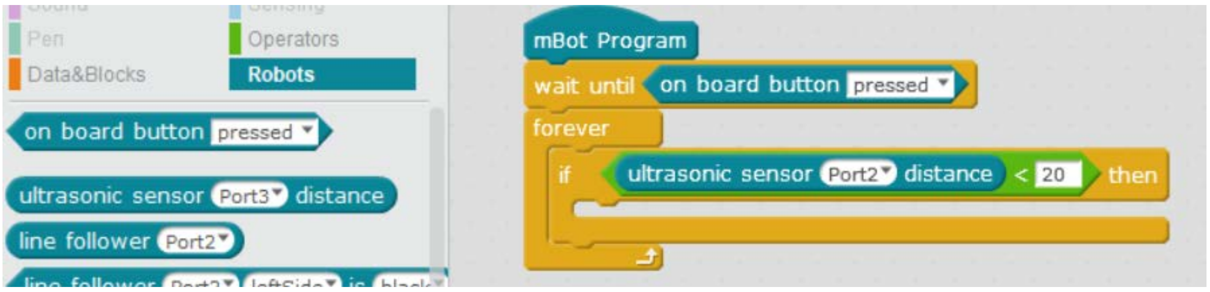


So, we have an IF condition. We need to test to see if the ultrasound sensor is less than 20cm from the object. I'm going to go with finding the less than operator.



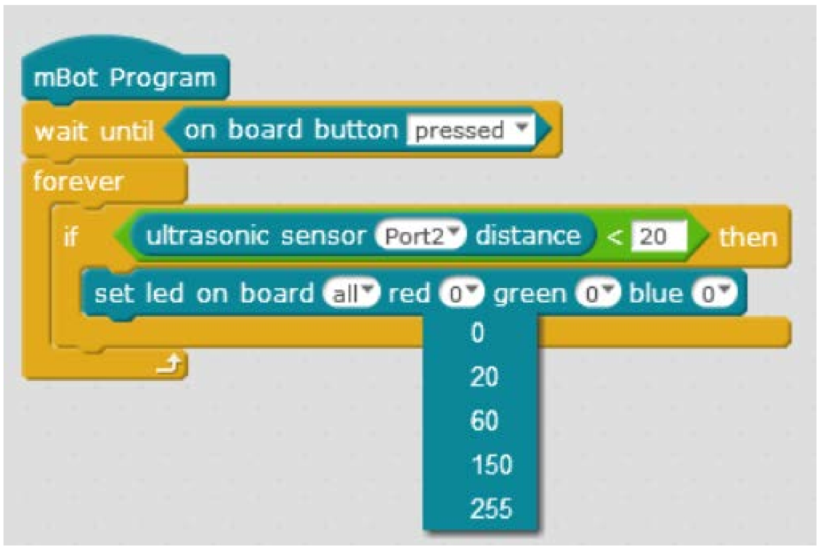
Great! We're almost there!

The next step is back to ROBOTS and find where the ultrasound sensor is. Once there we can also fill the distance we are measuring (20cm).


Pen
Operators
Data&Blocks
Robots
on board button pressed
ultrasonic sensor Port3 distance
line follower Port2
mBot Program
wait until on board button pressed
forever
if ultrasonic sensor Port2 distance < 20 then

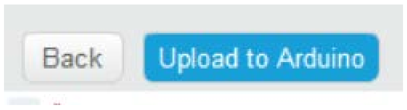
Now, here's something that is tricky. My mBot has the ultrasonic sensor on port 2 but yours might not have. You are going to need to follow the RJ-12 cable from the back of the ultrasound sensor to your board. Luckily, there is a little number on top that shows you what the port number is.

Anyway, last part. We have to change the LEDs on the top of the robot to RED. We'll luckily we're already in ROBOTS, so we can just find the LED on board block.


```
mBot Program
wait until < on board button [pressed ▾] >
forever
    if < (ultrasonic sensor (Port2 ▾) distance) < (20) > then
        set led on board (all ▾) red (0 ▾) green (0 ▾) blue (0 ▾)
```

To change the colour, you need to set the value. The value ranges from 0 to 255 with 0 being off and 255 being on at full power. Let's set it to 255.

Right! Let's do this! Click the upload button on the right side of your IDE (right above the Arduino code)



If it doesn't upload make sure your bot is on, and connected to your computer.

Now test it!

Oh, wait, there's a bug! The light doesn't turn off. How can we fix this? What's the new Behaviour that we want?

Scenario 1: The mBot isn't close to anything and so it makes sure that it's lights are turned off by turning them off.

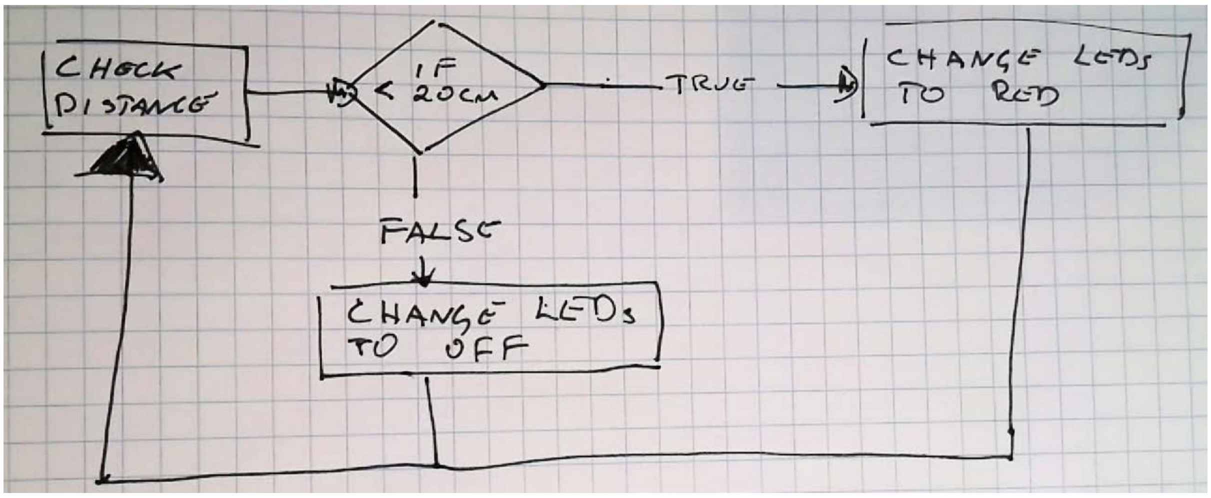
Scenario 2: The mBot is close to something so it turns it's red LEDs on to max.

IF distance < 20cm

THEN turn red LED to 255

ELSE turn all LEDs to 0.

So the logical diagram of this looks almost the same.



Now, quickly, while nobody is looking patch my buggy code for me!

Robotics Workshop 2

Behaviour that defines Development

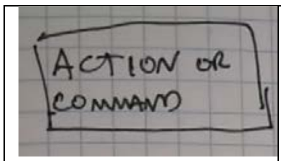
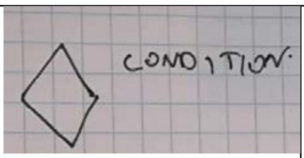
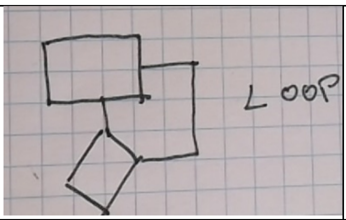
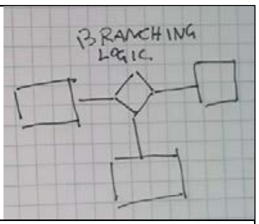
This workshop uses the foundation knowledge of Behaviour defined development to help us design more complicated interactions that we find in physical computing.

Behaviour defined development gets more complex later but at its most basic use it works like this:

- We define a behaviour as a story
- We define a test that shows if our behaviour is complete
- We define the logic of our program
- THEN we build our program.

There are no formal requirements for exactly how these stories must be written down but as a starting point I'd like to recommend that we use simple syntax of programming: IF, ELSE, THEN, WHILE

Visual Logic is defined as such:

			
This is where something happens. You call a function or check you check a sensor	This is where you test something.	Loops are where you do things over and over again.	Branching logic can be demonstrated.

Physical Demonstration

Think, inside/outside circle, demonstrate.

Scenario: Driving Ms/Mr Literal.

We have been tasked with driving a person around who can only take literal instructions. That is to say: they will do exactly what you say and only what you say, no more, no less.

You have 5 minutes to come up with a behaviour story and logic diagram that describes how somebody can walk around the room. The only tool they have for measuring distance is their hand and they only understand simple commands (like walk forward, turn left, and etc.). No compound movements like walk on a curve and etc.). Remember, there are going to be a bunch of scenarios to navigate the course so be liberal with the scenarios.

Form into a big circle and split into AB groups. The B group becomes the inside circle and the A group is the outside. The A person has 2 minutes to discuss their behaviour documentation and to get feedback from B. Then we swap.

After 4 minutes Both A and B move one space to the left (this should make a gap of 2 people between you). We repeat this a total of 4 times.

Test:

Form into groups of three.

Person 1: Is the robot – they must close their eyes and listen to the “Computer”

Person 2: The computer. They tell the “Robot” literally what the Program says to do.

Person 3: The Program. They interpret the script as best they can and tell the Program exactly what it says.

Advanced Robotics workshop

Refresher on programming with blocks.

We're going to quickly make a program that makes the panda move forward until it hits an edge. Then it will turn around, and go back. This will continue until we stop the program.

IF: the panda is touching the edge of the stage

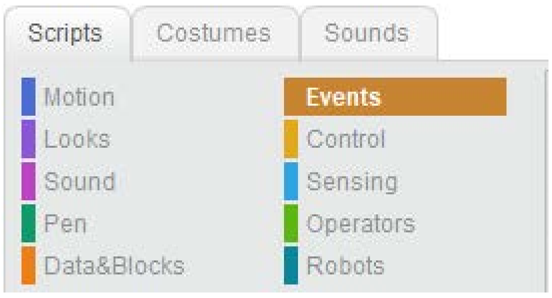
THEN: Turn 180 degrees

ELSE: step 10 pixles.

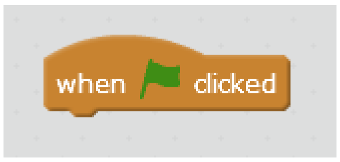
(note: You'll see a bug in my code where I only use 90 degrees ☹)

Steps:

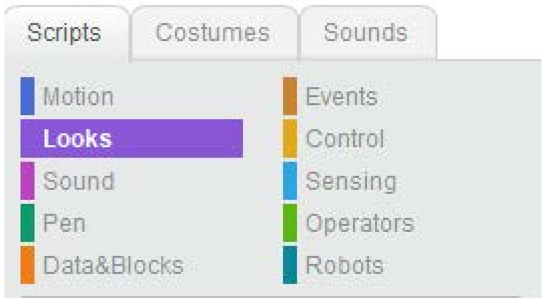
First we need to tell Scratch to do something when we click the flag. This is a EVENTS feature so we can find it in Events.



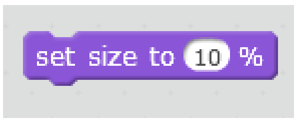
Now, click and drag the when Flag clicked hat onto your stage this will tell your program to use the blocks connected to this hat when the Flag is clicked.



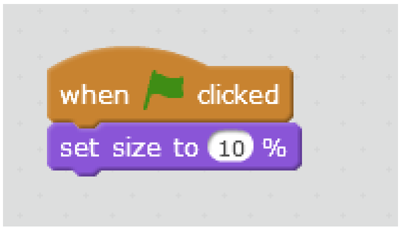
Now, just for aesthetics I'm going to change the size of the panda. I think he should be about 1/10th of the size he is now but you can choose your own value. Changing his size is a LOOK so we can find it in Looks.



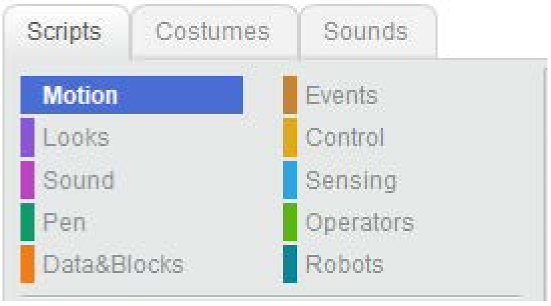
And it is called Set Size:



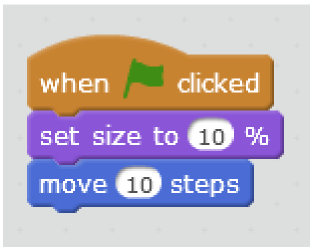
And slap it right under our When Flag Clicked hat



So, we want to move the sprite. Move is a MOTION so we can find it, you guessed it, Motions.



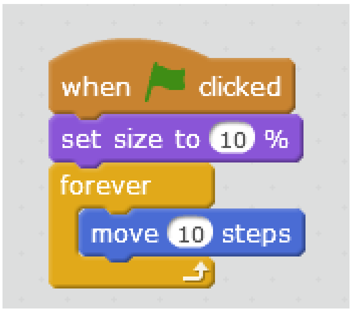
Find the move [10] steps block and connect it under set size.



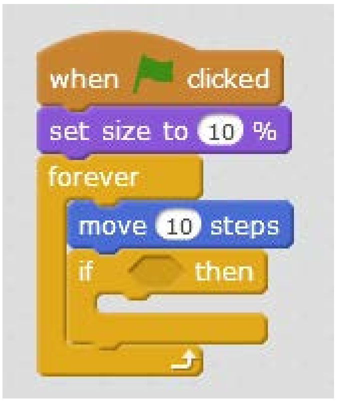
Now if you click the flag a few times, you'll see your panda move across the screen.

The next thing we're going to want to do is repeat that. We are going to want to keep walking across the screen. Obviously, this isn't complete, it's just a first step, we can handle the collision with the edge next.

Repeating in a Control, so we will click on CONTROL and because this repeat goes forever we'll drag the forever block out and make sure it wraps around the move.

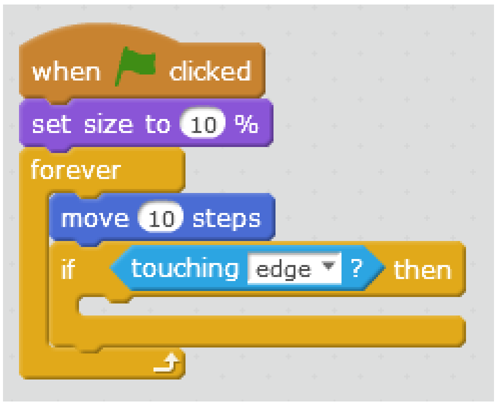


We need to see if we are touching the edge, but before that we need to tell the program to make a decision about if we are touching the edge or not. Decisions on conditions are in CONTROL. Click the If block and drag it under move 10 steps



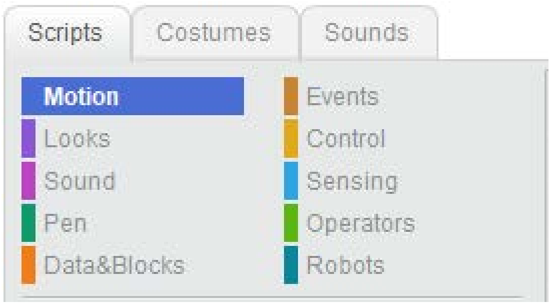
```
when [flag] clicked
set size to (10) %
forever
    move (10) steps
    if < > then
    end
end
```

Alright, now we can make a decision about something and that something is if we are touching the edge of the screen. Touching is a sense, so let’s click on SENSING and choose and slot it into our IF statement. We need to change the thing we are checking on our touch to ‘edge’.

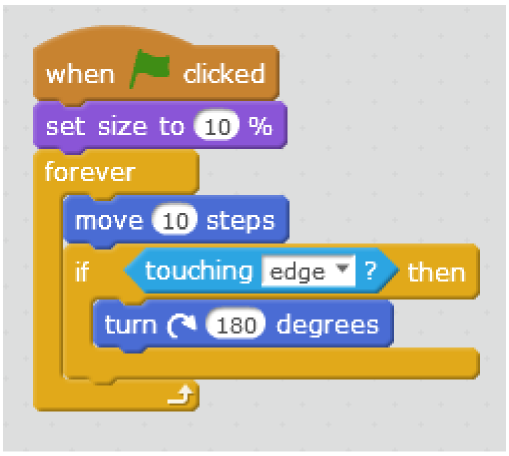


```
when [flag] clicked
set size to (10) %
forever
    move (10) steps
    if <touching [edge ▾] ?> then
    end
end
```

Almost done! Now, all we need to do is turn around 180 degrees.



Well, turning is a movement so let’s look in MOTION and click turn __ degrees. Except, we’re going to change the value in there to 180. Slot that in between our IF statement.



```
when 🏴 clicked
set size to (10) %
forever
    move (10) steps
    if <touching [edge ▾] ?> then
        turn ↻ (180) degrees
```