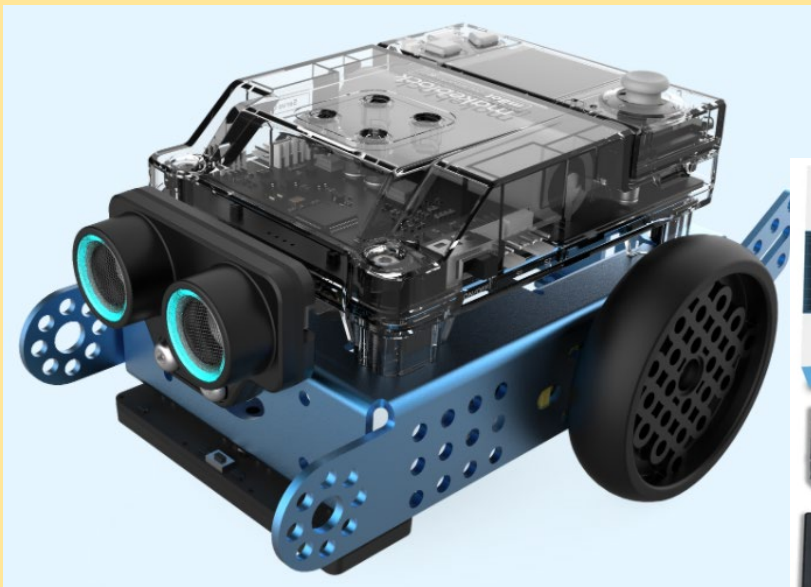


Digital Technology

MakeBlock mBot2 / CyberPi Python Coding



Version 4
March 2023

Barry Butler
bbutl58@eq.edu.au



Content

Section	Content
A	The mBot2 Vehicle
B	Introduction and Setup
C	Our First Program – Hello
D	Turn on the Lights
E	Information Display
F	Ringtones and Sound Bites
G	Run the Motors
H	Avoid or Seek
I	Detect and Follow a Simple Line
J	Grabbers and Other Mechanics
K	SumoBot Competition
L	Line Follower with Intersections
M	Mecanum Wheels
N	Robotic Arm with 4 DOF
O	Keep the Light Just Right
P	Rollover Warning
Q	Connect Other Sensors
Appendix 1	CyberPi Extras
Appendix 2	RoboRave a-Maze-ing Track Details

Documentation

<https://www.yuque.com/makeblock-help-center-en/mcode/mblock-python>

CyberPi Python <https://www.yuque.com/makeblock-help-center-en/mcode/cyberpi-api>
(including Pocket Shield, mBot2 Shield and mBuild Modules)

Firmware Update

To update the CyberPi firmware:

1. Open the online ide at <https://ide.mblock.cc/#/>
2. Click on **Devices** and add the CyberPi device to the list, if it is not there already
3. Click **Connect** and connect the CyberPi (download and install the device driver, if asked)
4. Click on **Settings** and select Firmware Update

A. The mBot2 Vehicle

Documentation

MBot2 Introduction

<https://education.makeblock.com/help/cyberpi-series/cyberpi-series-cyberpi-series-packages-and-extensions/mbot2-introduction/>

Operational Guide

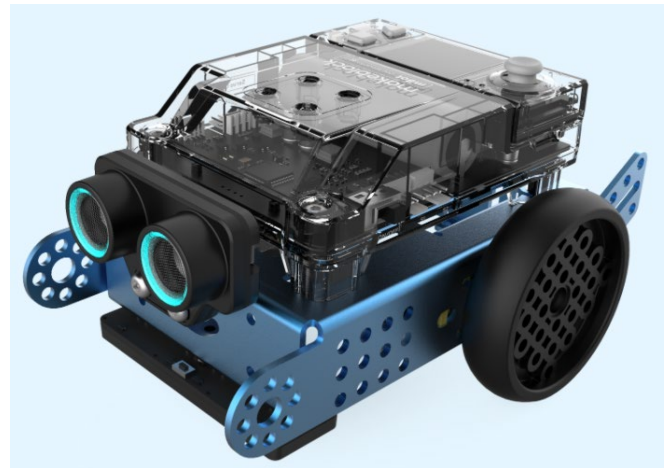
<https://education.makeblock.com/help/mbot2-start/>

Python Reference

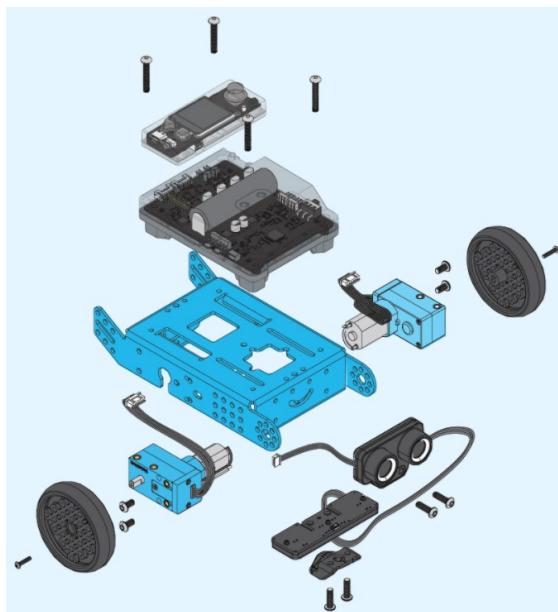
<https://education.makeblock.com/help/mblock-python-editor-python-api-documentation-for-cyberpi/>

<https://education.makeblock.com/help/mblock-python-editor-apis-for-extension-boards>

<https://education.makeblock.com/help/mblock-python-editor-apis-for-mbuild-modules/>

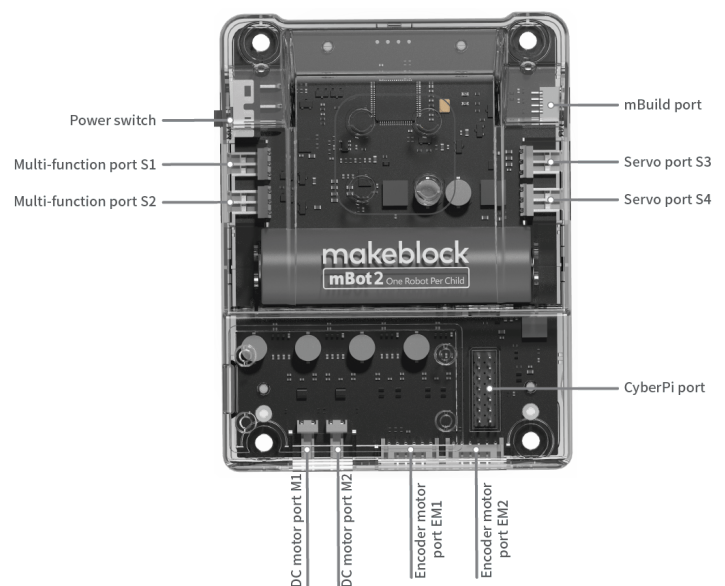


The Build



The Connections

(Ultrasonic into the mBuild port, motors to EM1/EM2)



**The Power Switch must be turned on
before the you can upload code**

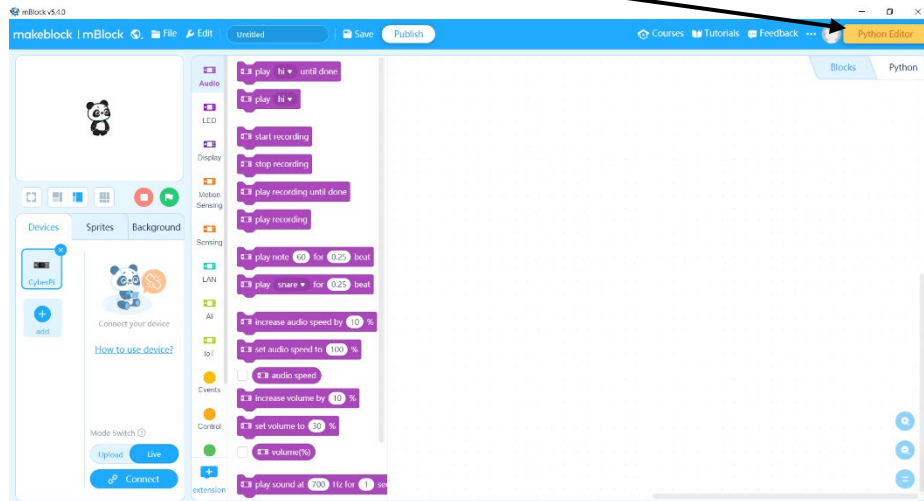
B. Introduction and Setup

Download and Install the Software

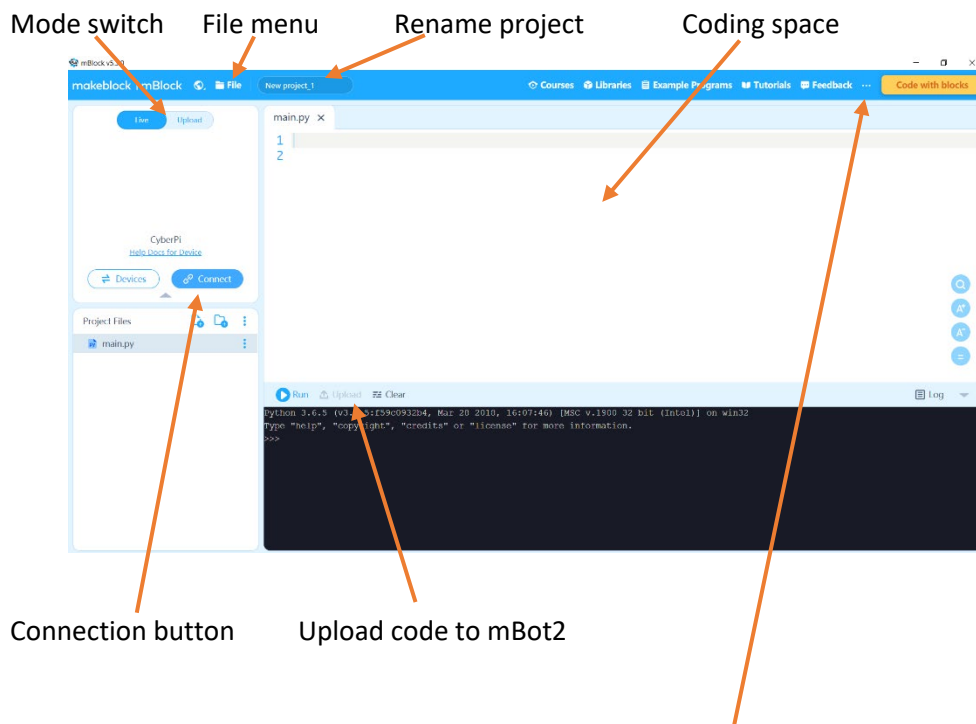
Download and install the mBlock Windows or Mac software from <https://mblock.makeblock.com/en-us/download/>

(The PC software seems to be more stable than the web version located at <https://python.mblock.cc/>)

1. Run the software and select the Python Editor.



The Python Editor program will open. The block editor will stay open, but you can close it at any time.

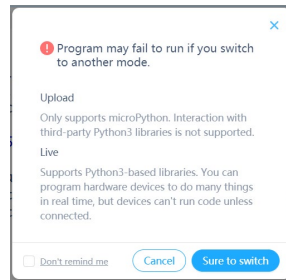



To have the Python Editor open by default, click ... and choose **Set as default editor**

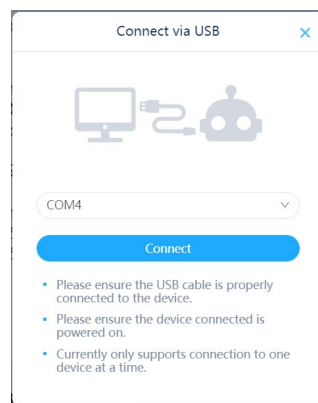
2. TURN ON THE MBOT2 USING THE SWITCH ON THE SIDE

The lights on both the ultrasonic sensor and the line follower sensor should turn on. If they are don't, the wiring is incorrect or unplugged, and needs to be fixed.

3. Select **Upload** mode. If a message appears, tick “Don’t remind me” and then click *Sure to switch*.



4. Plug the mBot2 into a USB port and click the **Connect** button  **Connect**.



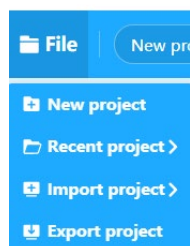
Find Your Port

You can easily find your device by first leaving your mBot **unplugged**. Click *Connect* and look at the list of USB ports.

Close the connect window, then **plug in** your mBot2. Click *Connect* again and look for the port that has just been added.

Select your USB port from the list and click **Connect**.

5. Click on the **File** menu and select **New Project**.



6. Start coding

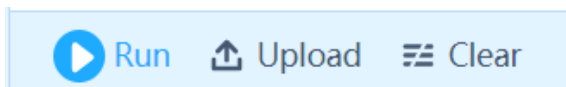
C. Our First Program – Hello

Our first program will write 'hello' on the console, say it on the audio speaker and turn all LED's to green for 2 seconds.

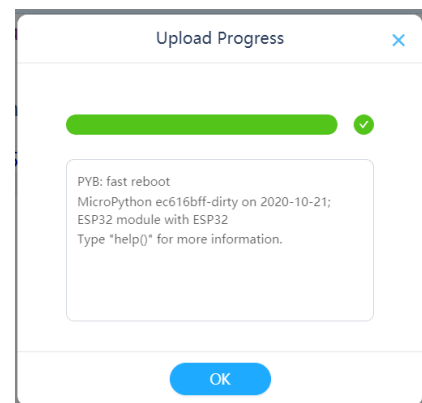
```
import cyberpi as cpi
import time

cpi.console.print("hello")
cpi.audio.play('hello')
cpi.led.on(0,255,0)           #red, green, blue values from 0 to 255
time.sleep(2)                #time delay in seconds
cpi.led.off()
cpi.console.clear()
```

Click the **Upload** button to send your code to the mBot2.



The code will start executing immediately it is uploaded.



Unsuccessful Upload

If the upload is unsuccessful check three things:

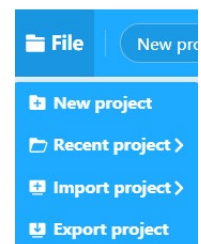
1. The **mBot2 is turned on** (the power switch on the left side).
2. The cable is plugged in and a **connection established** (see section B4).

Save the Project and Upload to the CyberPi

Save the project to your computer by clicking on the **File** menu and choosing **Export project**.

It is a good idea to create a folder to contain all your projects.

Make sure you type in a descriptive name for your file.



Coding Errors and Feedback from the CyberPi

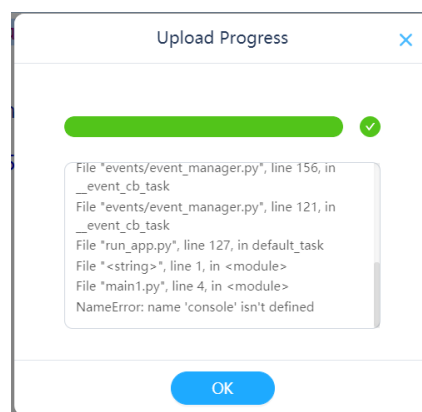
When you write code, errors show up with an explanation mark symbol.

In the example shown here there are a couple of errors:

- the statement in line 1 is ***import cyberpi as cpu*** rather than ***import cyberpi as cpi*** causing errors in all the other lines.
- line 4 is missing ***cpi***.

```
1 import cyberpi as cpu
2 import time
3
4 console.print("hello")
5 cpi.audio.play('hello')
6 cpi.led.on(0,255,0)
7 time.sleep(2)
8 cpi.led.off()
9
```

If you were to upload this code it would not run and the upload window will show you the first error. Scroll to the bottom of the text to see the error message.

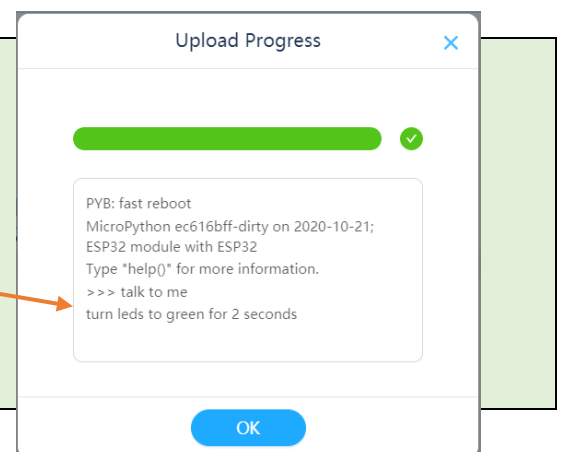


Program Feedback

You can also give yourself feedback in the code you write by using the **print()** function. This is *different* to the `cpi.console.print()` function. Try this:

```
import cyberpi as cpi
import time

cpi.console.print("hello")
print('talk to me')
cpi.audio.play('hello')
print('turn leds to green for 2 seconds')
cpi.led.on(0,255,0)
time.sleep(2)
cpi.led.off()
cpi.console.clear()
```



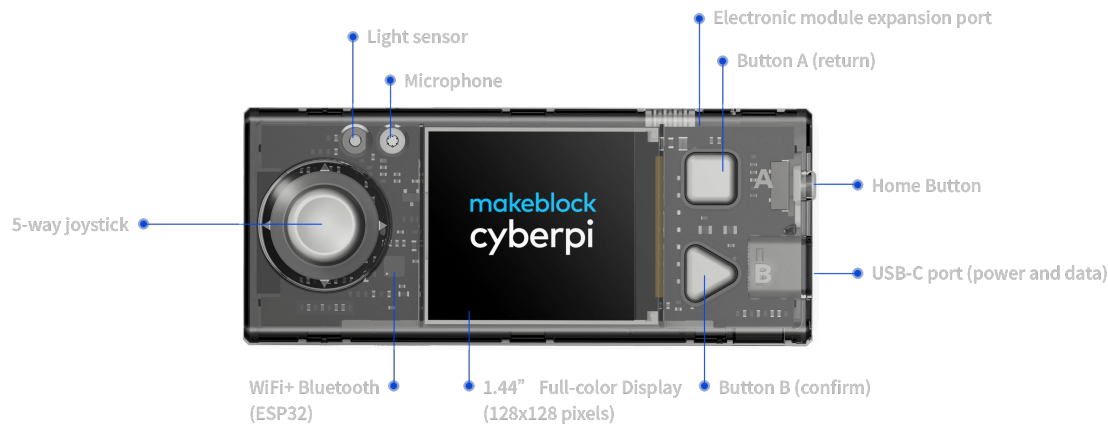
Comments and turning on/off code statements

Put a **#** in front of any line to create comments or to turn code statements into comments so they are not executed.

D. Turn on the Lights

The mBot2 is controlled by a module called **cyberpi**. This has a joystick, a home button and two push buttons (A and B). We can use the joystick and buttons in our code.

The **home button** is used to reset the program to run again from the start. After pressing the home button, press down on the joystick to select the first option that appears.



Instead of the code running automatically when it is uploaded, let's turn on the lights when we press button A. To do this we need to put the code into a forever loop and use an **if-else** statement.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.led.on(0,255,0)
    if cpi.controller.is_press('b'):
        cpi.led.off()
    time.sleep(0.1)
```

We can also turn on and off individual led's using the **cpi.led.on** command or **cpi.led.show**.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.led.on(0,255,0)
        time.sleep(0.3)
        cpi.led.on(0,0,255, id=3)
    if cpi.controller.is_press('b'):
        cpi.led.show('r o y g c')
    time.sleep(0.1)
```

The colour options for **cpi.led.show** are:

red r	orange o	yellow y	green g	cyan c
blue b	purple p	white w	black k	

Try different combinations of lights and make your own patterns. Use the **time.sleep** command to create delays between lights.

Turn on Lights using the Joystick

Add the code below to use the joystick to change colours. We use an **elif** statement because only one option can occur at one time. Before the forever loop we can set the led brightness.

```
import cyberpi as cpi
import time

cpi.led.set_bri(50)
while True:
    if cpi.controller.is_press('up'):
        cpi.led.on(0,255,0)
    elif cpi.controller.is_press('down'):
        cpi.led.on(255,0)
    elif cpi.controller.is_press('left'):
        cpi.led.on(0,0,255)
    elif cpi.controller.is_press('right'):
        cpi.led.on(0,255,255)
    time.sleep(0.1)
```

Continue exploring different light patterns for each joystick direction.

Shake It or Tilt It

We can use the motion sensor (accelerometer) to turn on the lights using a shake or tilt.

```
import cyberpi as cpi
import time

cpi.led.set_bri(50)
while True:
    if cpi.is_shake():
        cpi.led.on(0,255,0)

    elif cpi.is_tiltforward():
        cpi.led.on(255,0,0)
    elif cpi.is_tiltback():
        cpi.led.on(255,255,0)

    elif cpi.is_tiltleft():
        cpi.led.on(0,0,255)
    elif cpi.is_tiltright():
        cpi.led.on(0,255,255)
    time.sleep(0.1)
```



Yell (or clap) at it

```
import cyberpi as cpi
import time

cpi.led.set_bri(50)
while True:
    level = cpi.get_loudness("maximum")
    if level > 50:
        cpi.led.on(0,255,0)
    else:
        cpi.led.on(0,0,0)
    time.sleep(0.1)
```

Sound level values range from 0 to 100. Change **if level > 50** to a different value and see what happens.

Finding and Correcting Errors

By now you will have probably encountered quite a few errors or bugs in your code. You will often see messages saying that you have a **syntax error** or **indentation error**.



Common Bugs

1. Spelling mistakes
2. Incorrect case – Upper and lowercase are different in Python
3. True and False have capital first letter
4. Missing colon (:) at the end of **if**, **def** and **while** statements
5. Incorrect indentation (use the tab key to indent statements)

Make sure you keep the upload window open so you catch **runtime errors** – the errors that only appear when you run the code.

E. Information Display

Having a built-in display is really useful for displaying instructions and information from sensors. Writing to a display is relatively slow, so when we have tested the ultrasonic sensor or line tracker sensor we will probably turn off the display so we can respond quickly to the new information.

Text

Let's display text when pressing the two switches. The ***println()*** function writes each message on a new line.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.console.println('button a')
    elif cpi.controller.is_press('b'):
        cpi.console.println('button b')
    time.sleep(0.1)
```

Numbers

We can only write text (called strings) to the display. To write numbers on the display we have to convert the number to text. This code writes a sequence of numbers.

```
import cyberpi as cpi
import time

i = 0
while True:
    msg = str(i)
    cpi.console.println(msg)
    i += 1
    time.sleep(0.5)
```

Loopy Numbers

Loops are one of the fundamental coding structures. Let's create a loop to display a sequence of ten numbers between 0 and 9.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        for i in range(10):
            cpi.console.print(str(i) + ',')
            time.sleep(0.5)
```

Notice two things:

1. The ***print()*** function is used rather than `println()`. The numbers appear on the same line each time.
2. Two bits of text are joined using the `+` operator.

We can change the start and end values of the loop. For example, loop between 3 and 7. Note that the end number in the code must be one greater than the number we want to finish on.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        for i in range(3, 8, 1):
            cpi.console.print(str(i) + ',')
    elif cpi.controller.is_press('b'):
        cpi.console.clear()
    time.sleep(0.1)
```

This loop statement has three numbers – the start value, the end value, and the step value.

We are also using button B to clear the screen.

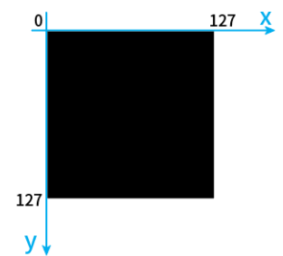
Change each of the three values and see what happens.

Writing to a Specific Display Position

We can write messages at particular positions on the screen by either using x,y coordinates or a position string.

```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.display.show_label('Msg at (10,10)', 12, 10, 10)
    elif cpi.controller.is_press('b'):
        cpi.display.show_label('Bottom left', 12, 'bottom_left')
    time.sleep(0.1)
```



The position strings are

- top_mid: in the upper center
- top_left: in the upper left
- top_right: in the upper right
- center: in the center
- mid_left: in the middle left
- mid_right: in the middle right
- bottom_mid: in the lower center
- bottom_left: in the lower left
- bottom_right: in the lower right

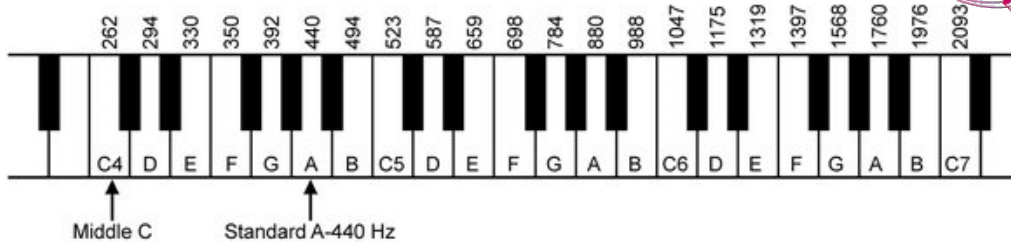
Unfortunately, the command clears the screen before displaying the message so we cannot have more than one message on the screen. To show more information, use a combination of `println()` and `clear()` commands.

F. Ringtones and Sound Bites



Phone Ringtones

Ringtones are just a series of notes played for a certain duration.



```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.audio.play_music(60,0.5)
        cpi.audio.play_music(60,0.5)
        cpi.audio.play_music(67,0.5)
        cpi.audio.play_music(67,0.5)
        cpi.audio.play_music(69,0.5)
        cpi.audio.play_music(69,0.5)
        cpi.audio.play_music(67,0.5)
    time.sleep(0.1)
```

60	C4	96	C7
61	C#4	97	C#7
62	D4	98	D7
63	D#4	99	D#7
64	E4	100	E7
65	F4	101	F7
66	F#4	102	F#7
67	G4	103	G7
68	G#4	104	G#7
69	A4	105	A7
70	A#4	106	A#7
71	B4	107	B7
72	C5	108	C8
73	C#5	109	C#8
74	D5	110	D8
75	D#5	111	D#8
76	E5	112	E8
77	F5	113	F8

Create your own ringtone. Add more notes to the list of constants if you need to. You can use the `time.sleep()` function to create spaces in your music.

Sound Bites

There are many built-in sound files – see <https://education.makeblock.com/help/mblock-python-editor-python-api-documentation-for-cyberpi/> audio for a full list.

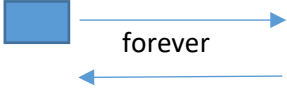
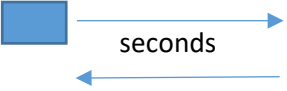
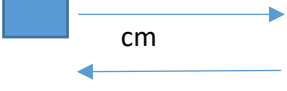
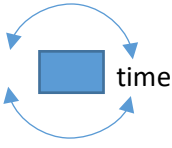
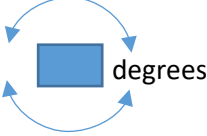
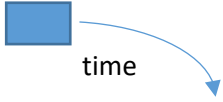
```
import cyberpi as cpi
import time

while True:
    if cpi.controller.is_press('a'):
        cpi.audio.play('yeah')
    elif cpi.controller.is_press('b'):
        cpi.audio.play('bye')
    time.sleep(0.1)
```

File name
hello
hi
bye
yeah
wow
laugh
hum
sad
sigh
annoyed

G. Run the Motors

There are a number of ways we may want to move the mBot2. Forward motor speeds are between 0 and 100. Backward motor speeds are between 0 and -100. Movement still occurs at speeds close to zero.

Movement		Commands
Forward or backward forever. (Should only be used when the ultrasonic sensor or colour sensors are used to control when the motors should stop)		<code>cpi.mbot2.forward(speed = 50)</code> <code>cpi.mbot2.backward(speed = 50)</code> <code>cpi.mbot2.forward(speed = -50)</code> <code>cpi.mbot2.EM_stop(port = "all")</code>
Forward or backward for a length of time		<code>cpi.mbot2.forward(speed = 50, run_time = 1)</code> <code>cpi.mbot2.backward(speed = 50, run_time = 1)</code> <code>cpi.mbot2.forward(speed = -50, run_time = 1)</code>
Forward or backward for a fixed distance		<code>cpi.mbot2.straight(40, speed = 50)</code> <code>cpi.mbot2.straight(-40, speed = 50)</code>
Turn on the spot for a length of time (wheels turning in different directions)		<code>cpi.mbot2.turn_left(speed = 50, run_time = 1)</code> <code>cpi.mbot2.turn_right(speed = 50, run_time = 1)</code>
Turn for a number of degrees of heading		<code>cpi.mbot2.turn(90, speed = 50)</code>
Gradual turn for a length of time (wheels turning in the same direction or one wheel stopped)		<code>cpi.mbot2.drive_power(60, -40)</code> #left +, right - <code>time.sleep(2)</code> <code>cpi.mbot2.EM_stop(port = "all")</code>
Stop motors		<code>cpi.mbot2.EM_stop(port = "all")</code>

Code Templates

There are two basic code templates we use when running motors. In both cases, we use button A to turn on the mBot2 to start the actions.

Separating code into sections makes it much easier to understand the code and make changes to it. Later, we will add more sections as we require them.

1. **One Time Actions.** Use this when the mBot2 actions should only occur once and then finish.

```
#IMPORTS-----
import cyberpi as cpi
import time

#WAIT TO START-----
cpi.console.println('Press A')
while not cpi.controller.is_press('a'):
    cpi.led.on(255,0,0)
cpi.led.on(0,255,0)

#ROBOT ACTIONS-----
cpi.mbot2.forward(speed = 50, run_time = 2)    #Example commands.
cpi.mbot2.backward(speed = 50, run_time = 2)    #Replace with your own!

cpi.led.off()
```

If we have actions that are repeated, we can use a **for loop**. For example, to move in a square:

```
#IMPORTS-----
import cyberpi as cpi
import time

#WAIT TO START-----
cpi.console.println('Press A')
while not cpi.controller.is_press('a'):
    cpi.led.on(255,0,0)
cpi.led.on(0,255,0)

#ROBOT ACTIONS-----
for i in range(4):
    cpi.mbot2.straight(40, speed = 50)    #cm
    cpi.mbot2.turn(90, speed = 50)        #degrees

cpi.led.off()
```

CHALLENGES

1. Place one or more large objects on the floor. Navigate the mBot2 through and/or around them.
2. One of the RoboRAVE competitions is AMAZE-ing. It consists of a series of boards that make up a maze. You do not know the shape of the maze until the competition. The person who keeps the robot on the boards and has the fastest time wins.

2. **Forever Actions.** This code has a **while True** loop that repeats the actions forever – or until you press the **home button** next to the USB connection.

```
#IMPORTS-----
import cyberpi as cpi
import time

#WAIT TO START-----
cpi.console.println('Press A')
while not cpi.controller.is_press('a'):
    cpi.led.on(255,0,0)
cpi.led.on(0,255,0)

#MAIN LOOP-----
while True:
    cpi.mbot2.forward(speed = 50, run_time = 2)    #Example commands.
    cpi.mbot2.backward(speed = 50, run_time = 2)    #Replace with your own!
```

This code is mainly used in conjunction with the joystick and buttons, or the ultrasonic and line follower sensors, where the mBot2 will respond to changes in sensor values.

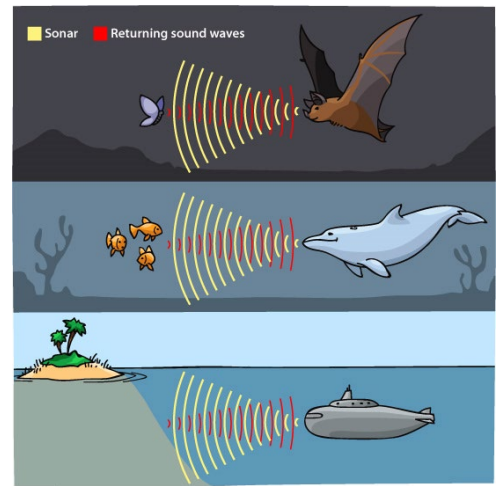
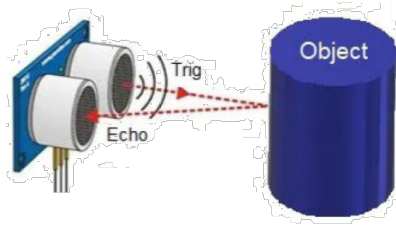
CHALLENGES

3. Place two small objects on the floor at least 1m apart. Drive around these multiple times in a figure of 8. When you turn use the led's to indicate your turns.
4. Place a large object on the floor and turn around the object 3 times in a large, smooth circle. (*Use the `cpi.mbot2.drive_power()` function*)

H. Avoid or Seek

The **Ultrasonic Sensor** is used to measure the distance between the mBot2 and anything in front of it (up to about 200cm). It can be used to avoid obstacles or seek out an object and move toward it.

The minimum distance detected is 4cm. Smaller distances give a reading of 300.



Test your Ultrasonic Sensor with this code. Putting all the sensor reading code into a function unclutters the main loop.

```
#IMPORTS-----
import cyberpi as cpi
import time

#GLOBAL VARIABLES-----
distance = 300

#FUNCTIONS-----
def get_all_values(output=True):
    global distance

    distance = cpi.ultrasonic2.get(index=1)

    if output:
        cpi.console.println( str(distance) )
        time.sleep(0.1)

#WAIT TO START-----
cpi.console.println('Press A')
while not cpi.controller.is_press('a'):
    cpi.led.on(255,0,0)
    cpi.led.on(0,255,0)

#MAIN LOOP-----
while True:
    get_all_values(output=True)
```

Obstacle Avoidance

```
#MAIN LOOP-----
while True:
    get_all_values(output=False)

    if distance < 10:
        cpi.mbot2.EM_stop(port = "all")           #collision test
        cpi.mbot2.straight(-5, speed = 50)         #stop
        cpi.mbot2.turn(135, speed = 50)            #move back 5cm
        cpi.mbot2.turn(135, speed = 50)            #turn 135 degrees
    else:
        cpi.mbot2.forward(speed = 50)              #forward
```

Seek Objects and Move Toward Them

Rotate to detect an object closer than 80cm, then move toward the object.

```
#MAIN LOOP-----
while True:
    get_all_values(output=False)

    if distance > 80:
        cpi.mbot2.turn_left(speed = 50)           #rotate to locate
    else:
        cpi.mbot2.EM_stop(port = "all")           #object detected
        cpi.mbot2.forward(speed = 100)            #stop
        cpi.mbot2.forward(speed = 100)            #forward full speed
```

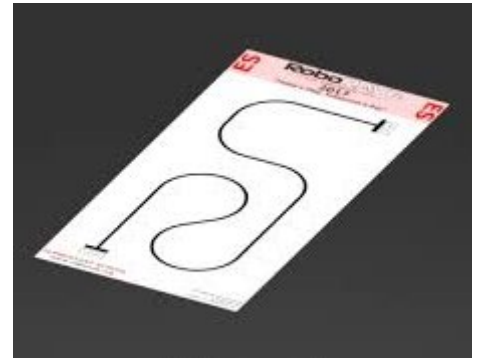
CHALLENGES

5. Place 4 objects at the corners of a square. Find one of them and stop just before you hit it. Turn and find the next object, until you have found all four.
6. Find your way autonomously through a simple maze (sides are 10cm high)

I. Detect and Follow a Simple Line

The **Quad RGB Sensor** (color sensor) enables us to detect and follow lines, and detect colours and respond to the colours in different ways.

The order of the four RGB sensors are (from the left facing forward):



Detecting Lines

The first thing to do is successfully detect lines. Test the sensor using this code, by passing the mBot2 over a black line on a white background with *get_all_values* output set to True.

```
import cyberpi as cpi
import time

#GLOBAL VARIABLES-----
distance = 300
line = 15
last_line = -1
over_line = 0

#FUNCTIONS-----
def get_all_values(output=True, black_line=True):
    global distance, line, any_line

    distance = cpi.ultrasonic2.get(index=1)
    line = cpi.quad_rgb_sensor.get_line_sta(index = 1)

    if black_line: over_line = line < 15
    else: over_line = line > 0

    if output:
        cpi.console.println(str(line) + ' ' + str(distance))

#WAIT TO START-----
cpi.console.println('Press A')
while not cpi.controller.is_press('a'):
    cpi.led.on(255,0,0)
cpi.console.println('')
cpi.led.on(0,255,0)

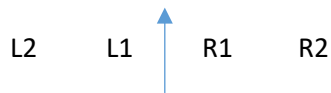
#MAIN LOOP-----
while True:
    get_all_values(output=True, black_line=True)
    if over_line:
        pass          #your code here
```

The sensor may need calibration.
See instructions at the end of this section.

Place the mBot2 inside a Sumo mat (white border on black background) or a white mat with a black border. Use code similar to that used for obstacle avoidance if a line is detected.

Following a Simple Line (No Intersections)

When we follow a line, we need to use all the values that are sent from the Quad RGB sensor. Remember the sequence of sensors (from the left facing forward):



- If both sensors L1 and R1 are on black – go straight ahead
- If only sensor L1 or L2 is on black – turn to the left
- If only sensor R1 or R2 is on black – turn to the right

The values correspond to which sensors are triggered over a line.

15	all off line		7	L2 on line		9	L1 and R1 on line
0	all on line		14	R2 on line			
11	L1 on line		3	L1, L2 on line		1	L1, L2, R1 on line
13	R1 on line		12	R1, R2 on line		8	L1, R1, R2 on line

First, add the following global variables. These will set default values of power to the wheels that can easily be changed.

```
#GLOBAL VARIABLES-----
```

```
hi_p = 25  
lo_p = 5
```

Set up a function with a series of conditional statements in the main loop to deal with each situation – run the motors to follow the line. Call the function in the main loop.

```
#FUNCTIONS-----
def drive_lines():
    global last_line

    if not last_line == line:
        last_line = line

        if line == 7:                                #L2
            cpi.mbot2.drive_power(lo_p, -hi_p)      #turn left
        elif line == 11:                             #L1
            cpi.mbot2.drive_power(lo_p, -hi_p)      #turn left

        elif line == 13:                             #R1
            cpi.mbot2.drive_power(hi_p, -lo_p)      #turn right
        elif line == 14:                             #R2
            cpi.mbot2.drive_power(hi_p, -lo_p)      #turn right

        elif line == 9:                             #L1 and R1
            cpi.mbot2.drive_power(hi_p, -hi_p)      #straight ahead

#MAIN LOOP-----
cpi.mbot2.drive_power(20, -20)

while True:
    get_all_values(output=False, black_line=True)
    drive_lines()
```

See if this works! What is reason for having a ***last_line*** variable?

Challenges

1. Oval or Circuit Race. Follow an oval line or a more complicated circuit from start to finish. Time the run. The robot that does the quickest time wins.
2. Follow a line from beginning to end. How will you detect the end?

Calibrating the Quad Color Sensor

The Quad Color Sensor has a button that enables the sensor to distinguish between the background and the line.

Double-press: When the button is double-pressed, Quad RGB Sensor starts to learn the background and line for line following.

1. Place the light sensors on the **background** of the line-following track map and double-press the button.
2. When you see the LEDs indicating the line-following state blink quickly, **sway the sensors from side to side above the background and line** until the LEDs stop blinking. It takes about 2.5 seconds. The parameter values obtained are automatically stored.
3. If the learning fails, the LEDs blink slowly, and you need to start the learning again.

Long-press: When the button is long-pressed, Quad RGB Sensor switches the color of the fill lights. Generally, you don't need to change the color. The color is set automatically after the learning is complete.

J. Grabbers and Other Mechanics

There are a range of grabbers and other mechanical devices that can be attached to the mBot2. All the devices are operated by servo's or motors. The mBot2 can operate up to 4 servos and extra motors at a time.

		
MakeBlock mBot - Mini Gripper (22-60mm) Servo (approx. \$30)	Makeblock mBot – Robot Gripper (67mm) N20 DC motor (approx. \$50)	DFRobot Maqueen Mechanic – Beetle Servo (approx. \$22)
		
DFRobot Maqueen Mechanic – Loader Servo (approx. \$22)	DFRobot Maqueen Mechanic – Forklift Servo (approx. \$32)	DFRobot Maqueen Mechanic – Push Servo (approx. \$22)

These are all easily mounted on the mBot2 by adding a wooden or metal bar in front of the ultrasonic sensor (which may need raising slightly).

There are two types of servos:

- Micro servos have a range of 180°.
- Continuous rotation servos revolve in a full circle.

Micro Servo Angle Values

0-180 angle of servo

Continuous Servo Angle Values

0-85 Backward
90 Stop
95-180 Forward



Operating the Servos

Up to 4 servos can be plugged in the servo ports on the right-hand side (S3 and S4), or the general IO ports on the left (S1 and S2). This code changes the angle of the servo connected to S1.

```
import cyberpi as cpi
import time

while True:
    cpi.mbot2.servo_set(90, 'S1')
    time.sleep(1)
    cpi.mbot2.servo_set(140, 'S1')
    time.sleep(2)
    cpi.mbot2.servo_set(40, 'S1')
    time.sleep(2)
```

Run DC motors

Additional motors can be run from the M1 and M2 ports.

```
import cyberpi as cpi
import time

while True:
    cpi.mbot2.motor_set(50, 'S1')          #power is -100 to 100
    time.sleep(2)
    cpi.mbot2.motor_stop('S1')
    time.sleep(2)
    cpi.mbot2.motor_set(-50, 'S1')
    time.sleep(2)
```

Both motors can also be run with one command:

cpi.mbot2.motor_drive(power1, power2)

K. SumoBot Competition

SumoBots use the ultrasonic sensor to seek and destroy another robot vehicle in the Sumo ring, while using the color sensor to sense the white border and avoid falling off the edge.

H1. Basic Sumo Code

The basic actions of a SumoBot are:

- A three second wait before doing anything
- Move forward from the edge 20cm
- Rotate until the ultrasonic sensor locates the other vehicle (less than 80cm away)
- Drive full speed toward the other vehicle
- If the white edge is detected (high reflectance value) then stop, back up and rotate to locate the other vehicle

```
#MAIN LOOP-----
found = False
cpi.mbot2.straight(20, speed = 40)

while True:
    get_all_values(output=False, black_line=False)

    if any_line:                                #white line detected
        found = False
        cpi.led.on(0,255,0)
        cpi.mbot2.EM_stop(port = "all")          #stop, back and rotate
        cpi.mbot2.straight(-10, speed = 40)
        cpi.mbot2.turn(90, speed = 40)           #turn to start location

    if distance < 80 or found:                   #other robot detected
        found = True
        cpi.led.on(0,0,255)
        cpi.mbot2.forward(speed = 40)           #forward full speed
    else:
        cpi.mbot2.turn_left(speed = 5)          #rotate to locate
```

H2. Enhancements

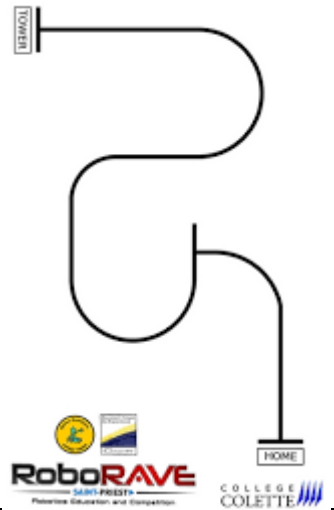
- Don't waste time moving forward at the start before starting to find the other vehicle
- Only scan left and right up to 90 degrees the first time
- Stop every 10 degrees when scanning to make sure scan detects vehicle (moving too fast doesn't work)
- Use movement sensor to detect a collision or the bot lifted off the ground (pitch or roll) and respond to that (see Appendix 1)
- If motion is stopped for x seconds, use a series of rapid wheel movements (e.g. back and forth) to try and get free
- Use a different strategy:
 - Follow white line around the outside (use L2 or R2)
 - Drive to a random place
 - Drive forward until white line and turn and randomly go somewhere else until white line
- Use more than one ultrasonic sensor at different angles

L. Line Follower with Intersections

Line following involving intersections, such as the RoboRave Line Follower competition, is a lot more complex than simple lines or circuits explored in section K.

Ideally, we would use the two inner sensors (L1 and R1) to track a line, and combinations using the two outer sensors (L2 and R2) to check for intersections.

However, the two outer sensors are often triggered if the lines are tightly curved. We need to switch the sensors between simply following lines and detecting sections, then use a timer to switch between sections of a track.



1. Add code for a simple timer.

```
#GLOBAL VARIABLES-----
```

```
timer_start = 0
finished = False
```

```
#FUNCTIONS-----
```

```
def start_timer():
    global timer_start, finished
    finished = False
    timer_start = time.time()

def elapsed_time():
    return time.time() - timer_start
```

2. Add the options to detect intersections to the *check_lines()* function. Note the option to check for intersections or not.

```
def drive_lines(intersection_on = False):
    global last_line, finished
    if not last_line == line:
        last_line = line

    if intersection_on and line in [0,1,3,8,12,15]:
        if line == 15:
            #all off line
            pass
        elif line == 0 or line in [1,3] or line in [8,12]:
            #all - L1, L2, (R1) - (L1), R1, R2
            cpi.mbot2.EM_stop(port = 'all')
            finished = True

    elif line == 7:
        #L2
        cpi.mbot2.drive_power(lo_p, -hi_p)
        #turn left
```

3. **Write a function to drive a section of line**, with or without checking for intersections. Note that all the commands from the main loop go into this section.

```
def drive_section(seconds = 99999, check_intersection = True):
    start_timer()
    while elapsed_time() < seconds and not finished:
        get_all_values(output=False, black_line=True)
        drive_lines(intersection_on = check_intersection)
        if show_led: led_lines()
```

4. **Add *drive_section()* functions to the main loop** in a sequence to cover the whole track. This sequence can later be looped to continue forever.

```
#MAIN LOOP-----
cpi.mbot2.drive_power(hi_p, -hi_p)

cpi.console.println('Running 1')
drive_section(seconds = 10, check_intersection = False)    #section 1 no intersection

cpi.console.println('Running 2')
drive_section(seconds = 8, check_intersection = True)      #section 2 stop at intersection

#your code to turn at the intersection and continue with more sections goes here

cpi.mbot2.drive_power(0, 0)
```

To follow the line faster, you might need the change:

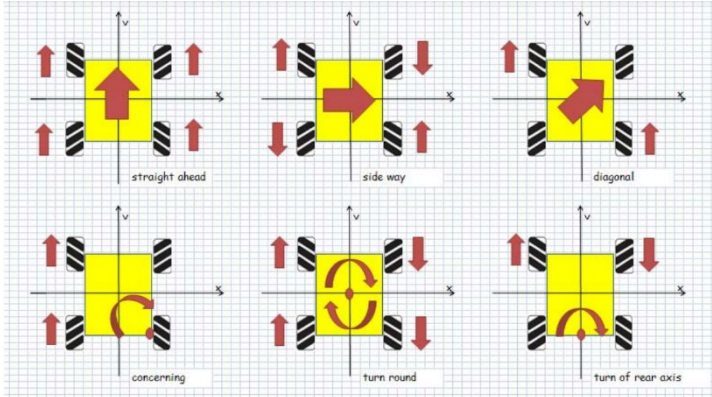
- The power to the left and right wheels
- The difference in power between the left and right wheels
- How you interpret the percentage color sensor values
- Use the L2 and R2 sensors as well

CHALLENGES

6. RoboRAVE Line Follower Competition. Be the fastest robot to get from home to the box.

M. Mecanum Wheels

Mecanum wheels provide multi-directional movement. All the mBot2 components (except for motors) can be installed on the back of the chassis shown here, leaving room for mechanical arms, ping-pong ball launchers etc.



These wheels can be driven using the EM motor outputs combined with motor outputs M1 and M2.

Write a function to run all four motors at the same time.

```
def run_motors(dir='straight', speed1=40, speed2=0, t=0):
    if dir == 'straight':
        cpi.mbot2.motor_drive(speed1, speed1)    #front
        cpi.mbot2.drive_power(speed1, -speed1)  #back em

    elif dir == 'right':
        cpi.mbot2.motor_drive(speed1, -speed2)
        cpi.mbot2.drive_power(-speed2, -speed1)
    elif dir == 'left':
        cpi.mbot2.motor_drive(-speed2, speed1)
        cpi.mbot2.drive_power(speed1, speed2)

    elif dir == 'turnright':
        cpi.mbot2.motor_drive(speed1, speed2)
        cpi.mbot2.drive_power(speed1, -speed2)
    elif dir == 'turnleft':
        cpi.mbot2.motor_drive(speed2, speed1)
        cpi.mbot2.drive_power(speed2, -speed1)

    elif dir == 'diagright':
        cpi.mbot2.motor_drive(speed1, speed2)
        cpi.mbot2.drive_power(speed2, -speed1)
    elif dir == 'diagleft':
        cpi.mbot2.motor_drive(speed2, speed1)
        cpi.mbot2.drive_power(speed1, -speed2)

    if t > 0:
        time.sleep(t)
        cpi.mbot2.motor_stop("M1")
        cpi.mbot2.motor_stop("M2")
        cpi.mbot2.EM_stop(port = "all")
```

Test the motors in various combinations for short periods of time.

```
while True:

    run_motors(dir='straight',speed1=40,speed2=40,t=3)
    run_motors(dir='diagright',speed1=80,speed2=0,t=3)
    run_motors(dir='turnleft',speed1=40,speed2=0,t=3)
    run_motors(dir='right',speed1=40,speed2=40,t=3)

    run_motors(dir='straight',speed1=-40,speed2=-40,t=3)
    run_motors(dir='diagleft',speed1=80,speed2=0,t=3)
    run_motors(dir='left',speed1=40,speed2=40,t=3)
```

N. Robotic Arm with 4 DOF

There are many 4 DOF robotic arms on AliExpress that are relatively inexpensive to purchase (less than \$30). However, be very careful of the quality.

It is very important to check the positions of the servos during construction – making sure that the middle position of the servos is about 90°.

Connect to the following ports.

- S1 base servo
- S2 left servo up/down
- S3 right servo forward/back
- S4 grabber servo open/closed

The left and right servos are best operated together.



To operate the robotic arm, create four global variables.

```
#GLOBAL VARIABLES-----
servo1 = 100    #S1 base
servo2 = 140    #S2 left up/down 120/90    170/30
servo3 = 80     #S3 right forward/back 170/60
servo4 = 160    #S4 grabber  open/closed 90/160
```

Write a function to reset servos to their default positions.

```
def reset_servos():
    #S1 base, S2 left up/down, #S3 right forward/back, s4 grabber
    cpi.mbot2.servo_set(servo1, 'S1') #base 100, 10 right
    cpi.mbot2.servo_set(servo2, 'S2')
    cpi.mbot2.servo_set(servo3, 'S3')
    cpi.mbot2.servo_set(servo4, 'S4') #90 open 160 closed
```

Write a function to move the servos slowly between the current position and a new position.

```
def slow_servo(s, old, new):
    global servo2
    if new > old: step = 1
    else: step = -1
    for angle in range(old, new, step):
        cpi.mbot2.servo_set(angle, s)
        if s == 'S3':
            servo2 = 200 - 0.6 * angle    #0.7
            cpi.mbot2.servo_set(servo2, 'S2')
        time.sleep(0.05)
    return new
```

Test the servos to make sure they work well. Servo 2 (up/down) is automatically operated in conjunction with servo 2 (forward/back).

```
servo1 = slow_servo('S1', servo1, 140) #base rotate
servo1 = slow_servo('S1', servo1, 40)
servo1 = slow_servo('S1', servo1, 100)

servo4 = slow_servo('S4', servo4, 80) #grabber open
servo4 = slow_servo('S4', servo4, 160) #closed

servo3 = slow_servo('S3', servo3, 50) #left
servo3 = slow_servo('S3', servo3, 160) #left
```

O. Keep the Light Just Right

They automatically turn on car headlights, street lights or house lights as it gets dark, brighten your phone screen if there is more sunlight, or turn it down to conserve power when you are inside. you are growing food, more hours of light and brighter light will make plants grow faster.

All these things need **light sensors** to work. We will use the built-in light sensor, located above the joystick.

To read and information from sensors, variables must be used to store the sensor values.



If

```
import cyberpi as cpi
import time

while True:
    light = cpi.get_bri()
    cpi.console.println(str(light))
    time.sleep(0.1)
```

Pass your hand over, then away from, the LED's.

Let's draw a **graph** of the light data instead.

```
import cyberpi as cpi
import time

while True:
    light = cpi.get_bri()
    cpi.linechart.add(light)
    time.sleep(0.1)
```

Brighten the led's in Sunlight

This code makes the led brightness proportional to the light level. When there is more light, the led's are brighter.

```
import cyberpi as cpi
import time

while True:
    light = cpi.get_bri()
    cpi.linechart.add(light)
    cpi.led.set_bri(light)
    cpi.led.on(0,0,255)
    time.sleep(0.1)
```


Turn on the Lights When it gets Dark

To turn on the lights as it gets dark we only need to make one small change to the code. If the maximum is only 70, change the `set_bri()` function value to `(70-light)`.

```
import cyberpi as cpi
import time

while True:
    light = cpi.get_bri()
    cpi.linechart.add(light)
    cpi.led.set_bri(100-light)
    cpi.led.on(0,0,255)
    time.sleep(0.1)
```

Light Level Warning

We can use **conditional statements** to make a light warning system. In the conditional statement, if a condition is True then a section of code is executed. If it is False, a different sequence of code is executed (or no code is executed).

```
import cyberpi as cpi
import time

while True:
    light = cpi.get_bri()
    cpi.linechart.add(light)

    if light < 20:
        cpi.led.on(255,0,0)
    elif light < 40:
        cpi.led.on(255,255,0)
    elif light < 60:
        cpi.led.on(0,255,0)
    else:
        cpi.led.on(0,0,255)

    time.sleep(0.1)
```

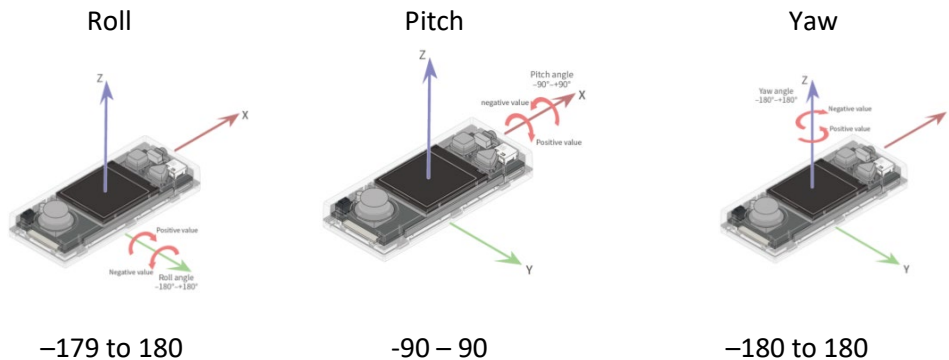
P. Rollover Warning

To control a drone in the air requires a sensor called an accelerometer. It senses the movement of the aircraft. There are three basic movements of a drone or robot:

- Roll sideways movement
- Pitch forward, back movement
- Yaw rotation or heading



An accelerometer can measure roll, pitch. To measure yaw, a gyroscope is used.



The following code draws a graph of the values on the display

```
import cyberpi as cpi
import time

cpi.reset_yaw()
while True:
    roll = cpi.get_roll()
    pitch = cpi.get_pitch()
    yaw = cpi.get_yaw()

    cpi.linechart.add( (roll+200)/4 )
    time.sleep(0.2)
```

As you move the mBot2 the graph will change.

Roll – With the mBot2 facing forward away from you, lower the left side, then lower the right side down.

```
cpi.linechart.add( (roll+200)/4 )
```

Pitch – Push the front of the mBot2 down, then push the back down.

```
cpi.linechart.add( (pitch+200)/4 )
```

Yaw – Keeping the mBot2 horizontal, rotate it to the left and right

```
cpi.linechart.add( (yaw+200)/4 )
```

4WD Rollover Warning

An accelerometer can be used to warn when the angles are too steep. We will create a warning system using LED messages and sound.



```
import cyberpi as cpi
import time

while True:
    roll = abs( cpi.get_roll() )
    pitch = abs(cpi.get_pitch() )

    cpi.console.println(str(pitch) + ', ' + str(roll))

    if pitch > 30 or roll > 30:
        cpi.led.on(255,0,0)
        cpi.audio.play_music(100,0.2)

    elif pitch > 20 or roll >20:
        cpi.led.on(255,255,0)
        cpi.audio.play_music(60,0.2)

    else:
        cpi.led.on(0,255,0)
        time.sleep(0.2)
```

Carefully move the mBot2 so you test the roll and pitch as described on the previous page. Note the different sounds produced.

Q. Connect Other Sensors

Read Analog Sensors

Read analog sensors (such as potentiometers or soil moisture sensors) using ports S1 and S2

```
value = cpi.mbot2.read_analog(port)           #returns 0 - 5V
```

Read and Write Digital Sensors

```
cpi.mbot2.write_digital(val, port)           #val = True, False, 0, 1  
value = cpi.mbot2.read_digital(port)         #returns True, False
```

Appendix 1 CyberPi Extras

Slider (potentiometer) and multi-touch

```
import cyberpi as cpi
import time

while True:
    pot = cpi.slider.get()
    touch = cpi.multi_touch.is_touch(ch = 1)    #1-8 or ch = "any"
    print(distance, pot, touch)
    time.sleep(0.1)
```

Appendix 2. RoboRave a-Maze-ing Track Details

Triangle Connector (Isosceles right triangle) of $45^\circ - 45^\circ - 90^\circ$

Hypotenuse @33 cm

Legs @23 cm

Each set of tracks is cut from 1 of 1200mm x 1200mm x 9mm MDF

All tracks are 230 - 240mm wide

Cut list:

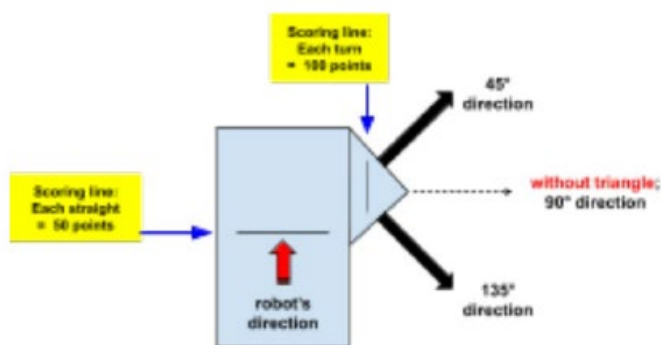
1 @ 1200 x 240

1 @ 800 x 240

4 @ 600 x 240

2 @ 400 x 240

3 @ 240 x 240 – Then cut diagonally to form 6 triangles.



template pieces for **EXAMPLE 1**
RoboRAVE Elementary School Design

Ignore Dimensions in this diagram

