

Review the following material on sensors. Discuss how you might use each of these sensors. When you have completed reading through this material, build a robot of your choosing that has 2 motors (connected to ports A and C), a light sensor pointed towards the ground on the front of the robot, and a touch sensor on the back of the robot. If you want and you have time, try building the bumper illustrated below. This robot will be used in the programming portion of the lesson.

Sensors provide feedback to the robot to tell it how to respond to its surroundings. The touch sensor and the light sensor perform like our own senses of touch and sight. The rotation sensor provides feedback on how many turns a wheel or gear makes. A person may program the robot to respond to each of these senses.

Touch Sensor

The touch sensor is a 2x3 gray block with a yellow button protruding from one end. A wiring harness is used to connect it to the RCX. The 2x2 plate end of the wiring harness must be placed on the top of the sensor toward the end with the yellow button. The direction that the wire protrudes from the sensor is not important. The other end of the wiring harness may be placed on one of the three numbered ports on the RCX. Once again the wire orientation is not important. Whenever the yellow button is depressed a circuit is completed and the RCX may be programmed to respond to the opening or closing of that circuit. See Figure 1 for wiring examples.

The RCX has a view function that allows one to see the value of a sensor. When the RCX is turned on, press the view button repeatedly and move the carrot (>) around the display screen until it points to the numbered port where the pressure sensor is attached. When the carrot is pointing at the port with a touch sensor, a zero will appear next to the stick figure on the display. If you press the yellow button on the touch sensor the zero(0) will change to a one(1). Touch sensors may be stacked. If a second touch sensor is wired to the same port depressing either sensors' yellow button will change the display from zero to one. Have your team members all get a chance to operate the touch sensor with the view function active.

The light sensor, rotation sensor, and motors may all be monitored using the view function on the RCX.

Light Sensor

The light sensor is a 2x4 blue block with its own wiring harness. It may be wired to any of the numbered ports on the RCX. The orientation of the 2x2 plate on the RCX port is not important. When the light sensor is active a red light shines from the end of the sensor. (When you first attach a light sensor to a port the red light may not be on. A program must be run to initiate the RCX to recognize the light sensor once it is attached to a port. Once the RCX recognizes that a light sensor is on a numbered port the red light will shine continuously, even when the program is stopped.) When the light is placed close to a surface the light reflects off of that surface and returns to the sensor. On a

white or shiny surface more light reflects back to the sensor and it records a larger signal. On a black or rough finished surface less light reflects back and a smaller signal is recorded. This sensor is not a simple on/off like the touch sensor. The amount of light that returns to the sensor is recorded as a number between zero and one hundred. Wire the light sensor to port 2 on the RCX and use the view function to see the output of the sensor. The display will show a number between zero and one hundred. Point the sensor at various surfaces and see how the value changes. Have your team members perform this test.

Bright lights in room increase the light returning to the sensor and it records a larger signal. This may be tested by turning the room lights on and off while viewing the output of the sensor using the view function. Notice that most of the readings vary between 20 and 60. It is difficult to find a surface that produces a reading near zero or one hundred. If a light sensor is used to follow a dark line the view function may be used to insure that the numeric value returned to the RCX for that dark line versus value returned for the white background crosses a threshold value that you set in the software.

Rotation sensor

The rotation sensor is a brick with a rotating shaft extending from it. When wired to the RCX it counts each 1/16th of a rotation as one unit. A full rotation of the sensor shaft returns a value of sixteen. A series of gears may be used to connect the rotation sensor to a robot wheel, or other rotating part on the robot to be sensed. A gear train may increase the number of rotations made by the sensor shaft versus the point of the robot being monitored to get higher precision in the rotation sensor. Think back to the extreme gear example to understand how this is accomplished. The view function may be used to see how the rotation sensor counts as the shaft turns.

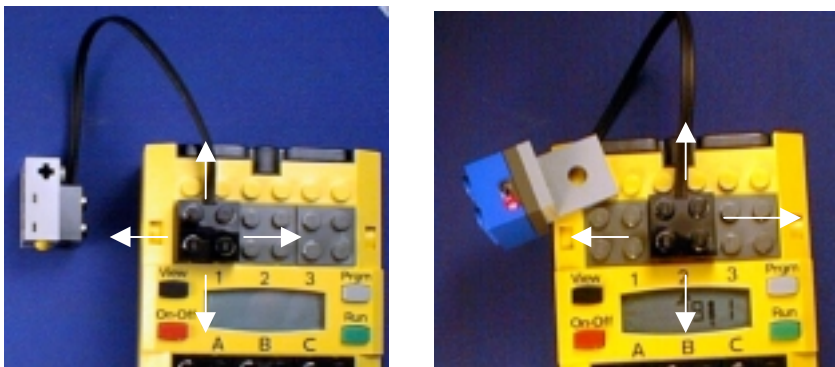
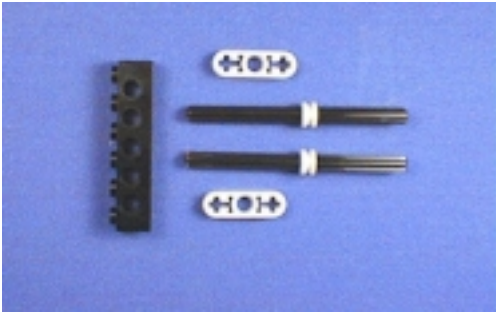
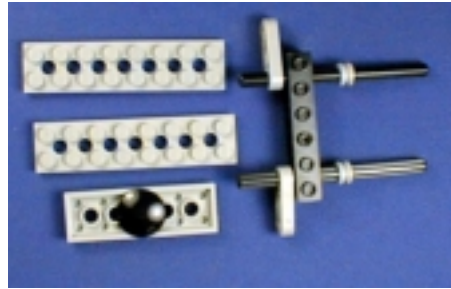


Figure 1. Touch and Light sensors may be oriented in any direction.

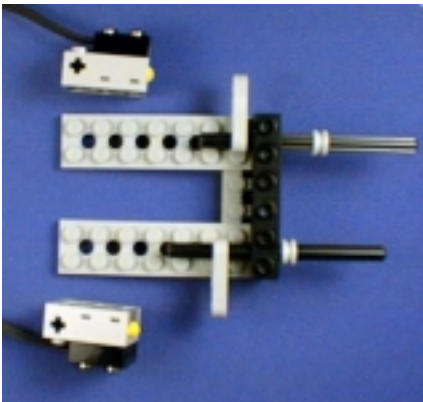
Here is one way to build two touch sensors into a robot.



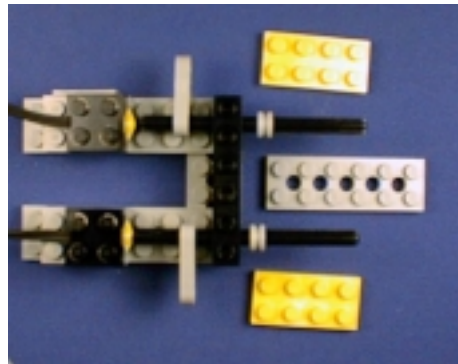
Step 1 Find two axles, a beam and the four smaller parts.



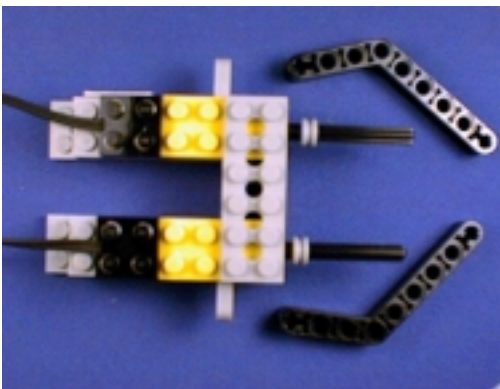
Step 2 – The axles slide in the holes in the beam.



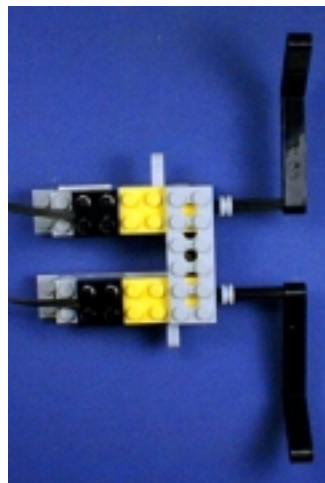
Step 3 – add the touch sensors



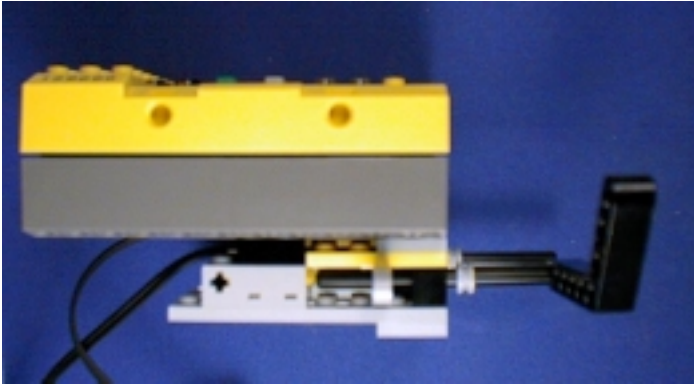
Step 4 – add these plates to complete the frame for mounting to the RCX.



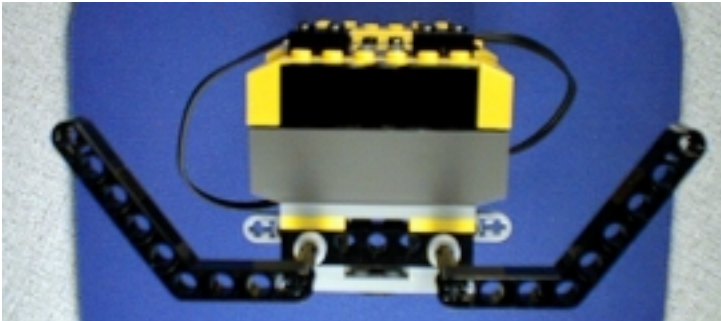
Step 5 – attach the “bumpers”



Step 6 – The finished sub-assembly.



The bumpers attached to the RCX.



And a front view. Add wheels and motors and your robot is ready to find its way around.

Introduction to RCX programming

The lesson plan for week 1 included an overview of the RCX programming environment. An RCX program is a sequence of instructions that controls how the robot responds to things it encounters in its environment. The robot gets information about its surroundings through sensors. It then makes decisions about how it is going to move. It senses, and reacts, just like we do. Programs are built by snapping together a series of RCX code blocks. Each code block represents a single instruction, and blocks are executed in order, from the top of the screen downward, forming a sequence of instructions for the robot to follow. A good overview of all the code blocks available in RCX can be found in the Mindstorms Robotics User Guide, starting on page 20, and also in the online help utility in RCX. To get to the help utility, click help menu when in the programming environment.

Sensors provide the RCX with information about the outside world. A light sensor provides the RCX with a number that indicates the intensity of light. Touch sensors inform the RCX whether they are 'pressed' or 'released', telling the robot when it runs into things. Timers provide the RCX with a number that indicates the amount of time that has passed. Rotation sensors provide a number that is a count of complete rotations of an axle. Programs can read these sensor values in order to make decisions about what to do.

Below is a brief description of the RIS code blocks that control how a robot behaves. The first thing to notice is that there are several types of code blocks that your kids will use:

1. Small blocks. Small blocks control motors, counters, and timers. You can set the direction and the power (speed) of a motor as well as turn it on, off, or on for a specific time. The internal counter can keep track of the number of times an event happens. The RCX also has an internal rotation counter that counts the number of times that a motor axle rotates. It is a method of keeping track of how far the robot has moved. The internal timer of the RCX counts in tenths of a second. The commands listed in bold below are included in the example programs that follow.

ON – turns a motor on by sending a signal through port A, B, and /or C.

ON FOR – same as above, except motor is turned on for a limited time.

OFF – turns a motor off by stopping the signal through port A, B, and / or C.

SET POWER – controls motor speed.

SET DIRECTION – controls the direction of rotation, and direction robot moves

REVERSE DIRECTION – reverses motor direction

BEEP – emits 1 or more beeps

TONE – emits 1 or more tones

RESET COUNTER – sets the internal event counter to 0.

ADD TO COUNTER – adds 1 to the internal event counter.

RESET ROTATION – sets the internal rotation counter to 0.

RESET MESSAGE – used to clear a message received from another RCX unit.

RESET TIMER – sets internal timer to 0.

SEND TO RCX – sends a number to another RCX unit.



2. Wait commands allow the program to pause for awhile or until an event occurs.
Note: Wait means that the program will wait. The robot may still be doing something! Think of how a person would behave. If he is commanded to walk forward, he will walk forward until he receives the next command! WAIT means don't issue another command for awhile.

3. Repeat blocks. give your program flexibility. They enable the program to execute a given command repeatedly, or to execute different sets of instructions for different circumstances the robot might encounter.

REPEAT – repeat a series of code blocks a specific number of times.

REPEAT FOREVER – repeat a series of code blocks indefinitely.

REPEAT WHILE – repeat a series of code blocks while a sensor is reading a given value.

REPEAT UNTIL – pause further program execution until a sensor achieves a specified value.

4. Yes or No allows a decision yes / no decision to be made, and then the program proceeds down one of two paths.

5. Sensor watchers. Stay away from these!! Only use the variable watcher, and only use it if you get a message that the program has too many blocks in a stack.

The problem is that once a sensor watcher is set up, there is no way to stop it from watching that sensor.

A series of code blocks are attached to sensor watchers. When the corresponding sensor reads a specific value (or falls within a specific range), the code blocks attached to the sensor watcher are executed. Afterward, code blocks following the sensor watcher in the main program are executed.

Variable – monitors for a change in the value of a variable.

TOUCH – allows for execution of one stack of code blocks when a touch sensor is pressed, and another when it is released.

LIGHT – allows for execution of one stack of code blocks when a light sensor value falls within one range of values, and another stack of code blocks when a light sensor value falls within another range.

ROTATION – executes an attached series of code blocks when a specified number of rotations is reached.

TEMPERATURE – executes one stack of code blocks if temperature falls within one range, and a second block of code if temperature falls within another range.

COUNTER – executes one stack of code blocks when the counter falls within a given range.

TIMER – executes one stack of code blocks when the timer is within a given range.

These are all the basic building blocks that are put together in different combinations to create programs in the RCX programming environment. To gain some familiarity with how some of these code blocks operate, work through the following example programs.

Write a program that turns the robot in place

You may want to have the lesson from week 1 handy to help you through these steps.

A robot is turned by giving different commands to the left and right motors. Motors can be driven in different directions, at different power (speed of rotation), and for different amounts of time (controlling the duration of the turn and the ground traveled). If the left motor rotates forward and the right motor backward at the same speed the robot will turn in place. If both motors rotate in the forward direction, but at different rates, the robot will move through an arc.

This program will look a lot like the one in lesson 1, described in ‘Add a turn to your program’, which is illustrated in that lesson.

- ① Plug one motor into port A and the other into port C.
- ② Start with a **set direction** tile to move motors A and C in opposite directions.
- ③ Add a **set power** tile to control the speed of motor rotation. Begin with equal settings for motors A and C.
- ④ Next, turn the motors on for a specified time. Add an **on for** tile, select motors A and C, and indicate a time value. Remember the time is indicated in tenths of seconds (this is because the motors can rotate very quickly and a lot of turning can be done in a short time). Enter values for power and time that you think will cause the robot to turn (change its heading) 180 degrees while staying in place.
- ⑤ Download the program you have written to the RCX and execute it. How good was your guess?
- ⑥ If the robot turned too much or too little, adjust time in the **on for** tile appropriately. Download the program to the RCX and execute again, taking note of the results. Repeat until desired turn is achieved.
- ⑦ The combination of power and time controls the turn. Once the robot is able to make an accurate 180-degree turn, adjust the power level and repeat steps 4 – 6 to find a new combination of values that work.

One thing to note is that these values can change depending on how new the batteries are. You’ll probably have to experiment to find good values for the different stages of battery life.

☺ **Congratulations!!** ☺

Now you can turn the robot on a dime when space is tight.

Write a program that moves the robot through an arc

When a car turns, it doesn't turn in place, but actually moves through a curve (arc). That means it changes its direction (heading) a little at a time while also moving forward (or backward).

- ① Add a block to set motors A and C to move in the same direction.
- ② In this example, we want the two motors to turn at different speeds. In order to do this, you need to use two **set power** blocks (each block can set 1 or more motors to the same speed only). Add the blocks and set each motor to a different speed. Can you guess what direction the robot will turn?
- ③ Next, turn the motors on for some time of your choosing.
- ④ Download the program you have written to the RCX and execute it. Did the robot perform as you expected?
- ⑤ If the robot turned too much or too little, adjust the time that the motors are on for. Download the program to the RCX and execute again, taking note of the results. Repeat until desired turn is achieved.

😊 **Congratulations!!** 😊

Now you can turn the robot in an arc.

Use “Try It” to get sensor values for ‘light’ and ‘dark’.

You will use light sensors to find out if the robot is over a ‘light’ or ‘dark’ surface. The light sensors output a value from 0 to 100 depending on the intensity of light present. A light sensor will produce a different value when reading the same surface under different lighting conditions. It is up to the user to determine the sensor values that are to be considered ‘light’ and ‘dark’. You will need the surface on which you will test the robot. This can be the test pad that came with the Mindstorms kit or black tape on the floor.

1. Attach a light sensor to input #1 on the RCX brick.
2. Start the RIS 2.0 Robotics Invention System software and click “Program” and “Freestyle” to begin a new program.
3. Add a **Wait Until** block to the program. It should look as follows:



4. Open the **Wait Until** block and change it from a “wait until touch” to “wait until light”. Click **NEXT** until you arrive at the screen with a “**TRY IT**” button. Click “**Try It**” and you will see the following window:



5. Point the light sensor attached to the RCX brick at various surfaces. Make sure that the RCX is within range of the transmitter.
6. Note how the light sensor value changes as you move the sensor over darker and brighter surfaces.
7. Determine a cutoff value half way between average light and dark readings.
8. Notice how the pointer on the window moves from “Bright” to “Not Bright”.
9. If the pointer doesn’t move, click **BACK** and alter the cut-off value.
10. Unplug the light sensor. Notice how the pointer moves to “No Reading”.


You now know how to interpret sensor readings. Readings above the cutoff value can be interpreted to indicate a light surface, readings below this value a dark surface.

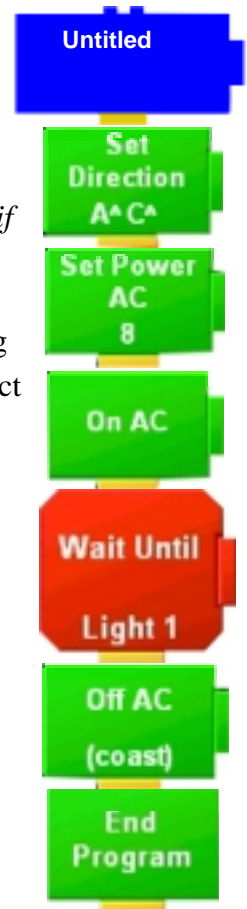
😊 **Congratulations!!** 😊

You now understand that the numerical sensor reading changes when different surfaces are 'read', and also when the amount of light is changed.

Write a program that moves the robot forward until it moves across a black line and then convert this program into a 'my command' block.

This example has two objectives. The first is to put together a sequence of code blocks that will move the robot forward until it crosses a black line and then stops. The second objective is to convert this series of code blocks into a 'my command' block. You can imagine circumstances in which it would be useful to have the robot move forward until it crosses a black line. Once a 'my command' block has been created to do this simple task, it can be easily reused without having to recode all the steps. You will need a robot that has a light sensor on the front of the robot pointing toward the ground attached to port 1.

1. Begin with **set direction**, **set power**, and **on** blocks to move the robot forward by activating two motors connected to ports A and C. 
2. When trying to decide which stack controller to use, form a sentence to express how you want the robot to behave. This will often tell you which controller is appropriate. In this example, we want the motor to move forward until it reaches a black line. You can see the word *until* in the sentence which tells us that a **Wait until** command is appropriate. Another example would be 'If the counter indicates that 3 lines have been crossed then turn right, otherwise go straight.'. Whenever you use *if then*, this is an indication that **YES or NO** is the appropriate controller.
3. Add a **wait until** block to your stack. Select the light sensor icon among the choices given to you (its the blue brick), and click **NEXT**. Then select the number 1 to indicate that the sensor is attached to port 1.
4. Specify a light sensor value in the **wait until** block.. The program will wait until the light sensor value goes past this value, then it will execute the code following the **wait until** block. Because the motor is on, the robot will continue to move forward while waiting.
5. Add an **off** tile after the **wait until**.



☺ **Congratulations!!** ☺

You have created a program to move the robot forward until it crosses a black line.

Make this stack of code into a ‘my command’.

1. Under the **my commands** menu, select **Create New My Block**.
2. Give the command a name, such as **FWD2LINE**.
3. Drag the stack of code you created between the **FWD2LINE** yellow blocks.
4. Click the small gray square in the upper left corner of the **FWD2LINE**. You will notice that the other commands disappears (if you want it back, click the gray square again).
5. Drag the **FWD2LINE** underneath the **blue untitled block**.
6. Download and test your program.
7. If you need to adjust the light values, click on the gray square to view your code and adjust accordingly. You can always edit your **my command** (just like any other program) by dragging that tile onto the screen.
8. Look in the **my commands** menu. You will see that there now exists a **FWD2LINE** command. This command can be used just like any other command such as **on for** or **set power**.

By packaging code into these **my commands**, you make programs easier to read, write, and debug. Imagine if you have multiple programs that all have to go forward until a black line is crossed. Having the **FWD2LINE** command makes it easier to add this unction to each of the programs and makes it easier if the lighting conditions change. You simply have to change the value in one location instead of in each program. Additionally, when you start building big programs, you will find by referring to a stack of code with a single block makes the overall program much easier to read and to understand. *Code modularity* refers to this packaging of stacks of code into a single command (sometimes referred to as a *subroutine*).

Write a program that moves the robot backward until its touch sensor is hit and then convert this program into a ‘my command’ block.

Similar to the last example, we could use the **wait until** block to write this program. However, in this example we will use the sensor watchers. When using sensor watchers, you cannot take the code and make a **my command** with it. For that reason, in this situation you might prefer to use the **wait until** block, but we will use the sensor watcher here to learn how they work. It is often the case in programming that there are multiple ways to achieve the desired behavior. When trying to decide which way to program, think about what will be the easiest to read, to write, and to debug. Simplicity is key when programming.

Note: Sensor watchers (the blue commands) are of little value in the FLL challenge. Once a sensor watcher is set up, the author knows of no way to tell the watcher to stop watching! Normally your kids will want the light sensor to watch for a black line, then move on to more commands. With a sensor watcher (the blue commands), the watcher keeps grabbing control every time a black line is crossed, which is not desirable! If your kids want to pursue this, there is a command **Set Priority**, which appears to give you some control of the sensor watchers. But there are easier ways to use sensors, such as the **Wait Until** and **Repeat Until** and **YES or NO** commands.

- ① First, drag the **FWD2LINE** block to the trash so that you can start with a blank program.
- ② Begin with a **reverse direction** block to move the robot in the opposite direction. Note that reversing the direction doesn't always mean backwards. If the robot is already moving backwards, then a **reverse direction** block will make the robot move forward.
- ③ Add an **on** block to get the robot moving.
- ④ Go to the **sensors** menu and find the touch sensor. Drag the block *beside* the blue **untitled block** (not underneath). Make sure that the port number selected on the sensor watcher is the same number that the sensor is plugged into.

The **sensor watcher** is put beside the **program block** because it *runs in parallel* to the regular program. *Runs in parallel* means that it runs at the same time. When code blocks are put one on top of the other such as underneath the **program block**, the code is said to *run sequentially*. *Runs sequentially* means that one command has to be complete before another one starts. What happens is that the code under the **program block** begins to execute. If the sensor watcher condition is met (e.g. the touch sensor is pressed), then the **program block** code will be interrupted and the code underneath the met condition (e.g. **press**) will be run sequentially. Once all the code under the met condition is done, the code under the **program block** will continue where it left off. This may be difficult to grasp. If you don't understand, ask someone to explain it further.

- ⑤ Place an **off** block underneath the **press** side of the **sensor watcher**. Notice that there are two places underneath the sensor watcher to add code. Here we are only concerned with what happens when the sensor is pressed. One scenario where you

might use the **release** side is when you have an arm that holds on to something. You can use the touch sensor to know if the robot has dropped the object or not.

- ⑥ Download and test your program.

😊 Congratulations!! 😊

You have created a program to move the robot backward until its touch sensor is bumped.

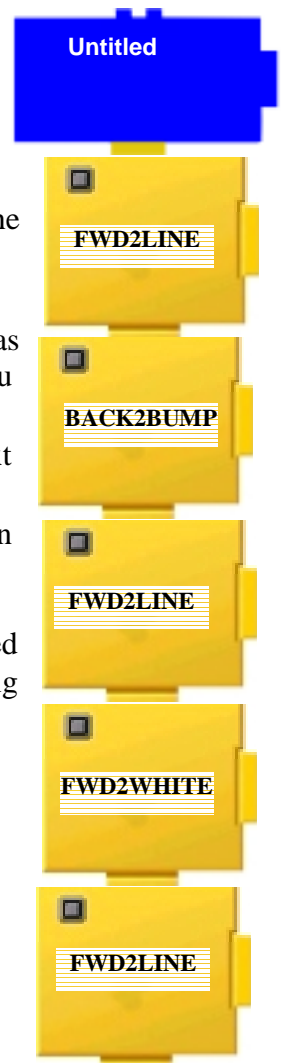
Now, let's put it all together! Write a program that moves the robot forward until it crosses a black line, then moves backward until it's bumper sensor is hit, then moves forward until it crosses 2 black lines.

- ① First, drag all the blocks to the trash so that you can start with a blank program.
- ② Begin with a **FWD2LINE** block to move the robot to the first black line.
- ③ Add the appropriate blocks to get the robot to move backward until the bumper is hit. Use whichever blocks your group decides on.
- ④ Add the appropriate blocks to move the robot to the first line once again.
- ⑤ Add blocks to move the robot over the first line. *Hint*: this is the same as moving it to a black line except you start with black and stop when you reach white.
- ⑥ Add the appropriate blocks to your program to move the robot to the next black line.
- ⑦ Download and test your program. Save it if you like. (To test it, you can use the test pad or use black tape on the floor.)

The figure below is a sample program that performs the task as explained above. Notice that only **my commands** were used. This makes reading the code much easier.

This is a program that, when executed, causes the robot

1. to move forward to the first black line,
2. reverse direction and move until its touch sensor is hit,
3. move forward to the first black line again,
4. move over that black line, and
5. go to the 2nd black line and stop.



😊 Congratulations!! 😊

You have completed this lesson. This lesson covered a lot of material. You did a great job of getting through all of it!

I hope you had fun.