Introduction

Meet ExoNaut! It is a small, 3-wheeled robot intended to teach students the basics of programming and robot control.

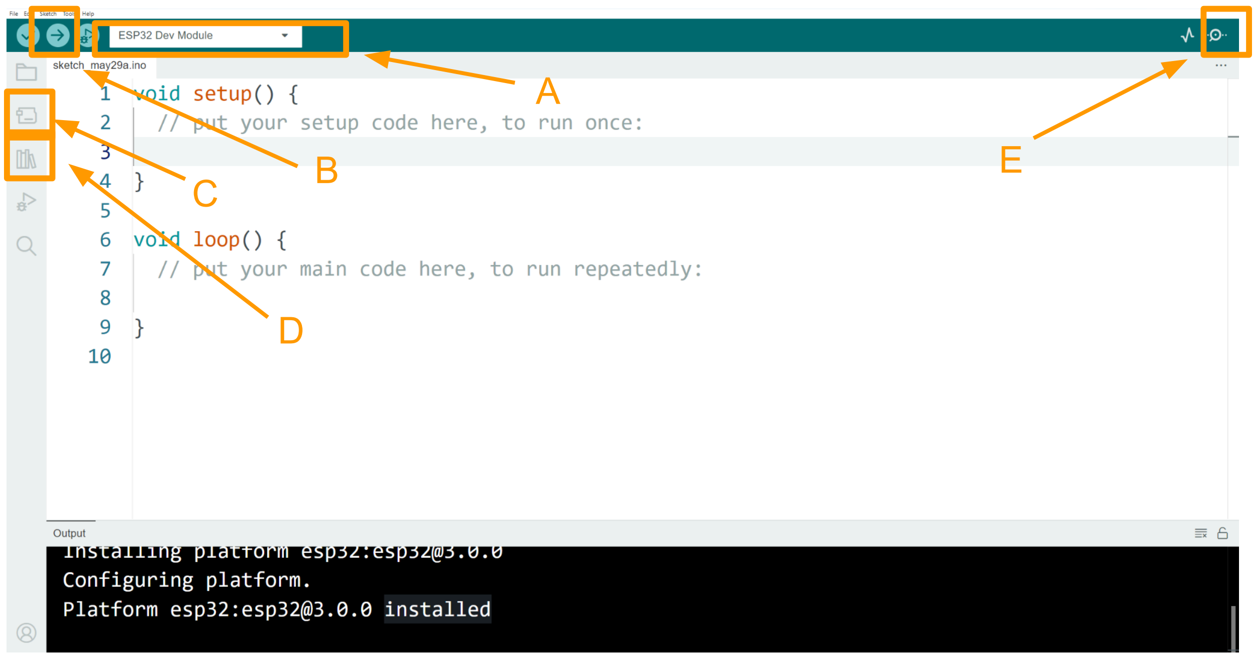
The standard kit comes with:

- A line follower sensor, and
- an ultrasonic distance sensor.

Several additional sensors are available, including an AI camera that allows for color and object detection, which provides the robot with a greater level of autonomy. The controller also has WiFi/Bluetooth and infrared control capabilities.

For robot assembly, see the kit. This booklet explains how to use an **already assembled ExoNaut robot** and includes examples of lessons, challenges, and supplementary material.

Programming the robot can be done with a Scratch (blocking) environment, Python, or Arduino. This guide will focus on the **Arduino environment.**

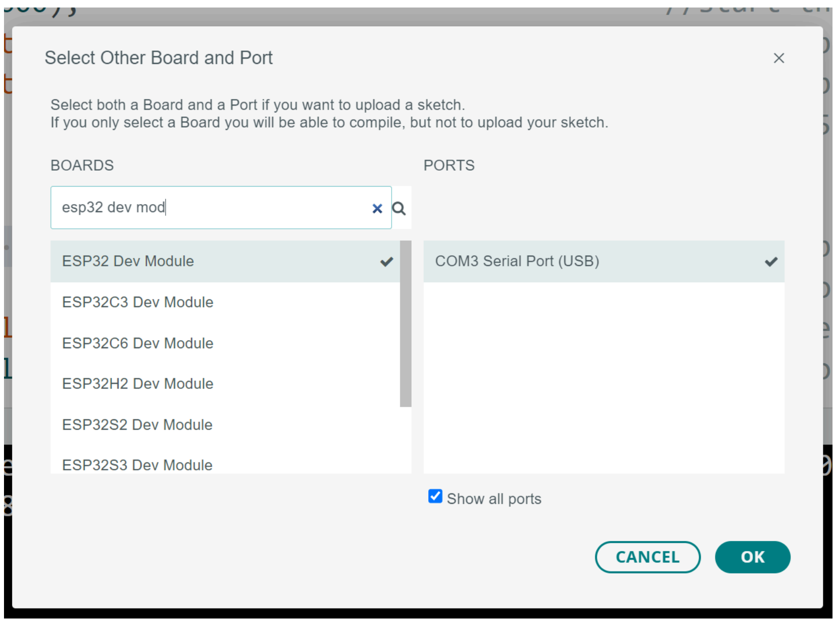


```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
Installing platform esp32:esp32@3.0.0
Configuring platform.
Platform esp32:esp32@3.0.0 installed
```

There is a section for changing the type of board and selecting your robot (A), a button to upload code (B), a boards manager tab (C), a library manager tab (D), and a serial monitor (E) where the program outputs text to the computer.

Setup


Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS	PORTS
esp32 dev mod	
ESP32 Dev Module ✓	COM3 Serial Port (USB) ✓
ESP32C3 Dev Module	
ESP32C6 Dev Module	
ESP32H2 Dev Module	
ESP32S2 Dev Module	
ESP32S3 Dev Module	

☑ Show all ports

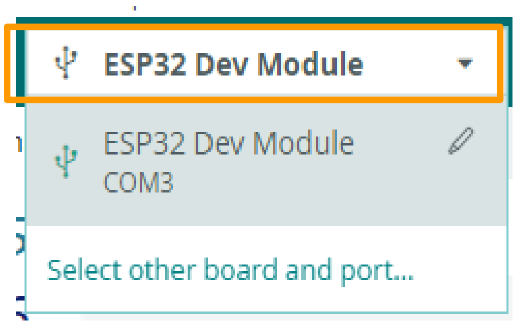
Step 1: Open Arduino IDE and plug your robot into the computer.

Step 2: Click the board and ports section, labeled A in the image on the previous page.

Step 3: Select ESP32 Dev Module and whatever COM# your robot is, then click OK.

ESP32 Dev Module should be in bold if done correctly, as highlighted below.

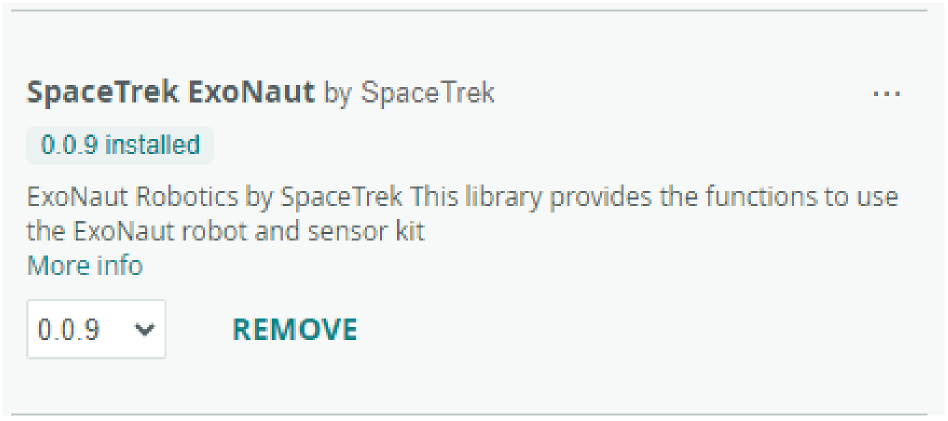
Your robot is now connected and ready to be programmed!



Skip the Below Step if at SpaceTrek:

Step 4: Select the 'libraries' tab on the left-hand side of Arduino. Search and download **SpaceTrek ExoNaut** including all dependencies.

You are now ready to begin coding in Arduino, and you are now able to access all examples that will be used in the next challenges on how to use the ExoNaut.


SpaceTrek ExoNaut by SpaceTrek

0.0.9 installed

ExoNaut Robotics by SpaceTrek This library provides the functions to use the ExoNaut robot and sensor kit
More info

Arduino Introduction

Variables and Data Types

In Arduino, there are various ways to store information. One of these ways is to store information in what is called a **variable**.

There are many different types of variables and **data types**, but for our purposes, we are going to focus on 4 types.

Void: a void returns nothing. This will be common in our setup functions and our loop functions.

```
void loop
// this does not return anything.
```

Boolean: a boolean variable can return only one of two states, which are true or false. If the answer to your question is a ‘yes’ or ‘no’, that is what this is referring to.

```
bool raining = True
// it either is or is not raining.
```

Integer: or int for short, refers to a number that is a whole number.

```
int people = 1
// you can only have whole numbers of people.
```

Float: a float is a number that has a decimal point. This can be useful for storing values such as a total dollar amount.

```
float money = 1.28
// even if you have $1.00, it still needs to be a float.
```

Arduino Syntax

```
1  void setup() {
2    int a = 12;
3    // this is a comment
4    /* this is also a comment */
5  }
```

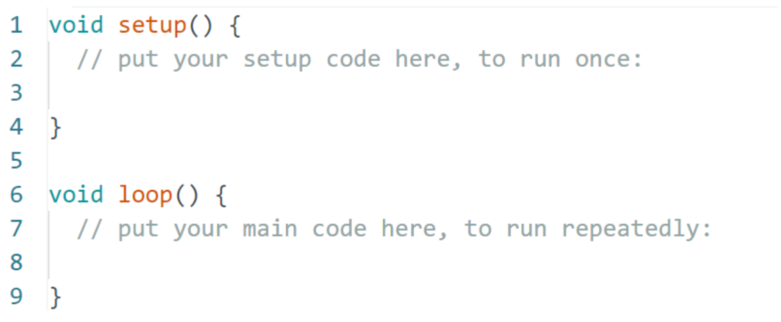
Each function must have a beginning and end brace, or the program will not run.

Nearly every line of code must end in a semicolon to let Arduino know your statement is done. There are a few exceptions, though.

Comments can either be formatted as in line 3 or line 4. You will notice there are lots of different ways to write the same things in Arduino. Let's dive in!

Getting Started

To add lines of code or program the robot, your code must be within the braces of a function. The ExoNaut robot needs two special functions to work correctly; "**void setup**" to run upon startup, and "**void loop**" to run repeatedly. Even if you have nothing to run in these sections, they must still be included in the code.

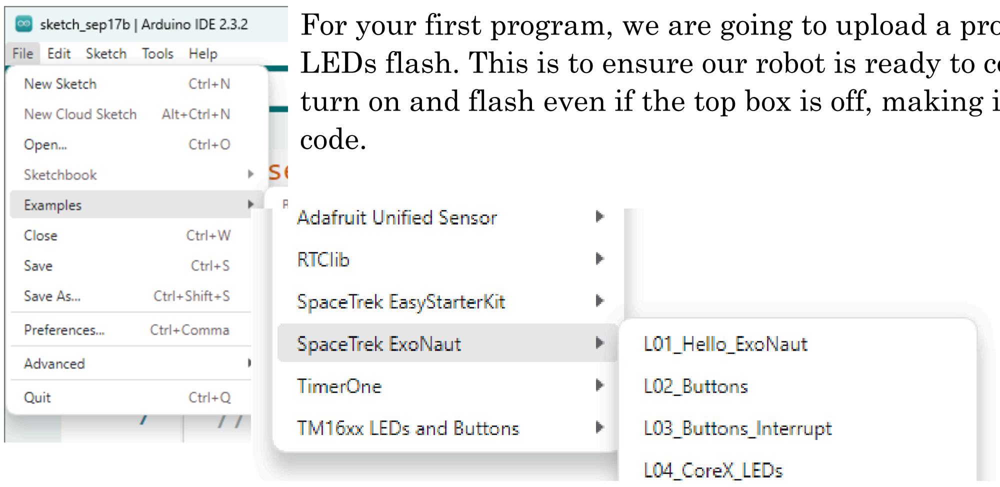


```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

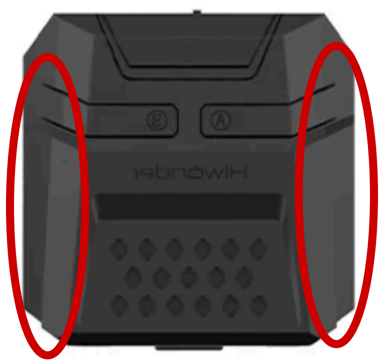
To upload code to your robot, make sure that the top box on the robot is **OFF** and then hit upload.

There is a library of examples that the ExoNaut comes with. We will use these to learn more about our ExoNauts, including how to use buttons, how to drive, turn, and use various sensors.

Note: Before uploading ANY code to your robot, ensure the top box is OFF. You should be able to visually check this by making sure that the ultrasonic distance sensor (or the robot's 'eyes') is not on.



For your first program, we are going to upload a program to make the LEDs flash. This is to ensure our robot is ready to code. The LEDs will turn on and flash even if the top box is off, making it a perfect starter code.



Step 1: Go to File > Examples > SpaceTrek ExoNaut and click **L04_CoreX_LEDs.**

Click upload, which is the arrow button at the top left. It will take a little while to upload, but once it finishes our robot should be lighting up. Take a moment to read through the code and comments to learn more about how the RGB LEDs work.

Challenges:

Throughout this booklet, you will see blue boxes like this one. After you complete the lesson on the page, see if you can complete these challenges for points. Call over an instructor to show them in order to collect points for your team if you succeed. If not, try again! Coding takes lots of trial and error, and it will be that much more satisfying in the end when it works. You've got this!

Movement: Going forward

Let's make our robot drive forward for 1 second and then stop. As explained previously, go to File > Examples > SpaceTrek ExoNaut and click L05_Go_Straight.

What you'll notice about this code is the comment at the top–it explains what the code does, and includes commands that you can use, how they work, and what they do. Let's take a look.

```
 * Commands:
 * exonaut robot;
 *
 *
 * robot.begin();
 *
 *
 * robot.set_motor_speed(left, right);
 *
 *
 * robot.stop_motor(motorID);
```

exonaut robot; This command sets up the exonaut robot object. This is the object that handles all of the motor commands and the features on the CoreX controller.

robot.begin(); This command is used once at the beginning of the program.

robot.set_motor_speed(left, right); This command sets both motor speeds. It takes integer numbers.

robot.stop_motor(motorID); This command stops the motors. MotorID 0 is both motors, motorID 1 is the left, and motorID 2 is the right.

```
25  #include <ExoNaut.h>
26
27  exonaut robot;
28  int speed = 50;
29
30  void setup() {
31    robot.begin();
32    robot.set_motor_speed(speed, speed);
33    delay(1000);
34    robot.stop_motor(0);
35  }
```

This code includes the ExoNaut library (25), initializes the robot (27), and sets an integer variable 'speed' (28).

Next, in the setup section, the robot is turned on (31), both motors are set to the same speed as specified in line 28 (32), and the delay (33) is for 1000ms before both come to a stop (34).

Upload your code, unplug your robot, and turn it on on a **flat surface** it can't fall off of.

Congratulations! You have successfully uploaded your first program to your robot. Now see if you can use what you've learned to earn points by completing the challenges below.

Challenges:

10 pts: Make your robot go backwards, then stop.

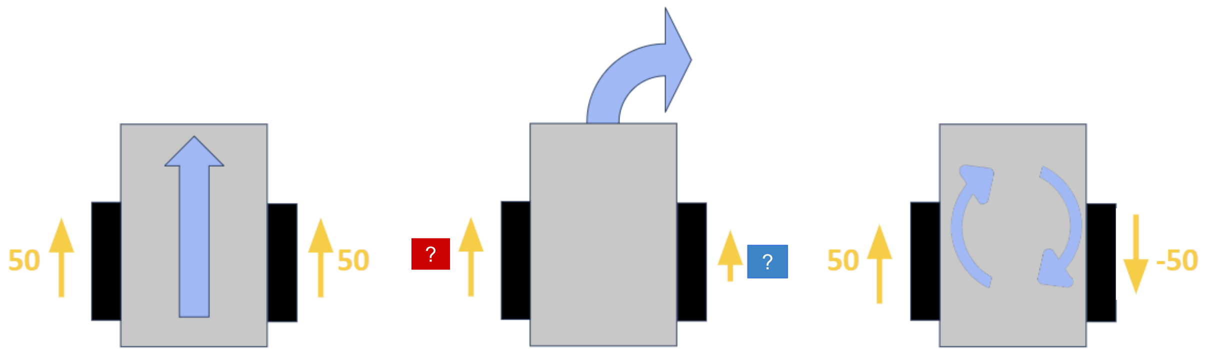
20 pts: Make your robot go forwards, wait one second, then go backwards, then stop.

100 pts: Make your robot go **exactly one meter**. **You only get one graded attempt, so make it count!** Ask an instructor for a meter stick. Every centimeter off is minus ten points.

Movement: Turning

Let’s make our robot pivot in place for one second.

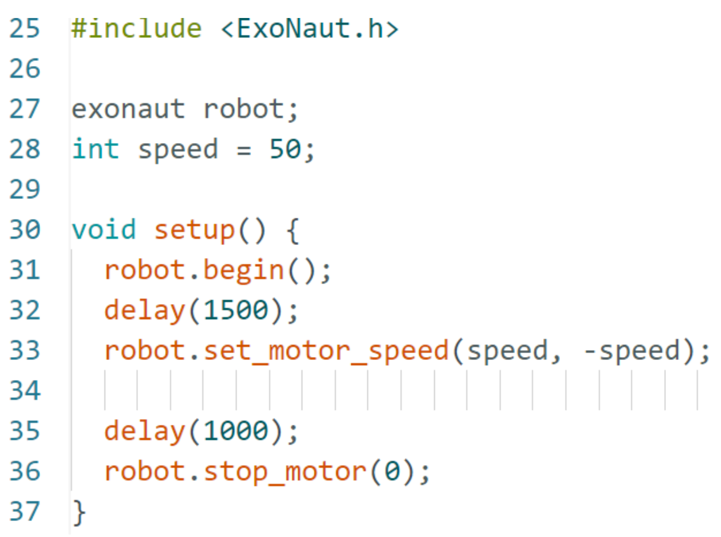
Earlier, we set both motors or wheels, to the same speed, causing the robot to move forward.



By this logic, if we want to make the robot turn, the wheels must be at different speeds. Using what we’ve learned so far, and making sure our robot is turned off during the uploading process, do the following:

Step 1: Open File > Examples > SpaceTrek ExoNaut > **L08_PivotRight.**

Step 2: Upload the code to your robot. What is different about this code than **L05_GoStraight**? What happens if you use different values for both motors?



```
#include <ExoNaut.h>

exonaut robot;
int speed = 50;

void setup() {
  robot.begin();
  delay(1500);
  robot.set_motor_speed(speed, -speed);

  delay(1000);
  robot.stop_motor(0);
}
```

Note: One second of turning your robot makes it turn some angle. For a challenge below, use this information to determine the value of an accurate 90-degree turn.

Challenges:

20 pts: Can you make your robot go in a circle with some radius?

50 pts: Make your robot turn exactly 90 degrees.

100 pts: Make your robot make a figure 8.

Buttons: Changing colors

Next, we'll need to learn how to use the buttons on top of the robot, labeled 'A' and 'B', to control the robot. Let's change the buttons to give a serial output depending on which button was pressed.

Step 1: Open File> Examples> SpaceTrek ExoNaut> **L02_Buttons**

Step 2: Let's look at the code. There are two states the button can be in. The way the robot knows if the button is pressed is if the pin that the button presses is high or low. So if you press the button, the pin is low. If it is not pressed, the pin will be high.

```
void loop(){
  if(digitalRead(BUTTON_A_PIN) == LOW){
    Serial.println("Button A was pressed");
    while(digitalRead(BUTTON_A_PIN) == LOW);
  }
  if(digitalRead(BUTTON_B_PIN) == HIGH){
    Serial.println("Button B was pressed");
    while(digitalRead(BUTTON_B_PIN) == HIGH);
  }
}
```

So this code is continuously checking to see if the pin for button A is low, or if the button is pressed. If this is true, the serial monitor will print "button A was pressed". The same is true for button B, but you might notice something different. The pin is high when the button is pressed. The logic, however, is the same.

Step 2: Run your program, open the serial monitor, and give it a try.

Congratulations! You now know how to make the robot do different things if the button is pressed. What else can you do if the button is pressed? Try the challenges below.

Challenges:

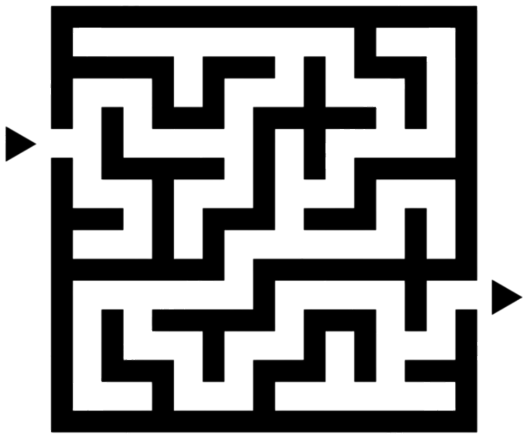
20 pts: Make the LEDs exactly white. The preset white isn't actually white!

50 pts: Make your robot go forward when you press button A and backwards when you press button B.

100 pts: Make it so you can hold down button A to make the robot go forward and then press button B to turn.

Competition 1: Manual Line Challenge

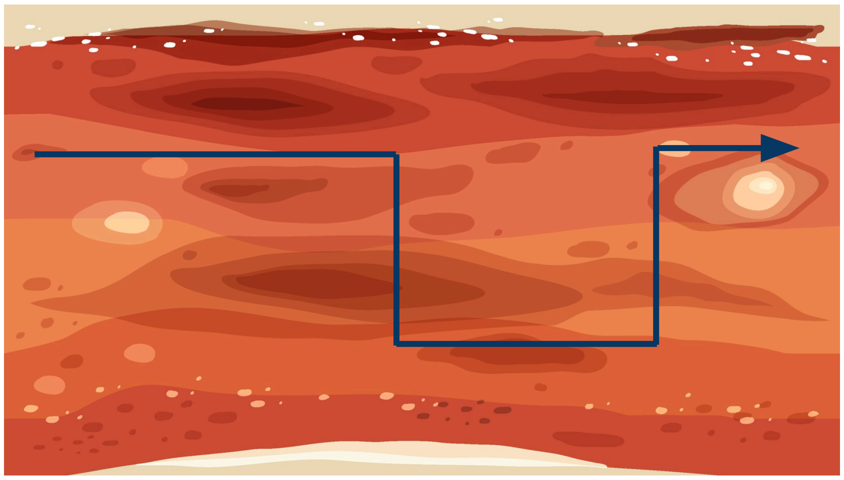
Congratulations! You now know how to go forward, use the LEDs, turn, and use buttons to perform different operations. Let's practice these skills and program our robot to complete a maze for points.



Your instructor will show you the maze your robot needs to complete in order to win this competition.

Your task is to have your robot complete this maze by manually following the line on the mat.

Take a look at the example track below. If the line looks like this, how do you tell your robot where to go if it is at the starting line?



```
robot.set_motor_speed(speed, speed);
delay(1000);
robot.stop_motor(0);
delay(1000);

robot.set_motor_speed(speed, -speed);
delay(1000);
robot.stop_motor(0);
delay(1000);

robot.set_motor_speed(speed, speed);
delay(1000);
robot.stop_motor(0);
delay(1000);

  robot.set_motor_speed(-speed, speed);
delay(1000);
robot.stop_motor(0);
delay(1000);
```

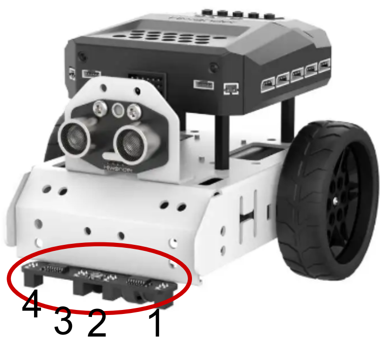
This example code has the robot go forward, turn right, go forward, turn left, go forward, turn left. Each action takes one second.

However, your robot may not have one second of turning as a perfect turn, which means you will have to find the correct time to make the turn. You also will need to determine how many seconds the robot has to drive for to follow the whole line.

Once your robot is ready to complete the actual maze, to earn points for it call over an instructor for your test trial. A successfully completed maze with no deviations from the line is worth 200 points. Every cm off from the end is minus 2 pts, and -15 pts for significant deviations from the line. +2 pts for creativity. You may have multiple attempts.

Competition 2: Line Follower Challenge

You now should have mastered how to go specific distances and turn precise angles. Now let's shift our focus to a new sensor and focus on speed.

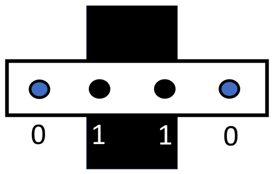


The new sensor we will be using for this challenge is the line follower. This is at the bottom of the robot shown here.

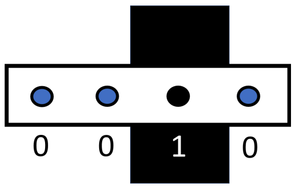
The line follower operates like a boolean–it senses if there is black below it (our line, in this case). We will code the robot to move differently on a case-by-case basis depending on where the line is and which sensor(s) have been triggered to go as fast as it can.

See the labels on the image above to determine which sensor is which. So for instance, if sensors 1 and 2 detect the line, the case will be 0b**0011**, where each number represents a sensor, 1 represents a 'true (black)', and 0 represents a 'false (white)'. The 0b at the beginning just indicates binary and can be ignored.

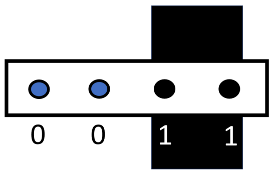
Step 1: Open File > Examples > SpaceTrek ExoNaut > L13_Line_Follower_Advanced. You should see various switch cases that make the motors go at different speeds, some of which are incomplete. Here is a helpful diagram to understand which way your robot will need to move in different scenarios.



The line follower detects black on sensors 2 and 3, or case 0b**0110**. How should your robot move in this scenario?



The line follower detects black on sensor 2, or case 0b**0010**. How should your robot move in this scenario to ensure the robot stays on track?



The line follower detects black on sensors 1 and 2, or case 0b**0011**. How should your robot move in this scenario to ensure the robot stays on track? What about in comparison to the previous diagram?

Step 2: Find the cases that do not have conditions under them (they should say //what to do?) and think about them. What should your robot do in these cases? Include robot.set_motor_speed(left, right) and include numbers for left and right.

Call over an instructor for your trial. Points will be provided with different point tiers depending on how fast your robot completes the challenge. You may have multiple attempts.