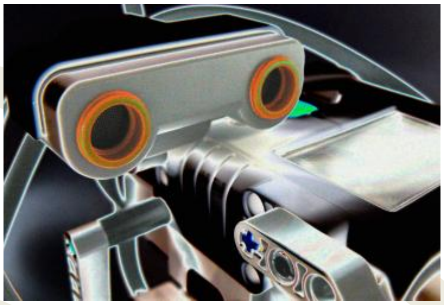
Workshop – **NXT Programming For Beginners**

Version 1.1

Summer, 2012



Based on the Lego NXT Robotics System

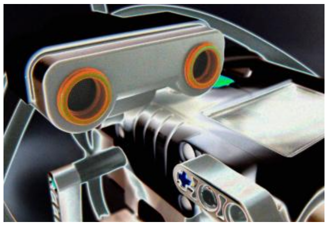
Neil@vectorr.com 828-484-4444
Weaverville, NC

This workshop is based on the Lego NXT Mindstorm robot and programming tools which are the intellectual property of Lego, Inc. to whom I give great thanks. No special rights to their materials is expressed nor implied by use or dissemination of this workshop.

When in doubt, contact the author:

Neil Rosenberg
106 Thistle Tree Lane
Weaverville, NC 28787
828-484-4444
Neil@vectorr.com

Thanks, and ENJOY!



Workshop – NXT Programming for Beginners

Beep!

Welcome to the "**NXT Programming for Beginners**" workshop, a fun hands-on learning adventure. You decide how fast to proceed, and you'll have plenty of chances along the way to take side-trips if something looks interesting to you. This workshop is particularly useful to students, mentors and parents who are involved with a ***FIRST***™ **Lego League (FLL)** team and want their robot to perform well in the competition.

- **This will make you think** -- Be prepared for some real challenges; believe it or not, failure is a GOOD THING! It's your chance to figure out a better way.

- **Get ready to program your own robot –** You will be using the powerful Lego NXT robot with its Graphical User Interface (GUI) programming language. You are in command -- for better or worse the robot will do exactly what you tell it to do.

- **Step-by-step is the best way** -- Skills you develop in one project are used in the projects that follow. So it's really best to do all of the projects in order (they each have a number). That way you won't get lost wondering how to do something that looks impossible.

- **This is only the beginning –** There's LOTS more that NXT can do. As you advance you'll want to check out multi-threading, re-usable MyBlocks, calculations, switch statements based on data and more. There are many educational resources and great NXT projects available on the web!

Note: Some of the wording and/or concepts in this material may be a bit advanced for youngsters -- this workshop is designed for adult mentors working one-on-one with students. If a student wishes to do it on their own, make sure they are reading at an adequate level and have an adult to whom they can ask questions.

Each project also has a difficulty rating, ranging from one to five "Einsteins". Look for the little icons in the upper right corner of the project sheet:

Easy Peasy.

A bit more challenging.

Put on your thinkin' cap.

Get ready for Brain Strain.

What’s needed

- LEGO MINDSTORMS Education NXT Base Set, model 9797
- Fully charged battery.
- NXT 2.0 Software, model 4558560 (included in 8547 kit). Available free from Lego at:
  http://service.lego.com/en-us/HelpTopics/default.aspx?questionID=2655
- PC running Windows XP, 2K, Vista or 7
- Other supplies per project, such as markers, black electrical tape, 2’x4’ white marker board, ruler/protractor/tape measure.

Step 1:

Open your Lego Mindstorms kit and take inventory

If you have a new kit, you’re in great shape. If you have a borrowed, hand-me-down or other non-new kit, you’ll need to make sure you have a full set. If it’s like most Lego kits, there’s no telling how many important parts may have been misplaced.

1. Find the sheet that describes all of the parts that are supposed to be in your kit. This is printed on a large sheet of cardboard. You can also find listings of the parts on-line, for example at:

   http://static.robotclub.ab.ca/pages/nxt/InventoryComparison/nxt_1_vs_2_vs_edu.html

   To see a photo of all of the specific parts needed for a FlexBot see the last page of Project 1.

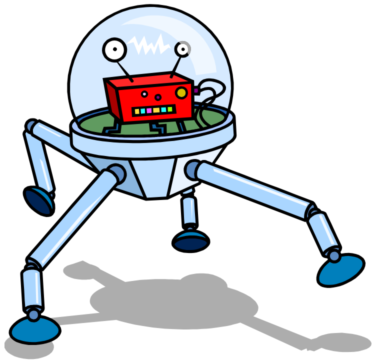
2. Follow the instructions to insert the battery pack and power up your NXT Brick.

3. Install the NXT 2.0 or NTX-G program on your Windows PC. Once you’re convinced it’s running and you can communicate with your NXT Brick, you’re ready to get going.

If you have any problems with your robot, computer, cable and etc. ask your workshop leader to help. It’s usually something really simple.

Step 2 and beyond:

Get the activity sheets for Project 1 and build your FlexBot. This unique design is custom made for all of the projects in this workshop. Once it’s built and tested, you’ll start your programming adventure in Project 2.

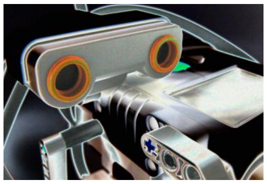


The Projects

Project Number	Title	Description	Difficulty
1	Build your FlexBot	Use your Lego pieces to make a special robot	☺
2	Sound OFF (and ON)	Learn about steps in a program, make some music	☺
3	Straight on till Dawn	Baby's first steps out into the real world	☺
4	A"maze"ing Turns	Sometimes the only way to get ahead is by going sideways	☺ ☺ ☺
5	Simple Sensors	Detecting the world around the robot	☺ ☺
6	Stop at the Line	First "exposure" to using a line sensor in your programs	☺ ☺ ☺
7	Using MyBlock	How to keep your programs small and simple	☺ ☺
8	Running Ragged (some call it "suicide")	A bit more structure to your program as you navigate from line to line	☺ ☺ ☺
9	Automatic Sensor Calibration	More advanced use of data	☺ ☺ ☺
10	Line Following	Follow the Black Brick Road (sorry Dorothy)	☺ ☺ ☺ ☺
11	Final Challenge	Putting it all together	☺ ☺ ☺ ☺

Whatever we don't finish in the workshop, do at home!

Select algorithms based on code courtesy Dave Parker of NXTPrograms (www.nxtprograms.com)



NXT Programming for Beginners

Project 1:Build your FlexBot

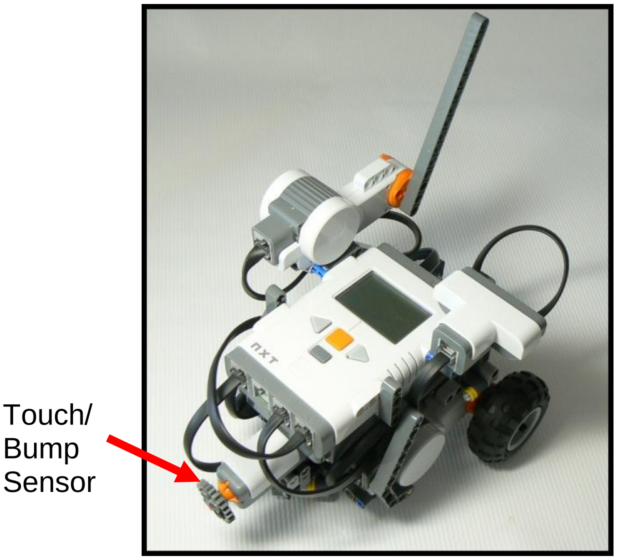
Use your Lego pieces to make a special robot!

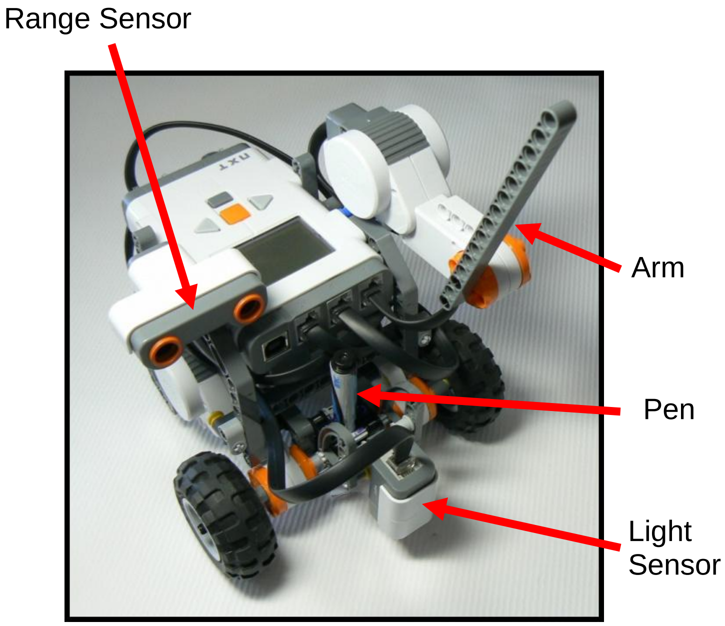
Chances are you've built with Legos for years, and that's **GREAT**. However unless you're experienced with the NXT Mindstorm kit, the parts you're about to use will seem pretty different. In fact, the FlexBot doesn't use any old-style Lego pieces (you know -- *bricks*), instead it's made entirely of new generation NXT parts. The good news is that your robot will stay together pretty well, much better than your Lego "masterpieces" used to. Unless of course if you drop it on the floor!

The robot you build in this project has lots of really cool abilities, all of which are used in this workshop. The main parts of the FlexBot are:

1. The NXT "Brick" that you program.
2. Two motors/wheels plus a caster so your robot can move forward, back and turn.
3. A "Pen Slot" located directly between the two drive wheels. This allows you to put a marker into the robot so that it will draw lines as it moves around.
4. Light, Range and Touch sensors that allow your robot to observe and react to the world around it.
5. An articulated arm to move objects.

Here are photos of the finished FlexBot, viewed from each end. Notice the Pen, Arm and three Sensors:





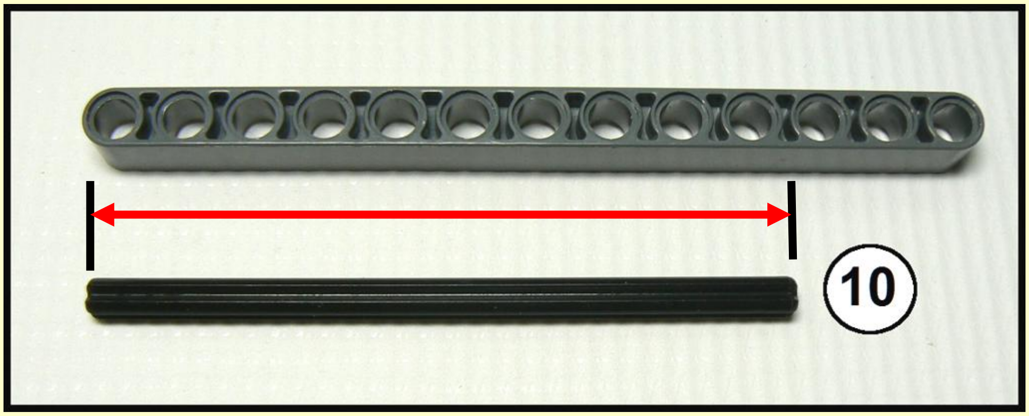
In the following steps you build your FlexBot from pieces found in your "LEGO MINDSTORMS Education NXT Base Set, Model 9797". If you find that a certain part is missing you will need to either find it (highly preferred), or invent a substitute way of assembling your robot with parts that you do have. ***The color of parts in your kit may be different than what's shown in the photos. ADAPT!***

The FlexBot is carefully designed and tested to work with this workshop. Building with parts other than those specified or assembled in a way not shown can cause problems with the projects. See the last page of this project for a photo of ALL of the parts needed for one FlexBot.

Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

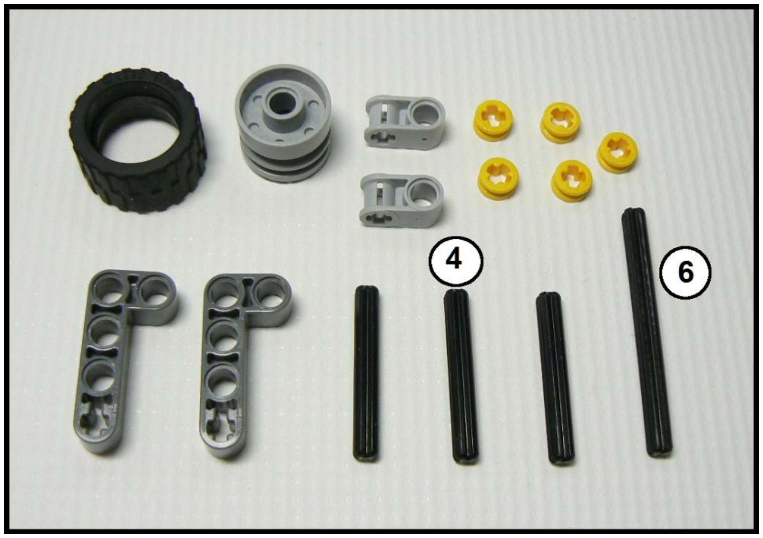
Builder's Note: When choosing parts from your bin, the way to get the right length part is to count the number of holes or bumps in the picture and compare it with the part you're considering.

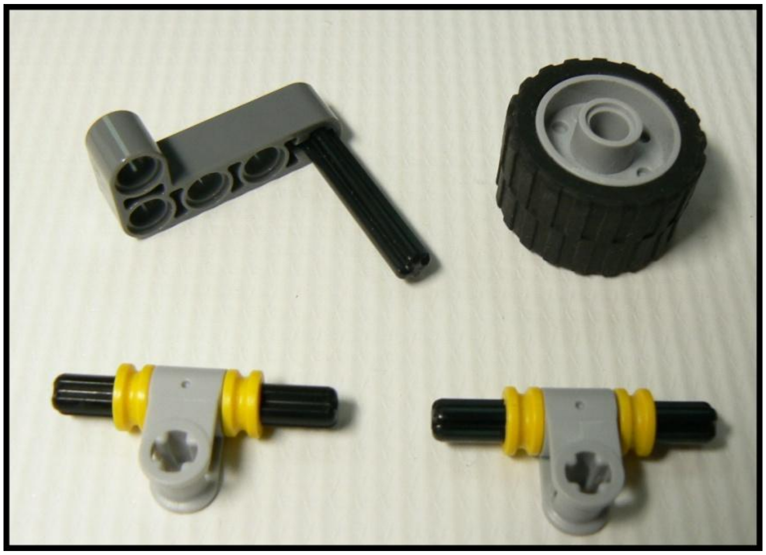
For certain parts, like shafts, you will often see a number next to it in the picture:



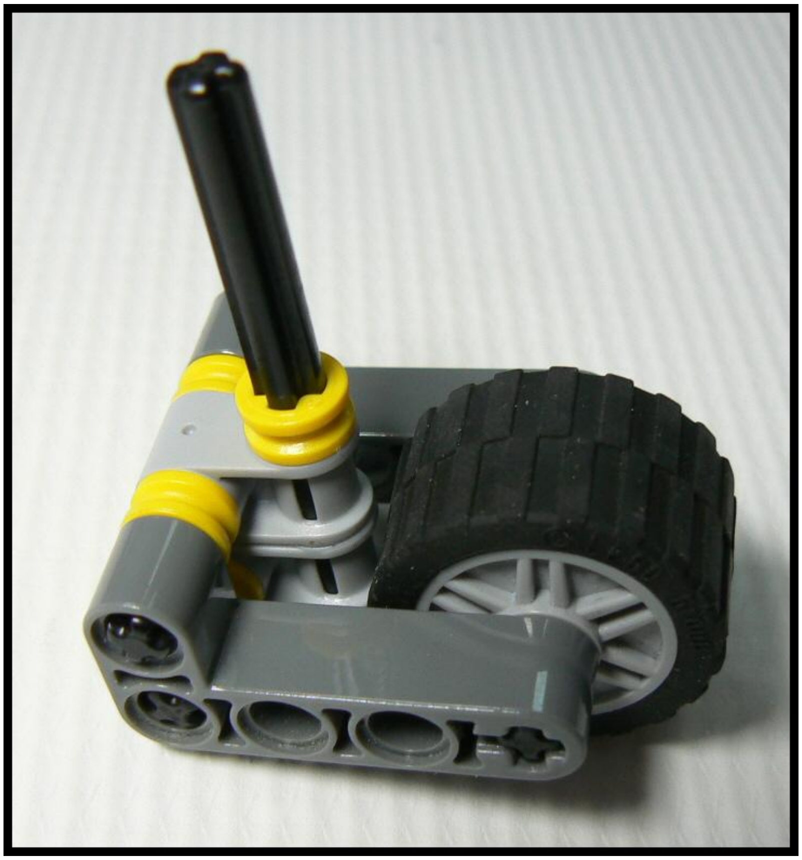
This means that the shaft is "10 Holes Long" (count them). It does NOT mean that you need ten of them. Some experienced builders keep a long piece handy, like the one at the top, and use it to compare with shafts to help find the right one.

□ **1.** Locate the following parts, and assemble the Caster as shown:

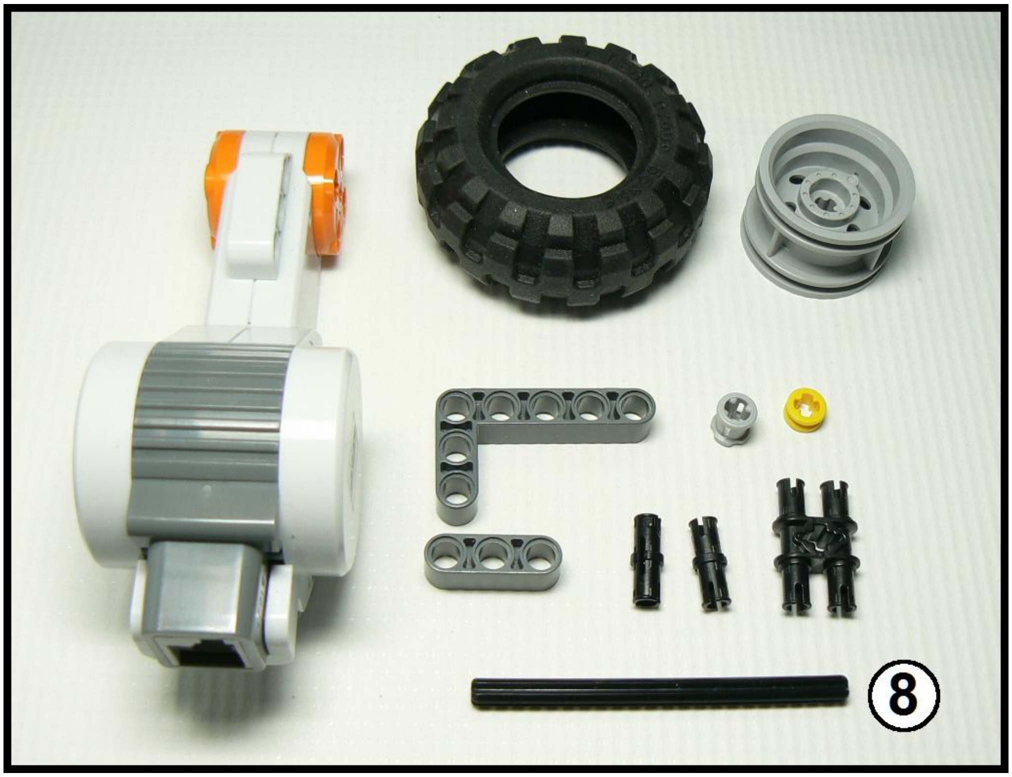


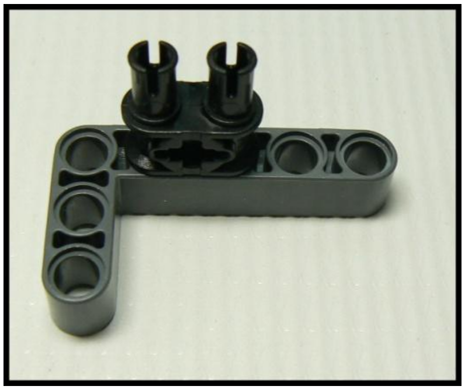


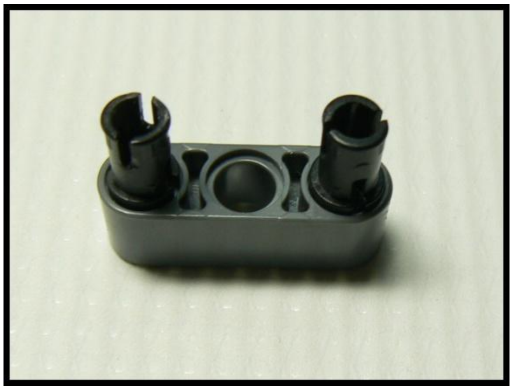
The finished Caster should look like this:

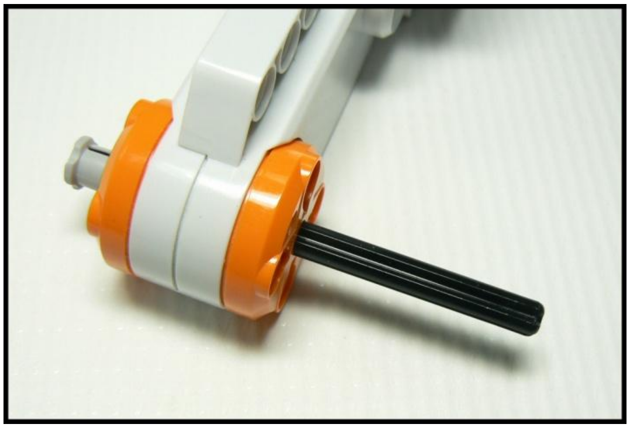


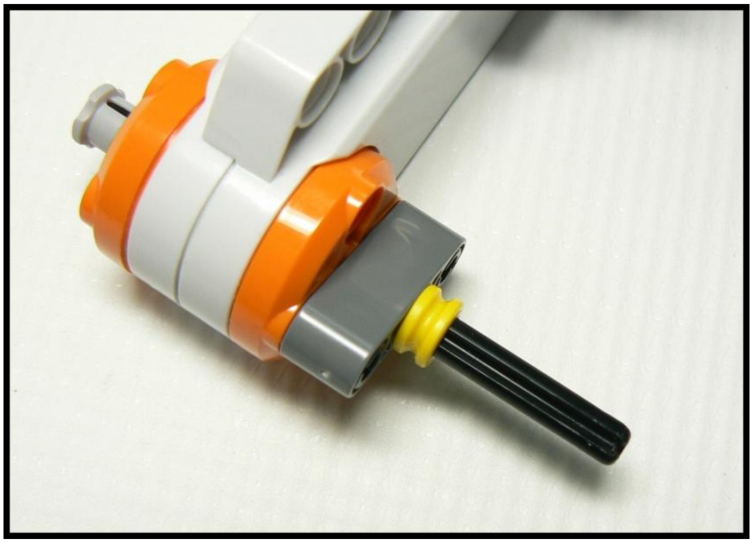
□ **2.** Locate the following parts, and assemble a Motor and Tire as shown:

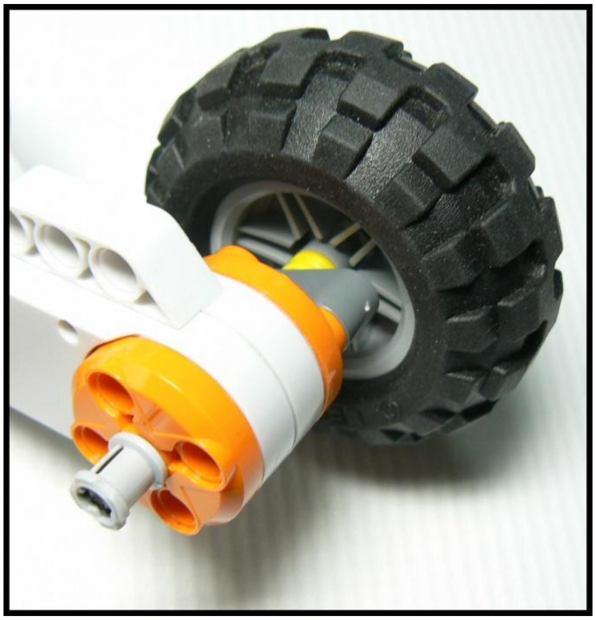


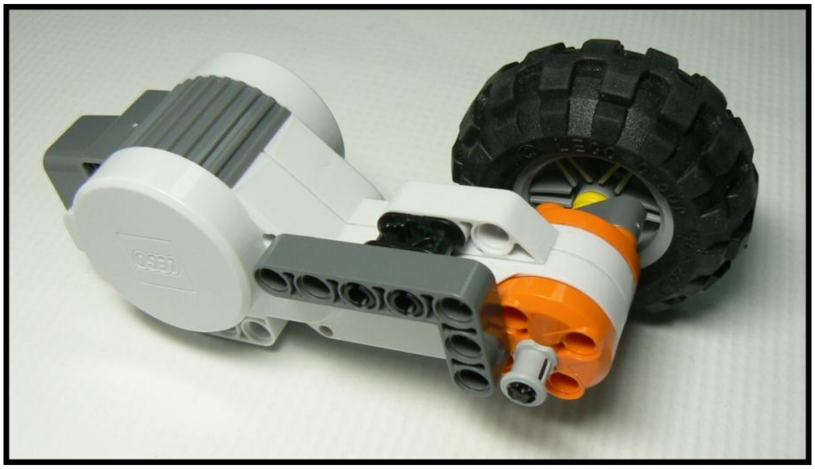




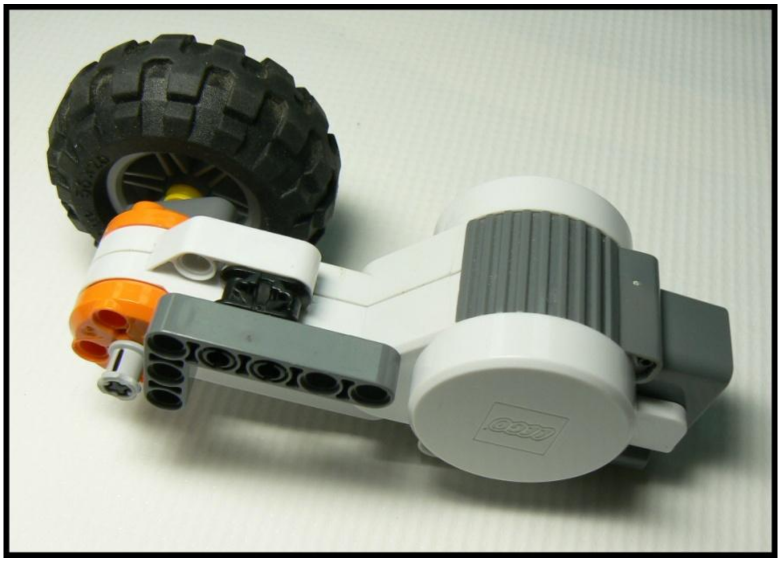




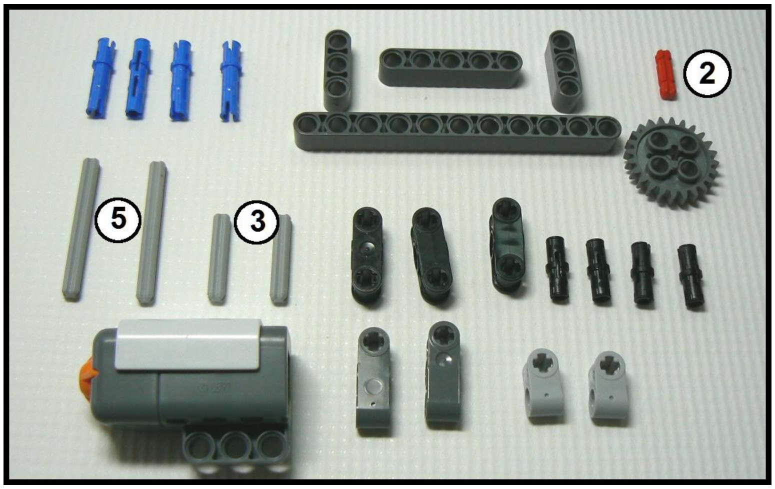


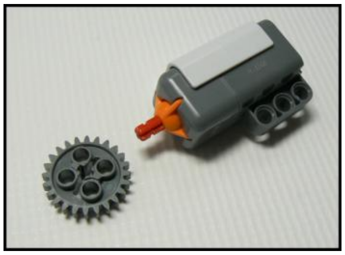


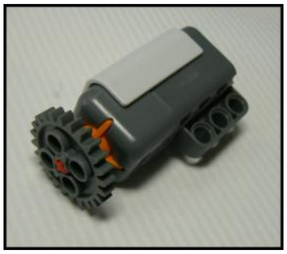
□ **3.** Using the same parts as the step 2 above (but in mirror image), assemble the other Motor and Tire as shown:

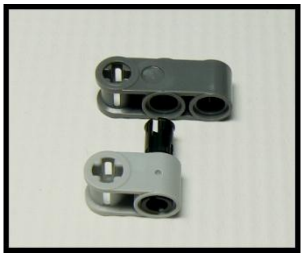


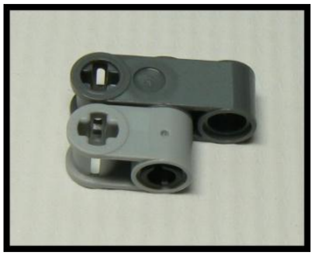
□ **4.** Build the Bump/Touch Sensor and Caster assembly:



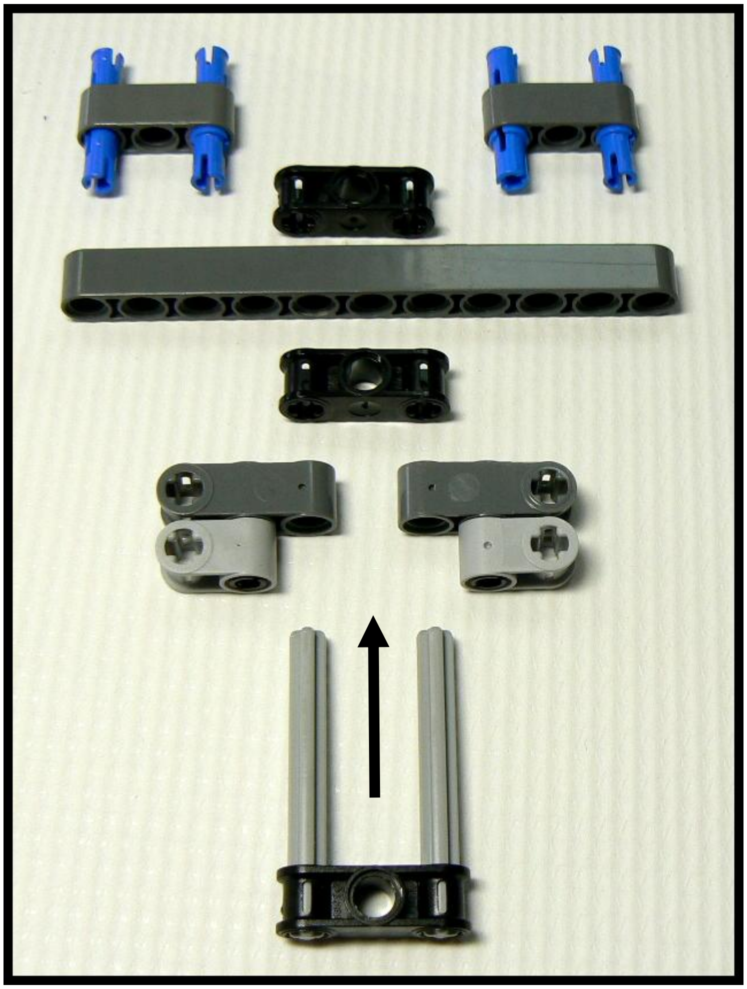


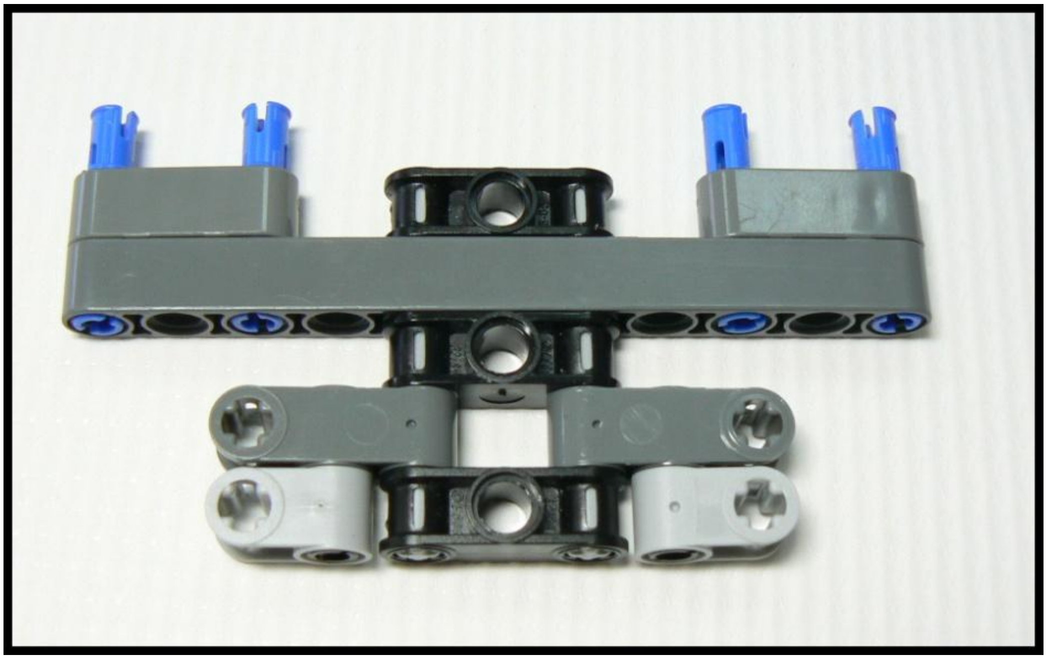


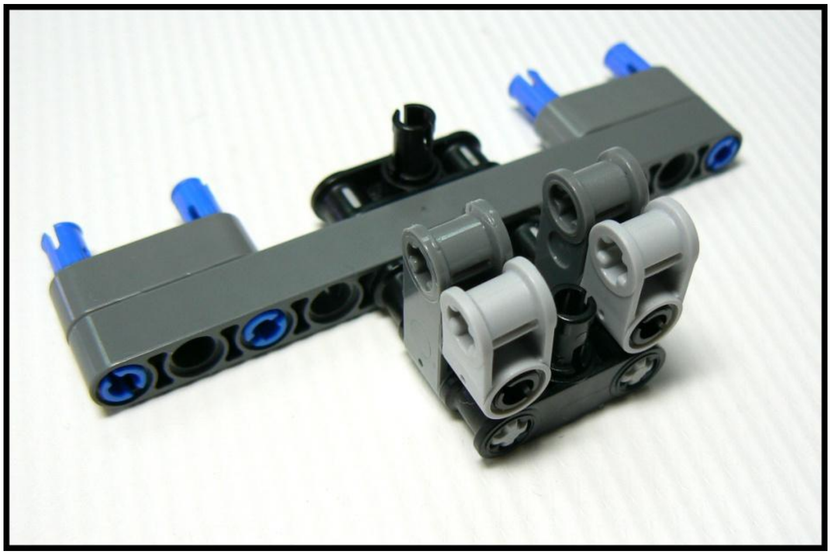


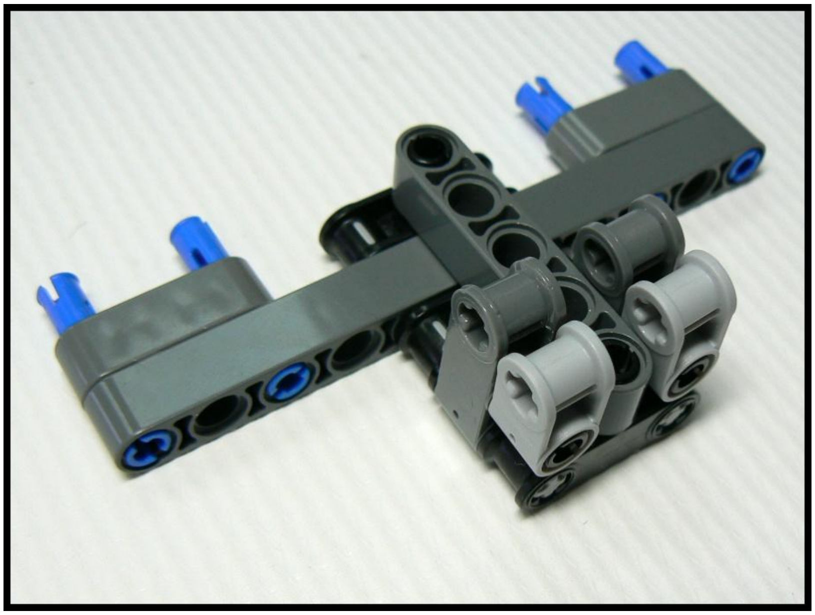


Build the above x2

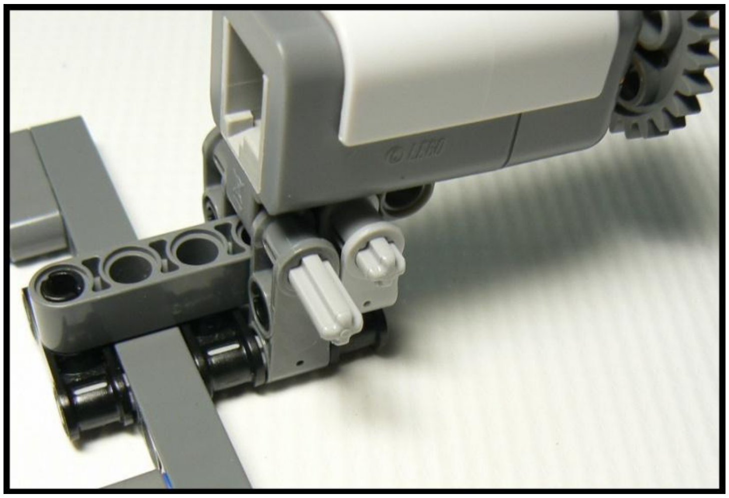


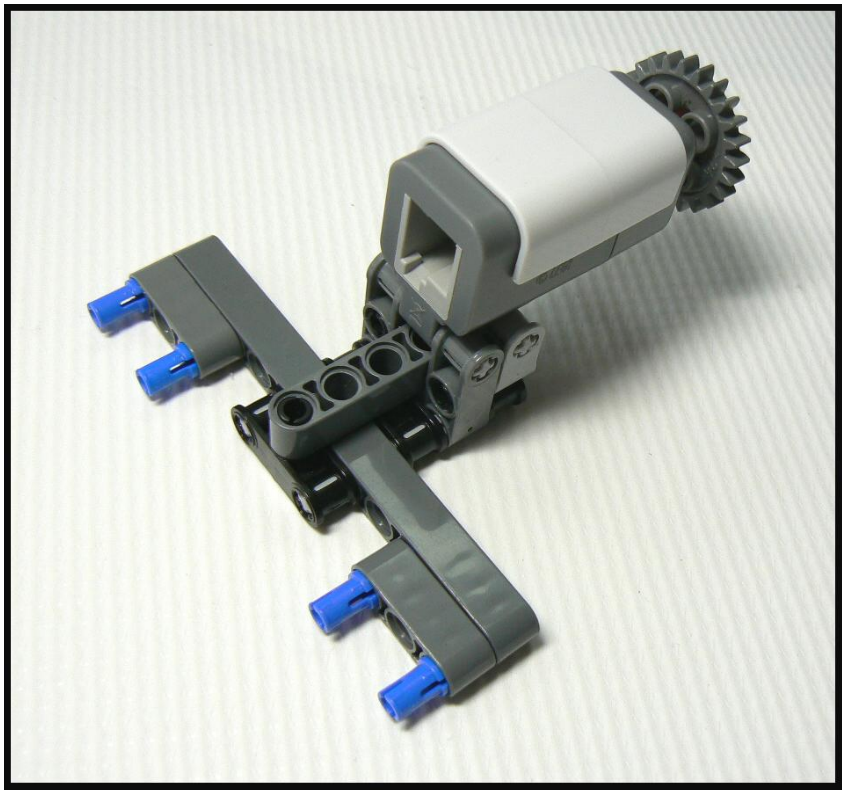


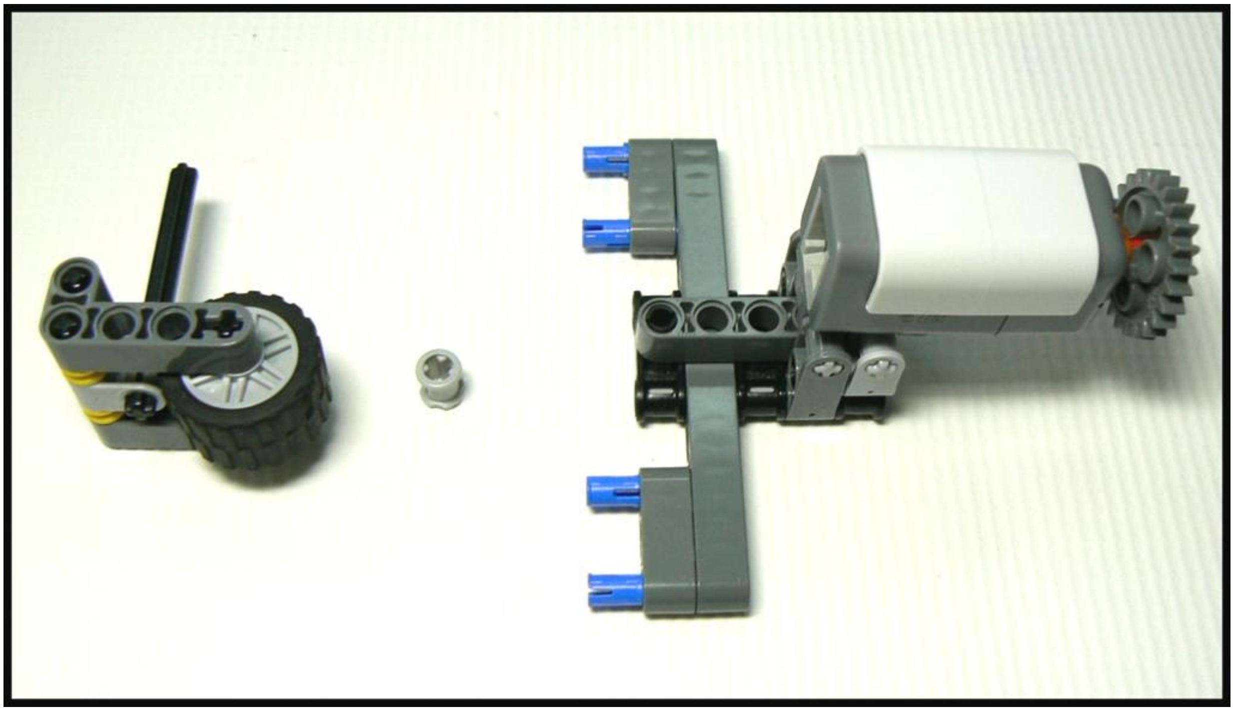


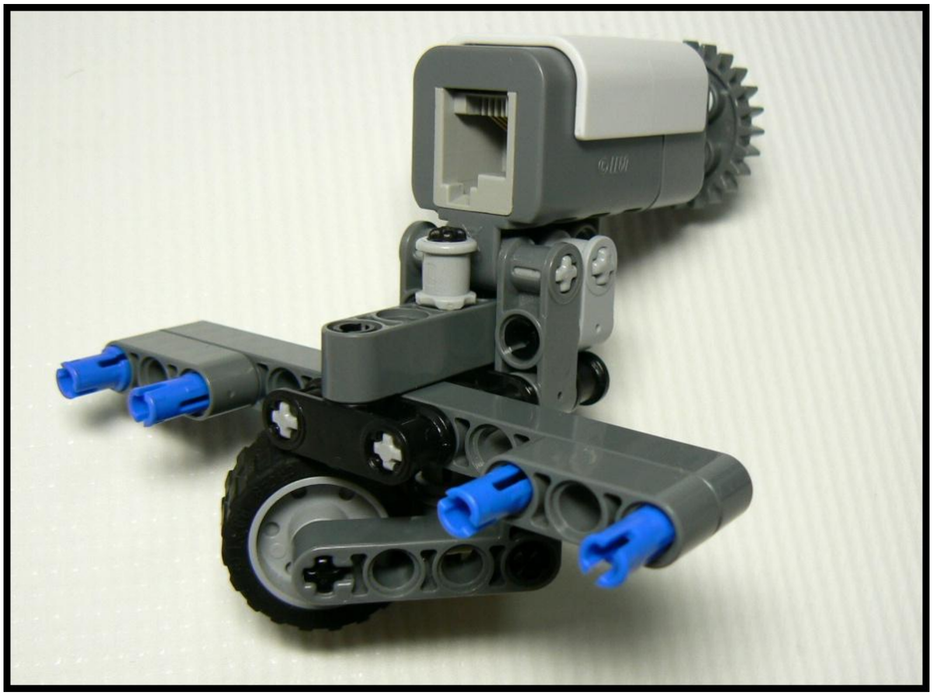


To attach the Bump/Touch Sensor, hold it in place with two shafts (shown partly inserted):

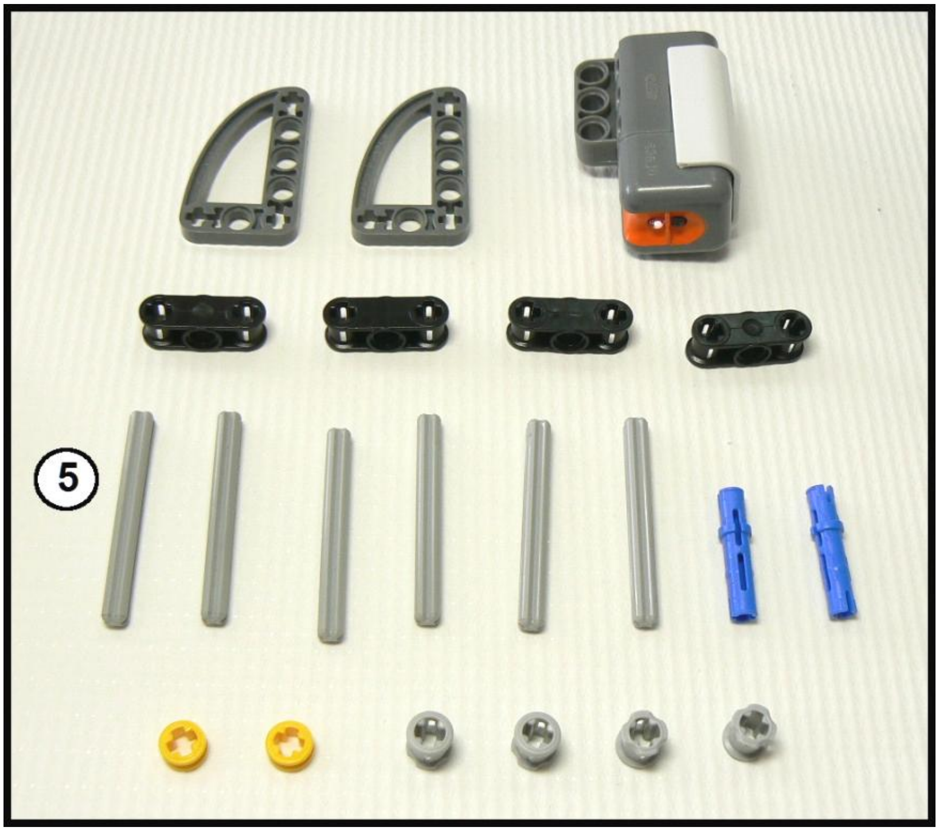


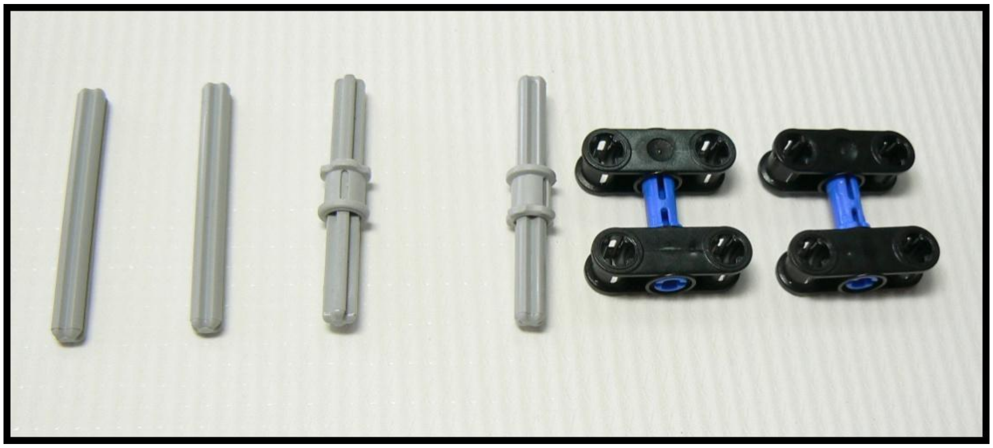


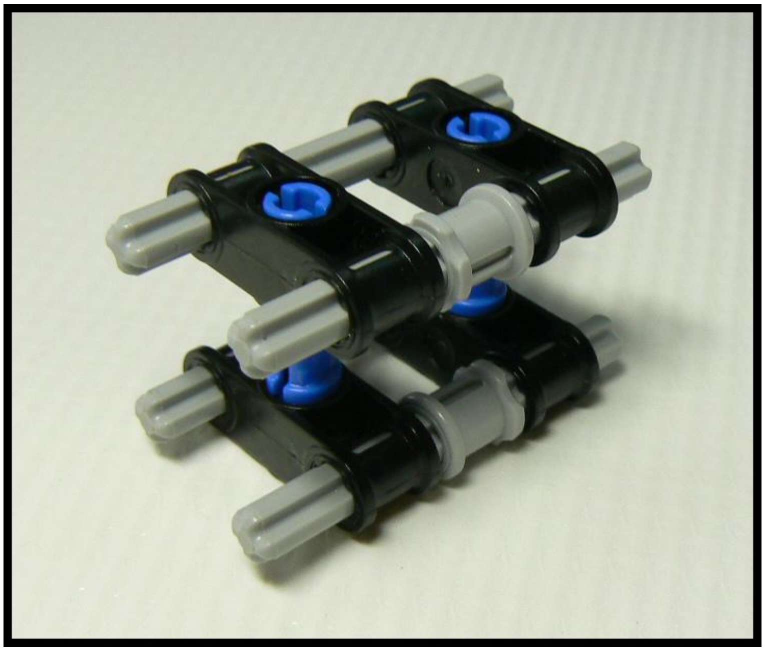


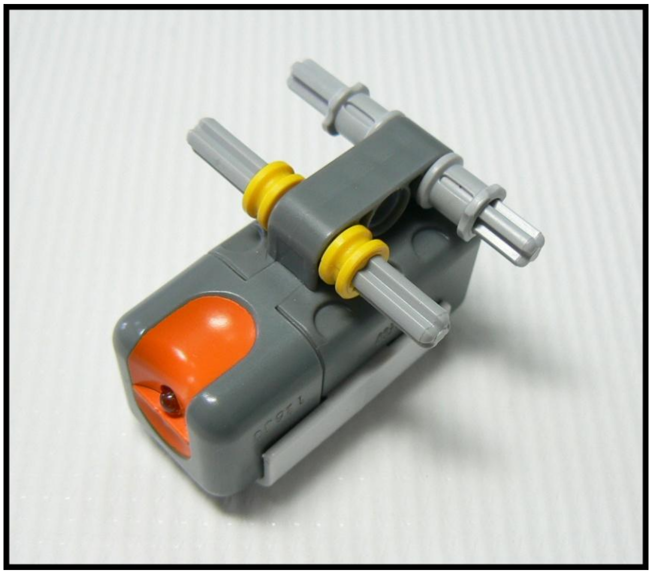


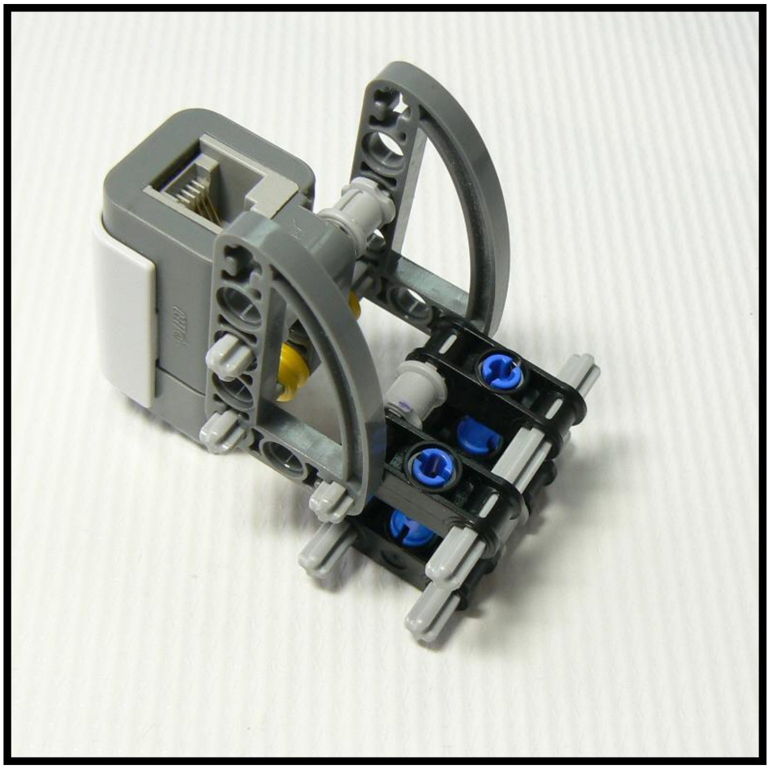
□ **5.** Build the Light Sensor / Pen Slot assembly:



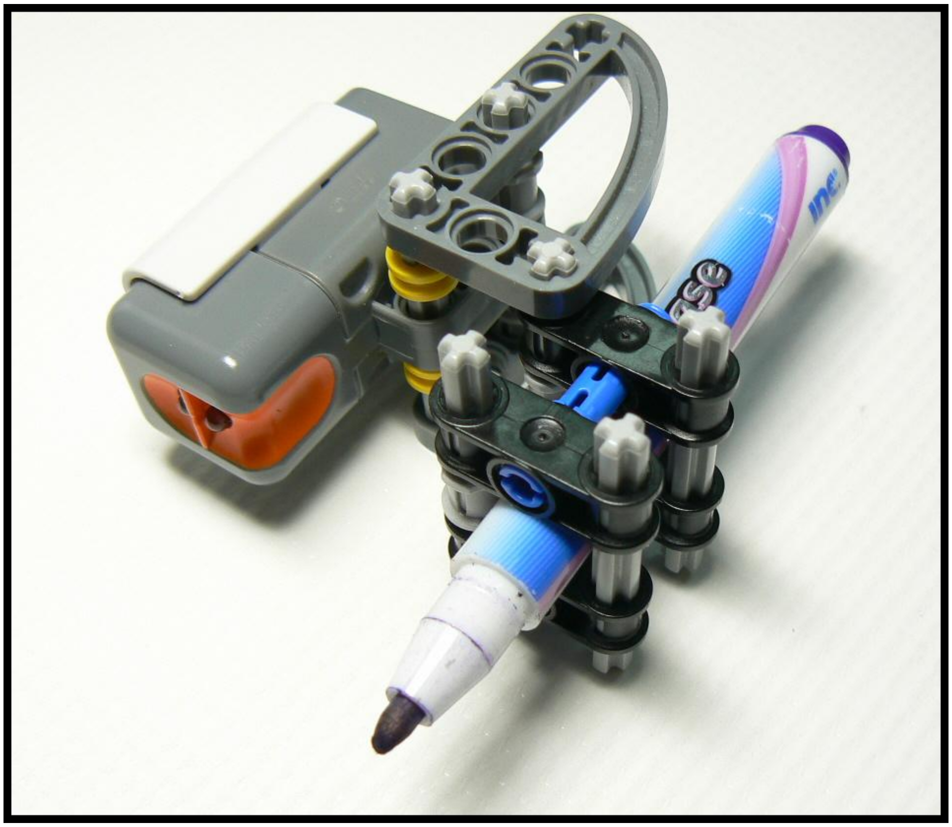




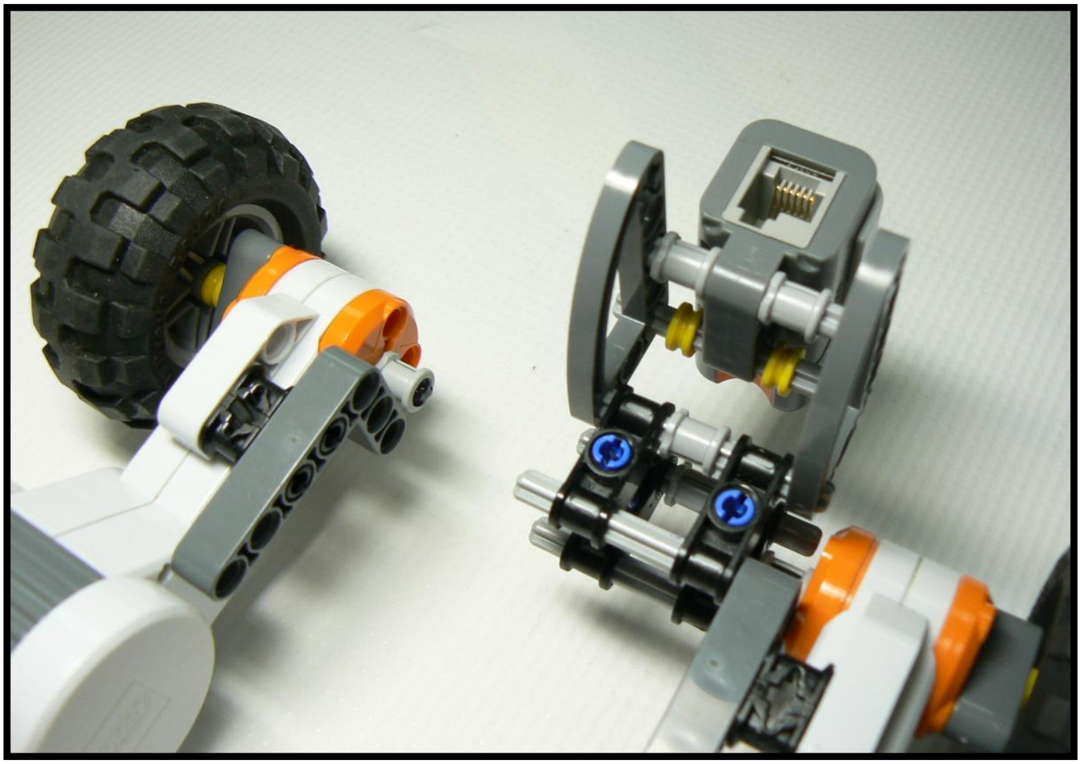


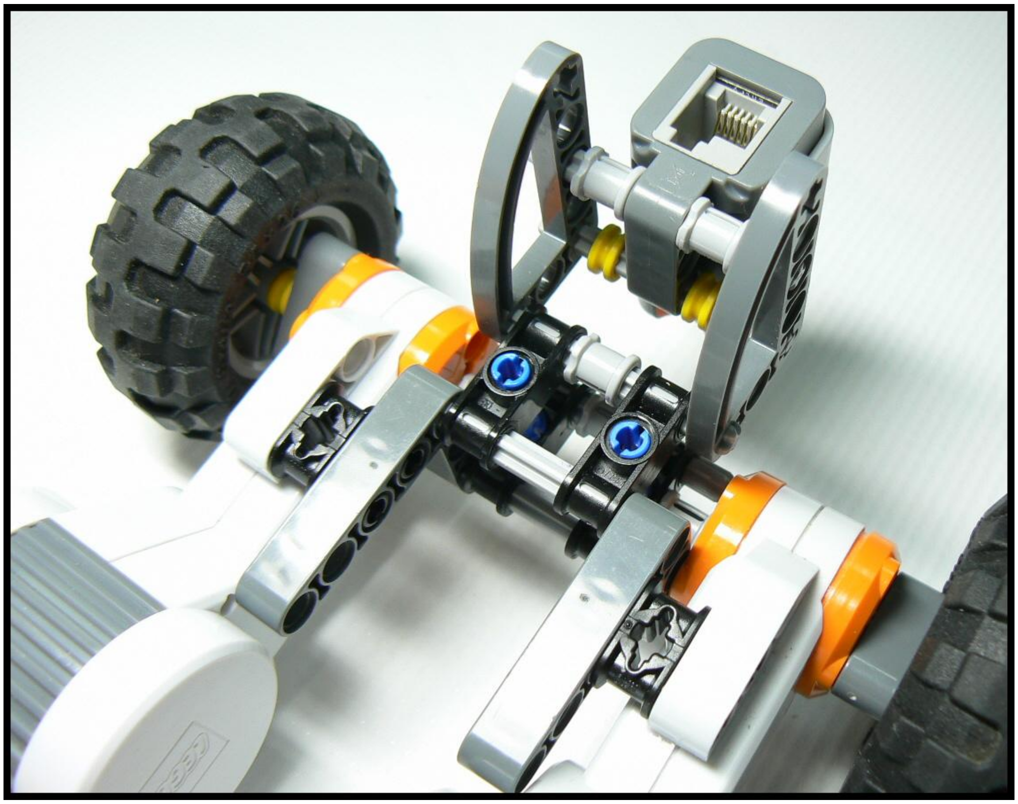


This assembly includes your Pen Slot, this is a good moment to check that your pen will fit. You may need to slide the black parts apart or together slightly – your goal is an easy slip fit without too much wobble.



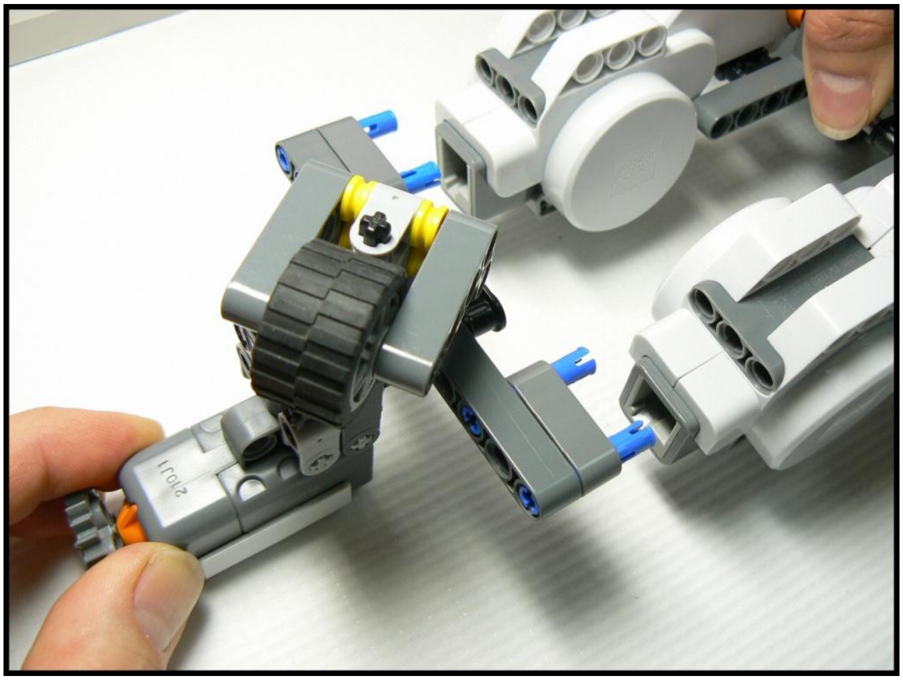
□ **6.** Assemble the lower chassis:



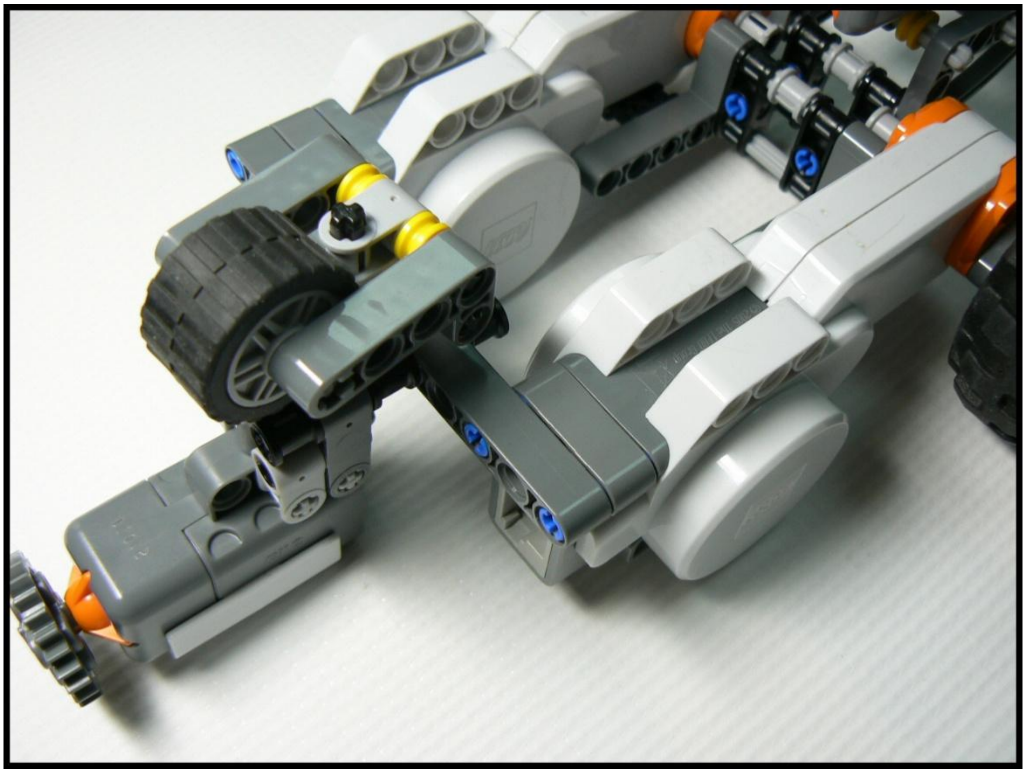


Note: this particular step is a loose fit, you'll need to hold things together during the next step.

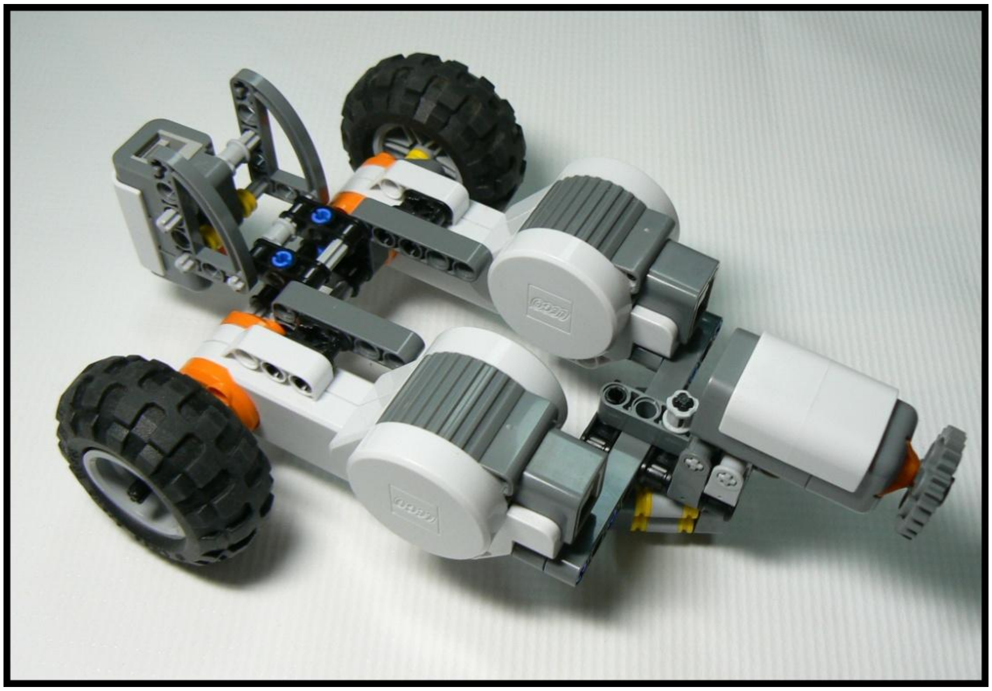
Flip over, this is the Bottom:



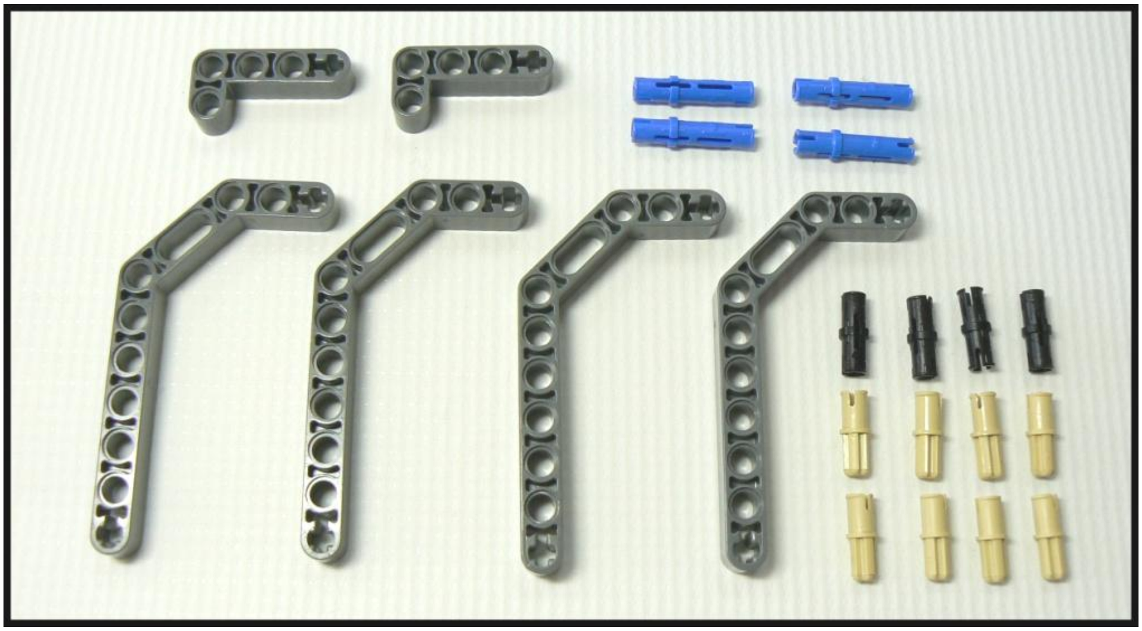
Still looking at the **bottom**:

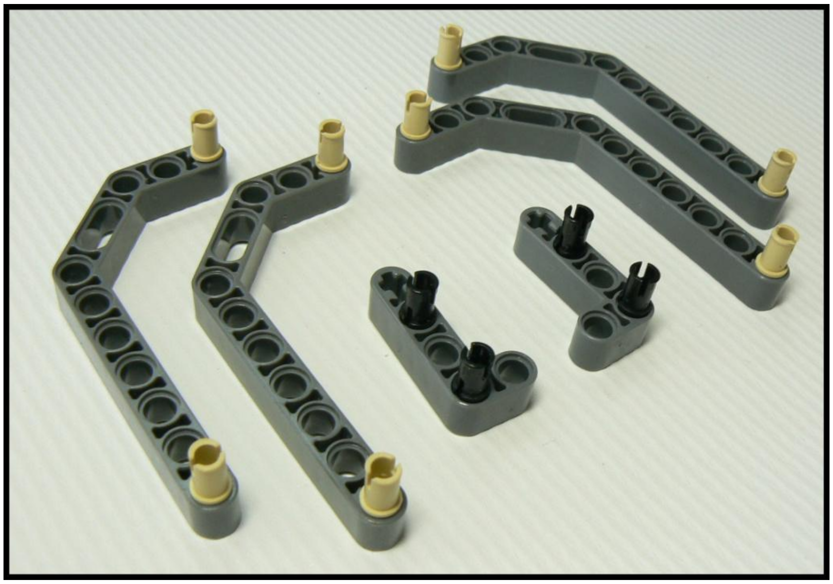


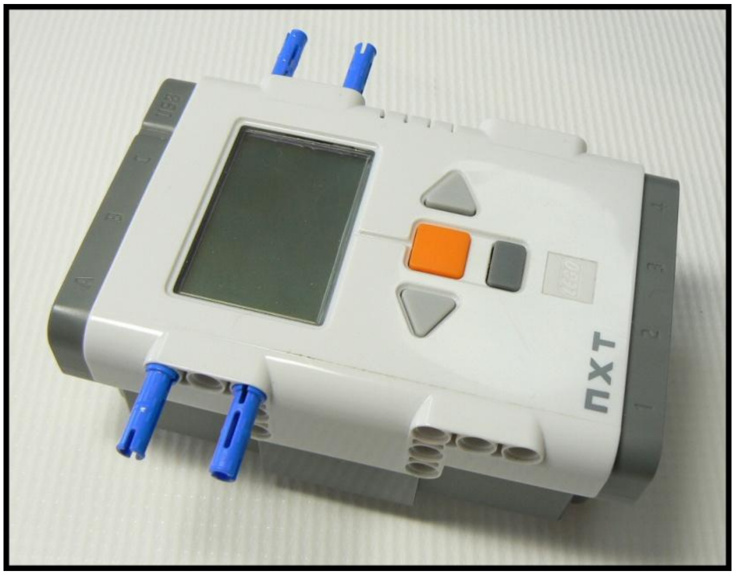
Inspect the Lower Chassis from the top, it should look like this:

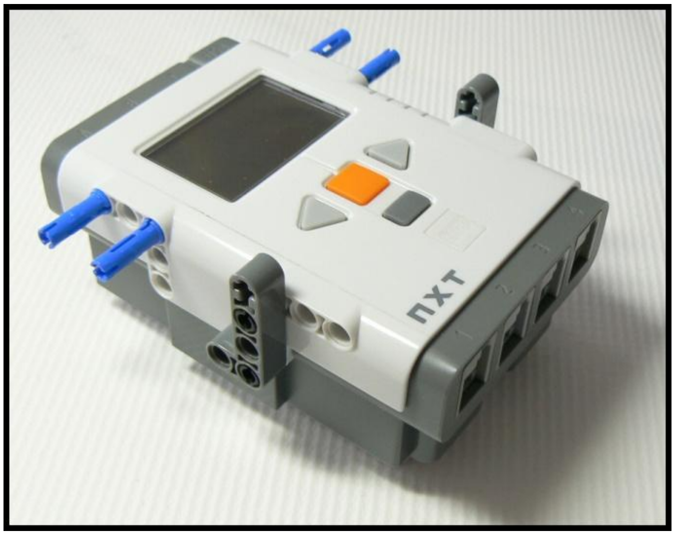


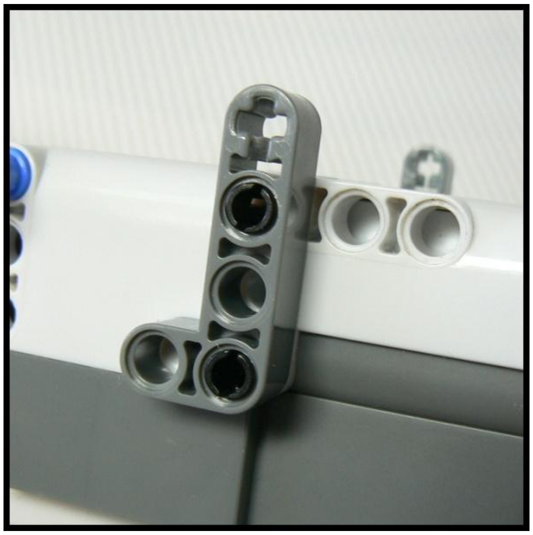
□ **7.** Mount the NXT Brick:

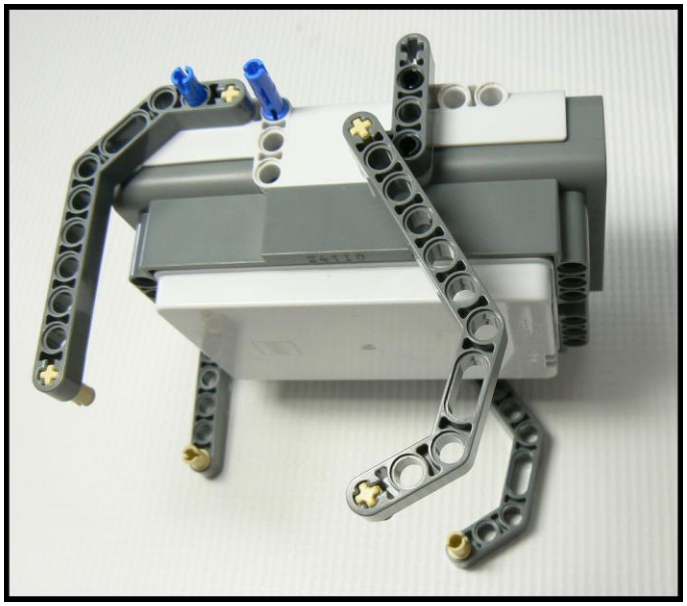


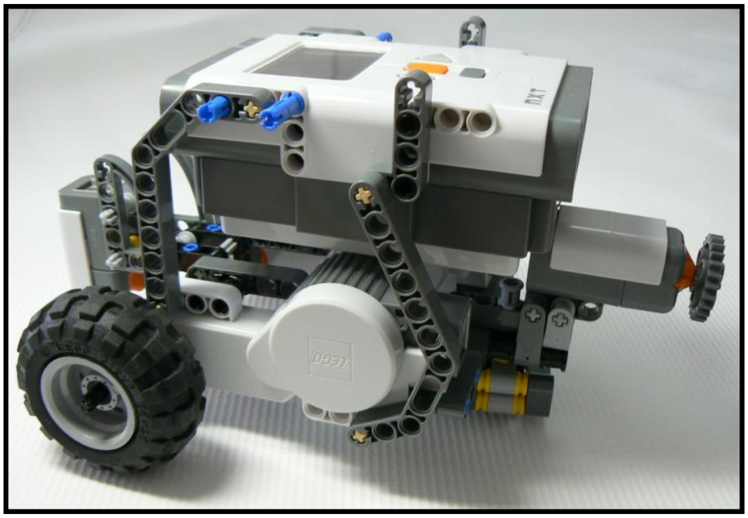



NXT

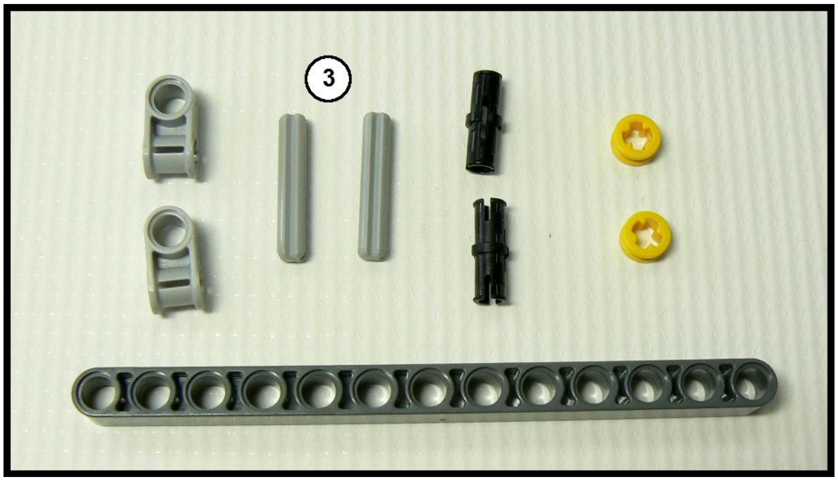

NXT

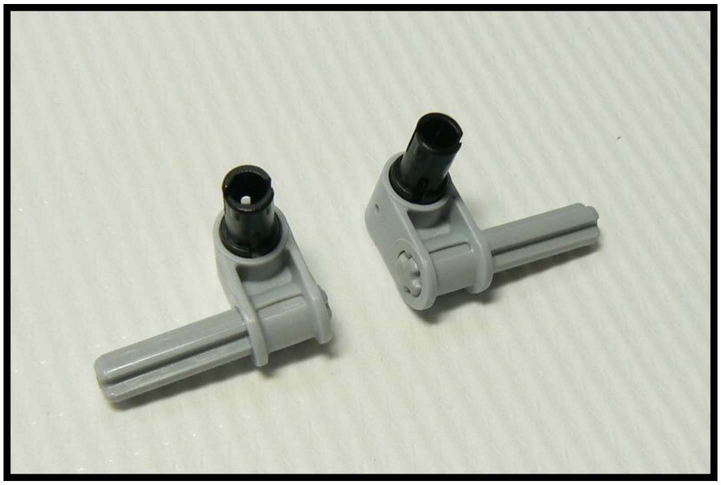


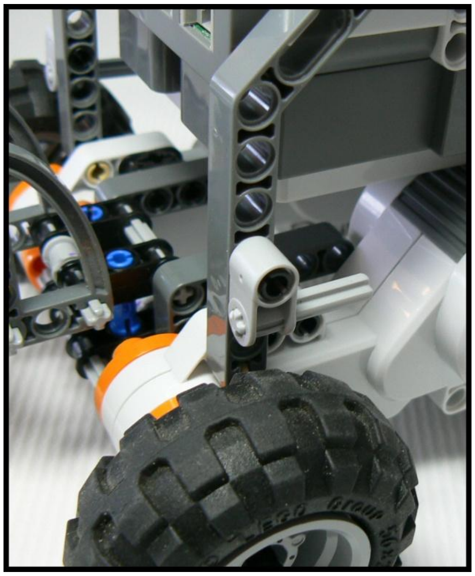


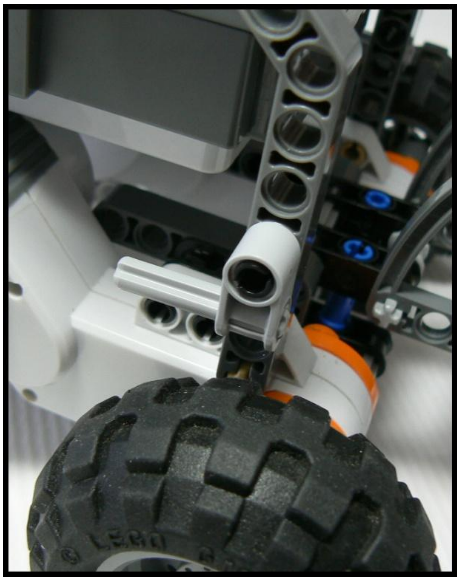


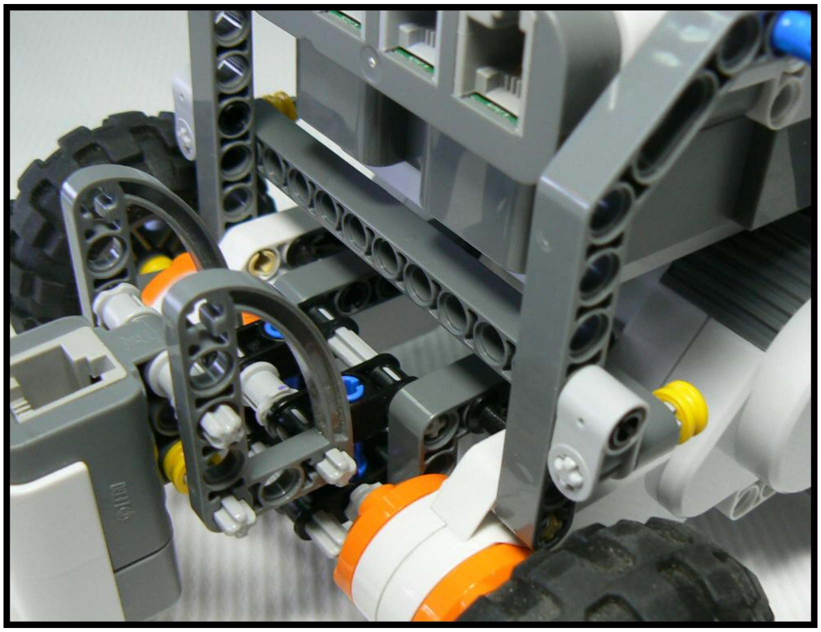
□ **8.** Build and attach the Stabilizer:



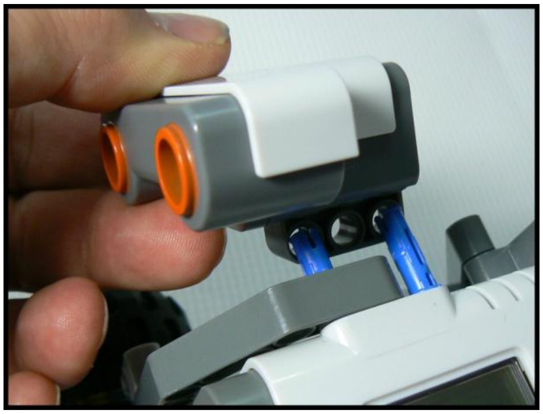


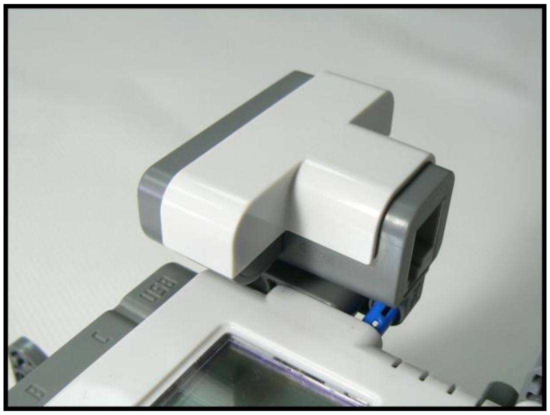




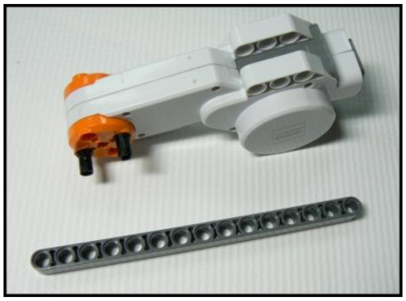


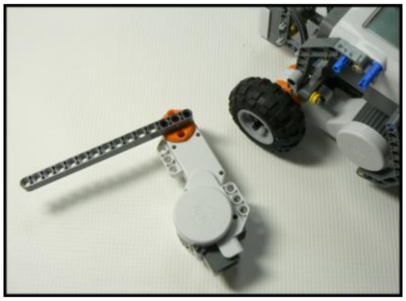
□ **9.** Attach the Ultrasonic Sensor:

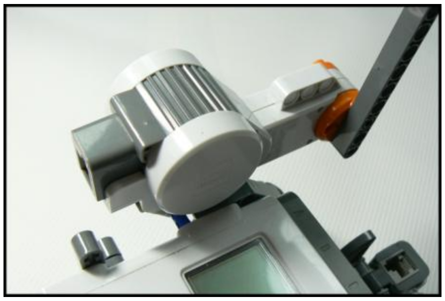




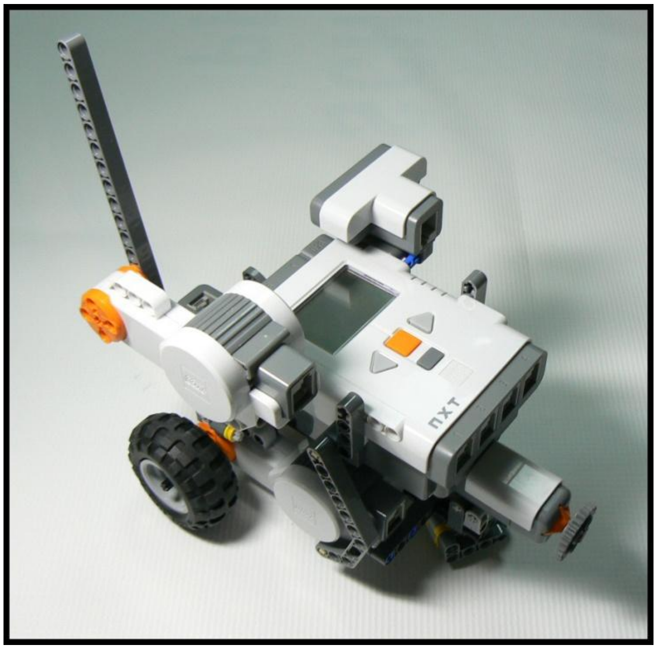
□ **10.** Attach the Arm motor:



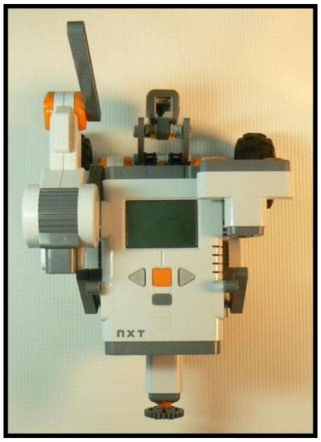




□ **11.** Your FlexBot should now look like this:

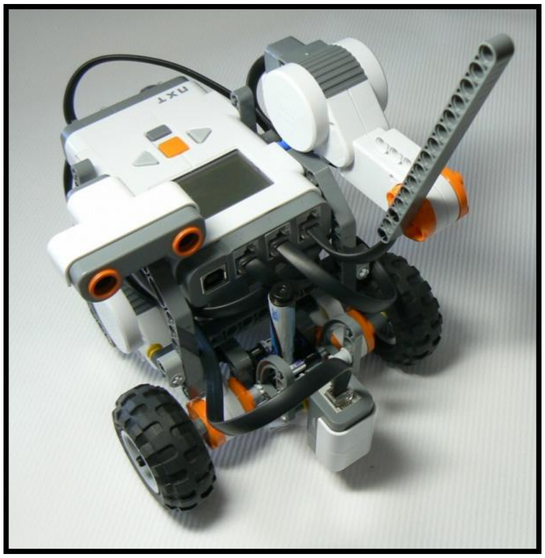


□ **12.** Add wires to each of the motors and sensors, according to this chart, with Left and Right as shown:

Left  Right

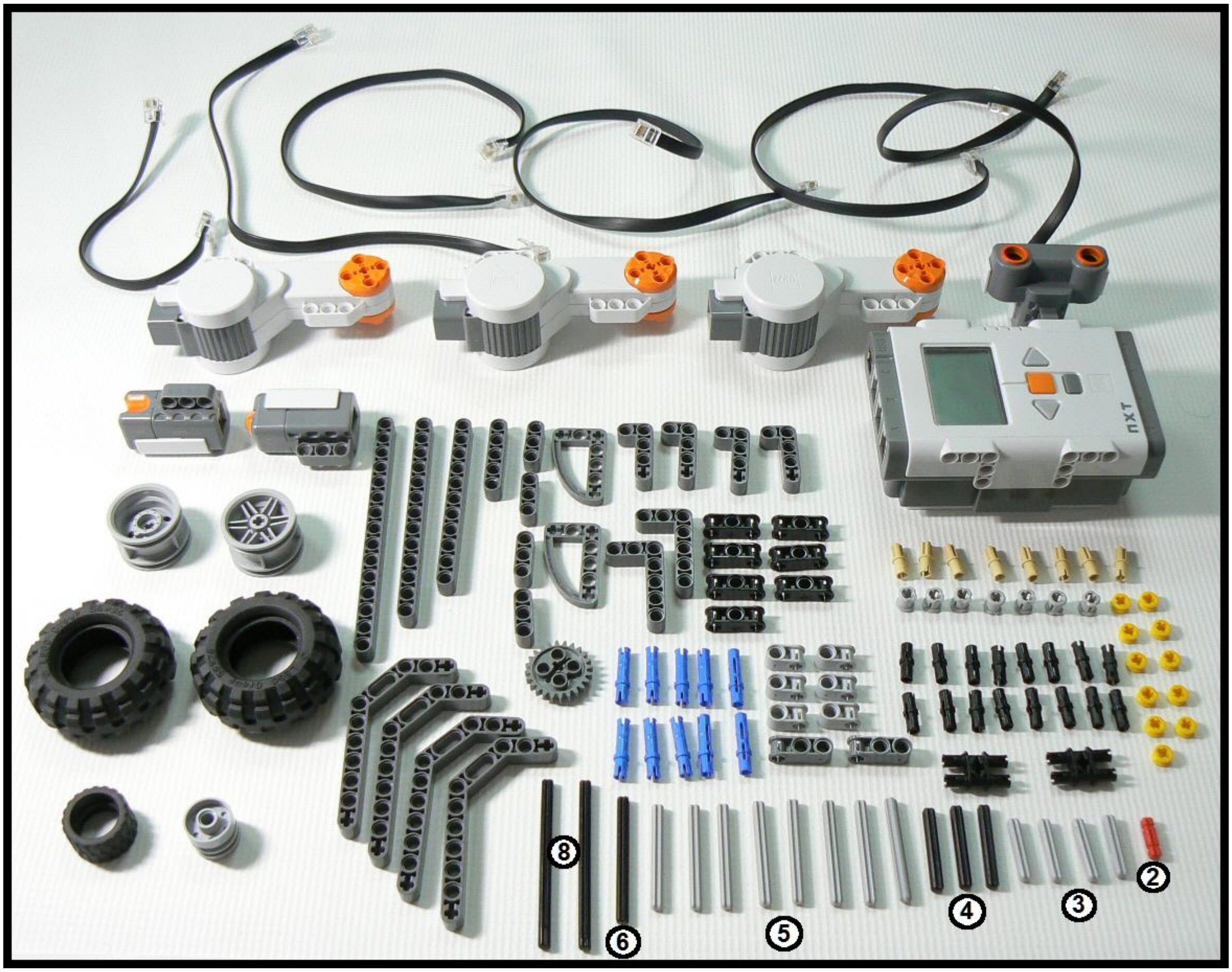
❑ Port A – Arm Motor
❑ Port B – Left Drive Motor
❑ Port C – Right Drive Motor
❑ Port 1 – Bump/Touch Sensor
❑ Port 2 – (not used)
❑ Port 3 – Light Sensor
❑ Port 4 - Ultrasonic Sensor

□ **13.** Tuck and coil the wires to keep them out of the way of all moving parts, buttons, the Pen Slot, display and sensors.

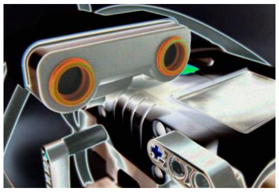


Congratulations you just built your FlexBot!

All the parts you'll need for one FlexBot:



Suggested Dry-Erase Markers (available from Amazon, Walmart and Others).
Search for "Dry-Erase Mini-Markers":



NXT Programming for Beginners

Project 2:Sound OFF (and ON)

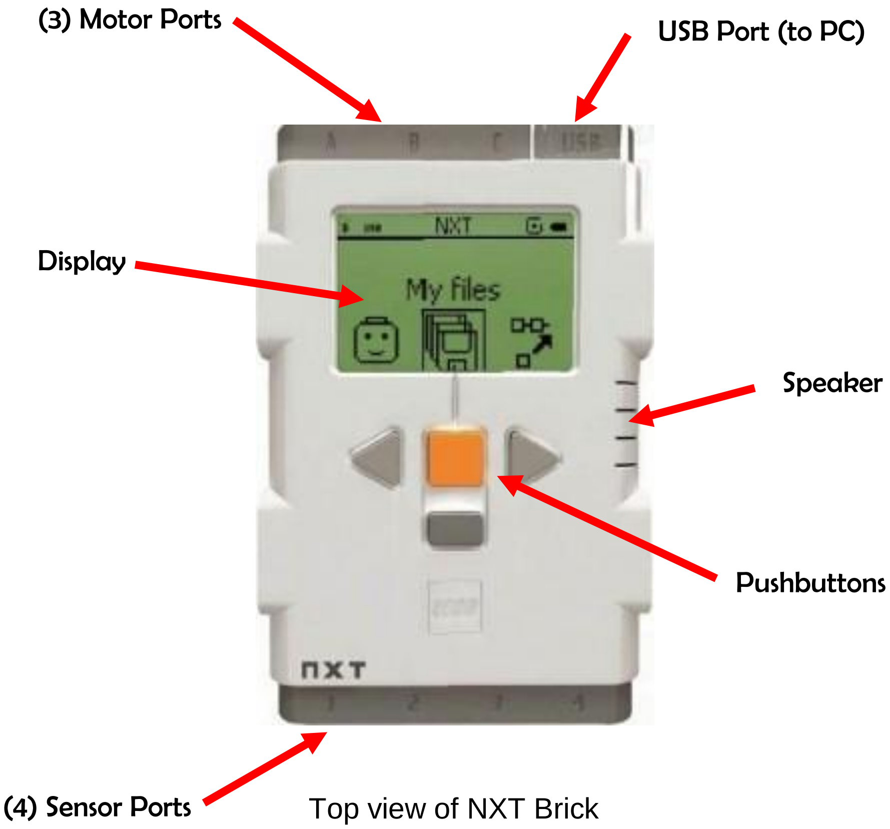
Learn about steps in a program, make some music.

Since this is probably your first time programming the NXT, we will take it fairly slow for now. Later projects will assume you know your way around the GUI and are stronger at troubleshooting programs.

Your NXT "Brick" (the brain) has the ability to store and run programs. On the outside it has a display panel, four pushbuttons, and a small speaker. There is a battery pack on the bottom -- either a rechargeable pack or a holder for AA batteries.

There are connectors (or "ports") at each end of the NXT. There are four sensor ports (1, 2, 3, and 4), three motor ports (A, B and C) and a USB port to link to your PC. You can also connect to your PC via Bluetooth, but for this workshop we assume a USB cable.

Here is a photo of the Brick showing these main components:



Top view of NXT Brick

As you'll find out, the display and pushbuttons on the NXT Brick can do quite a lot. For now we'll keep it simple – ***it's enough for you to know how to start the Brick, download your program and then run and stop your program.***

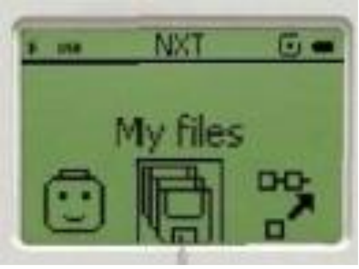
Note: Advanced or inquisitive users will find lots of great info about the NXT Brick in the "NXT User Guide", available online from many places.

Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

If you get stuck, try to work through it. If you get REALLY stuck ask for help.

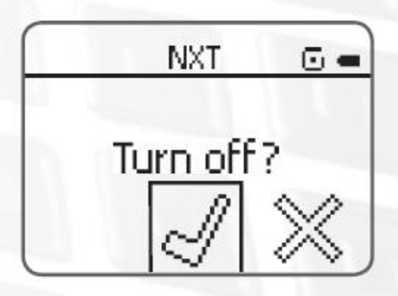
□ **1.** Your Brick needs fresh batteries. If your FlexBot has a rechargeable battery pack, make sure you fully charge it (overnight) before coming to the workshop. If you have the version with AA batteries, you may want to put fresh batteries in before starting the day.

Press the orange button on your Brick. After a moment you should see a startup screen and hear a little music. You will then see a screen that allows you to select a program to run. It will look like the following:



If this does <u>not</u> happen, your Brick probably needs new batteries, take care of this now.

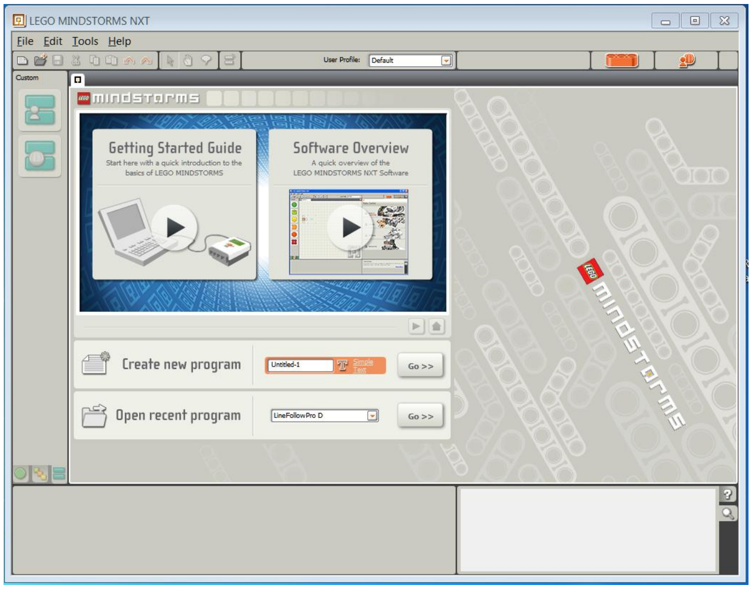
When you successfully see this screen, turn OFF your Brick. This can be done by pressing the dark gray button (repeatedly if necessary) until you see this screen:



When you see this screen, press the orange button one more time and your Brick will turn off.

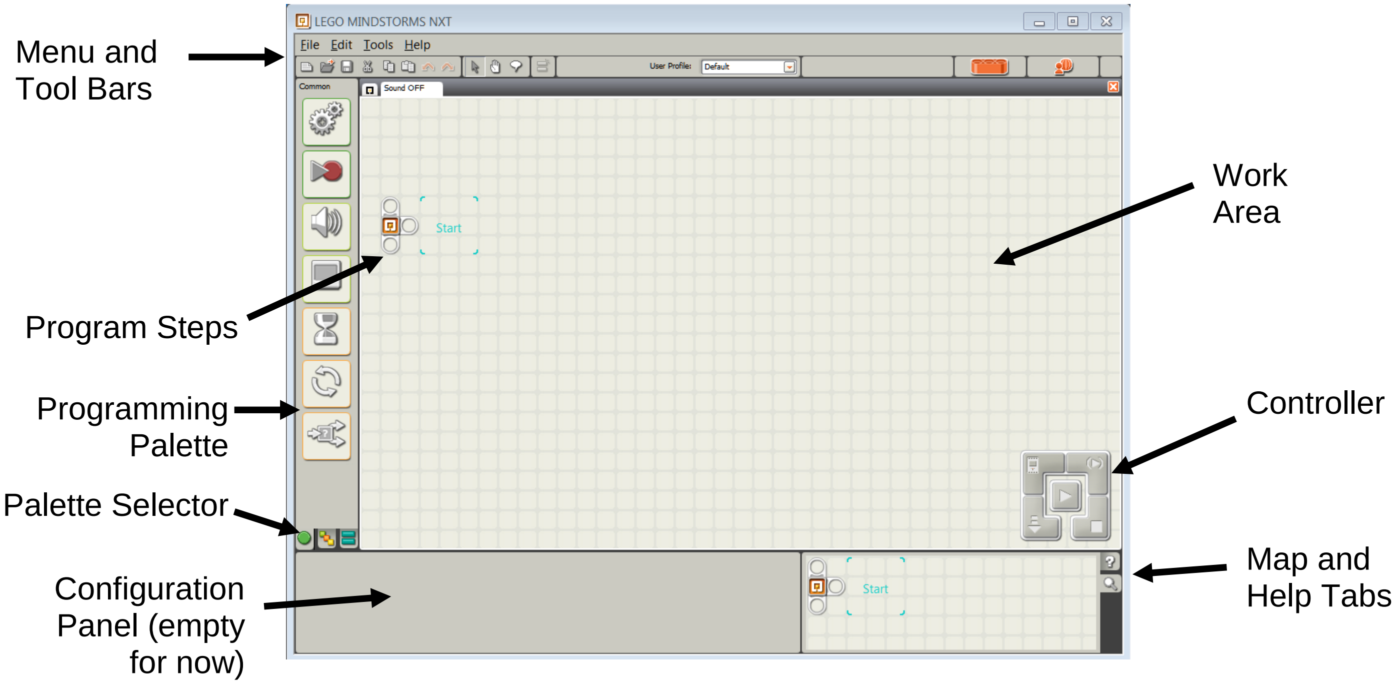
□ **2.** Connect your PC to the Brick with a USB cable. If you're not sure how to do this check with your neighbor or ask the instructor. Make sure all connections are fully seated but do not force anything.

□ **3.** Locate and run the LEGO Mindstorms NXT software on your PC. A window will appear that looks like the following:



You are encouraged (outside of workshop time) to view the "Getting Started Guide" and "Software Overview". Since we'll be covering much of this information, it's not strictly necessary.

□ **4.** Enter the name "Sound OFF" in the text field next to "Create New Program" and press the "Go>>" button. You will now see the following screen:



The big open area (the graph paper) is your **Work Area**. Notice that right now there are only two items in the program steps -- the left one signifies the Start of your program and the right one is a placeholder for the first block or statement in your program.

Since there's nothing after Start, this is an empty program; it will not do anything.

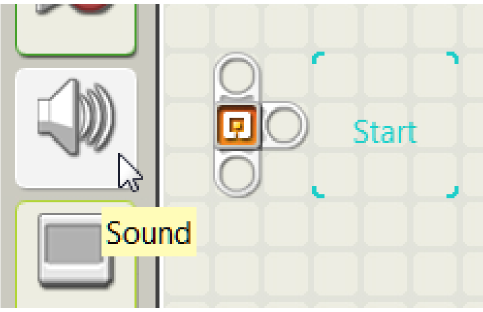
Along the left side of the window is the **Programming Palette** containing action blocks you select and place into your program. Each block does something different, you will learn about many of them in future projects. Think of them as "verbs", these are the commands that tell your NXT Brick what to do and when. At the bottom of the Programming Palette are Palette Selector tabs where you can select other palettes of blocks.

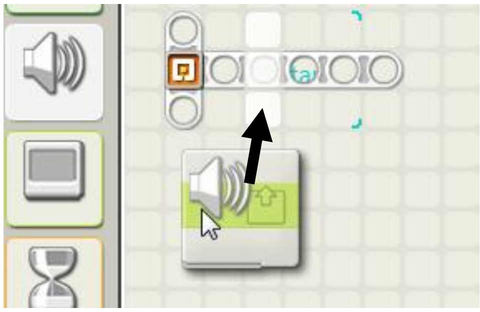
Along the top of the window are the **Menu and Tool Bars**. In the Menu bar you see the usual Windows menus, such as "File", "Edit" and so on. Below that is the Tool bar, with quick-access commands that can speed up many common operations. As you hover your mouse above them you see a brief description (a "tooltip") that tells you what that button does.

In the lower right corner are the **Map and Help tabs,** these let you scroll around in a big program and get help for each type of block. The **Configuration Panel** in the lower left allows you to make selections and adjustments for any action block.

Hint -- Try pressing the "?" tab now and then click on "More help >>". YOU'LL LIKE IT.

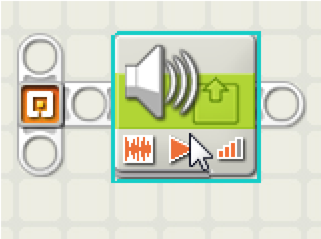
□ **5.** Move your mouse over the Programming Palette as shown; as you move around the each item's title is displayed, such as "Sound" in the left picture below. Click once (left button) and a block appears that you can drag with your mouse. You are now ready to insert this block into your program.

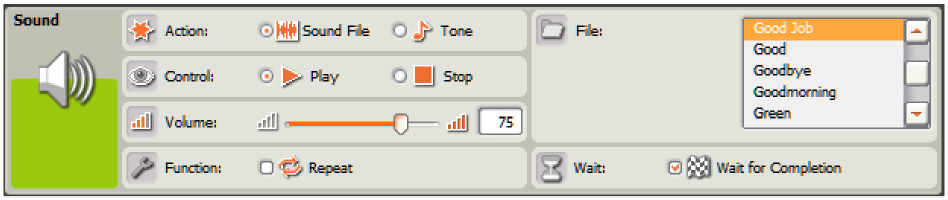




Notice there is a white vertical bar just to the right of the Program Start. Drag the block to this location. When you release the mouse button, it will snap to that spot.

□ **6.** You should now see your Sound block inserted after the Start Program Block as shown. Look at the bottom left of the window, the Configuration Panel should be displayed. This is where you make settings for that Sound block:





Sound

Action: ● Sound File ○ Tone

Control: ● Play ○ Stop

Volume: 75

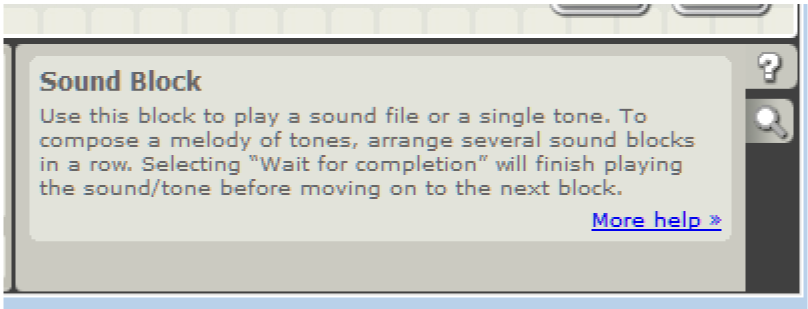
Function: □ Repeat

File: Good Job, Good, Goodbye, Goodmorning, Green

Wait: ☑ Wait for Completion

Hint – As you add Action Blocks, what you're doing is creating a step-by-step procedure for your NXT to perform. Later when this program is "downloaded" to your robot, it will do each of the steps from left to right.

□ **7.** If not already selected, click on the Help tab (**?**) as shown and hover your mouse over the Sound block. You should then see information about the Sound block. For more detailed help you can click on "More Help >>" and additional info, samples and other links will appear:



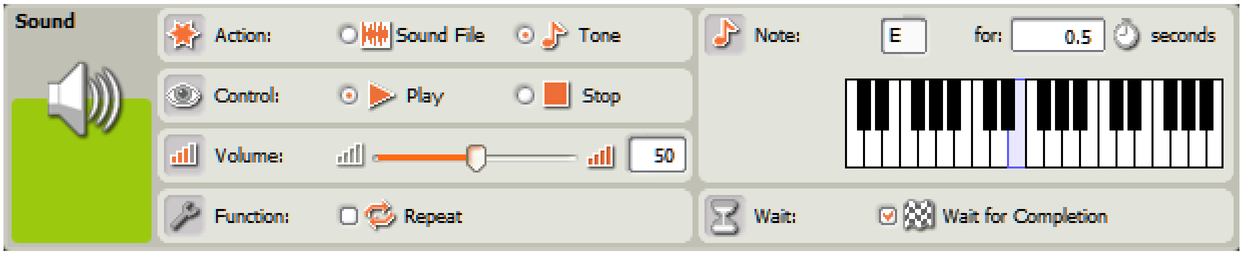
Sound Block

Use this block to play a sound file or a single tone. To compose a melody of tones, arrange several sound blocks in a row. Selecting "Wait for completion" will finish playing the sound/tone before moving on to the next block.

More help »

□ **8.** The Configuration panel for the Sound block allows you to select several kinds of sounds. Depending on the "Action" you select, your robot will play a sound from a Sound File (which you select) or a Tone, which you pick with a small piano-style keyboard. Your NXT Brick can play complex sounds (like voice and music) but it's a small speaker, so it's not very loud.

For this Sound block select "Tone" and the "E" key on the keyboard as shown. Set the time to be .5 seconds, volume at mid-range, and be sure to select "Wait for Completion":



Sound

Action: ○ Sound File ● Tone

Control: ● Play ○ Stop

Volume: 50

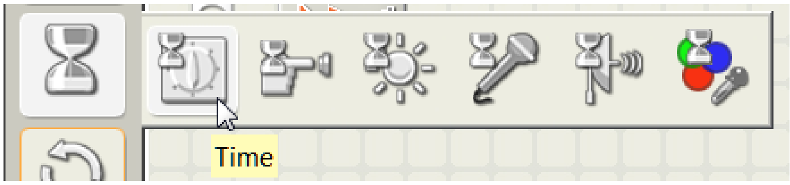
Function: □ Repeat

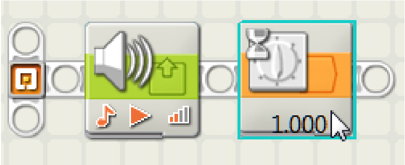
Note: E for: 0.5 seconds

Wait: ☑ Wait for Completion

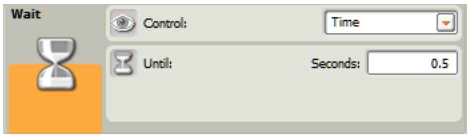
This tells the sound block what to do when executed. Since it's the first program statement, it will happen immediately.

□ ***9.*** Add a "Wait" block to your program, immediately to the right of the Sound block. When you move your mouse over the Wait icon (the hourglass) a menu will "fly-out" to the right. Select the left-most item, "Time", drag it to the right of the Sound block and drop.





□ **10.** In the Configuration panel you can now make selections for the Wait block. Edit the "Seconds" field to .5 seconds as shown:

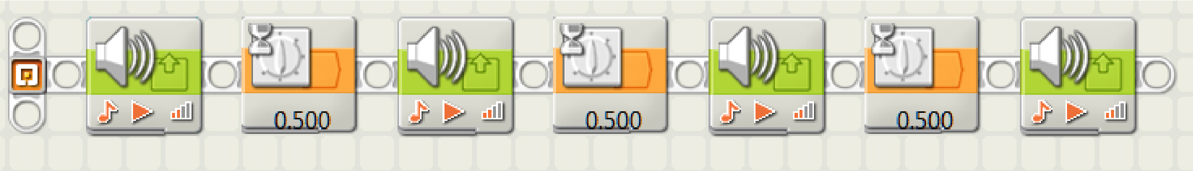


Wait

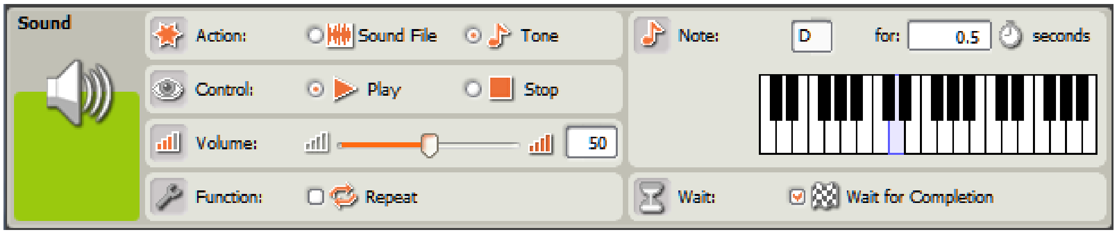
Control: Time

Until: Seconds: 0.5

□ **11.** Using the same technique above, add three more Sound blocks to your program, and two more Wait blocks. Using the Configuration panel, modify each of the Wait blocks to be .5 seconds. Your program looks like this:



□ **12.** Using the Configuration panel, change the second Sound block to be a "D" note as shown:



Sound

Action: ○ Sound File ● Tone

Control: ● Play ○ Stop

Volume: 50

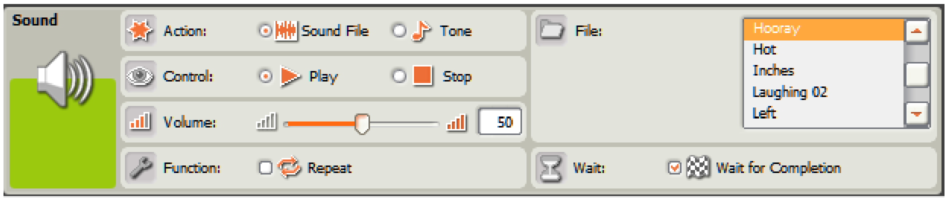
Function: □ Repeat

Note: D for: 0.5 seconds

Wait: ☑ Wait for Completion

□ **13.** Similarly change the third Sound block to be a "C" note.

□ **14.** Lastly, change the fourth Sound block to use a Sound File, such as "Hooray" (your option), and don't forget to select "Wait for completion". Otherwise you won't hear the last item!:



Sound

Action: ● Sound File ○ Tone

Control: ● Play ○ Stop

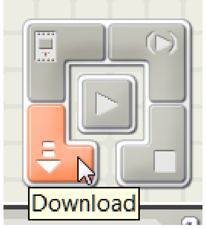
Volume: 50

Function: □ Repeat

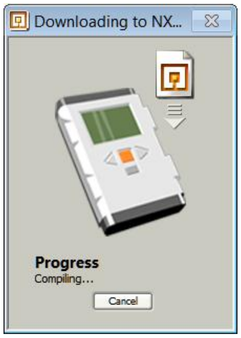
File: Hooray
Hot
Inches
Laughing 02
Left

Wait: ☑ Wait for Completion

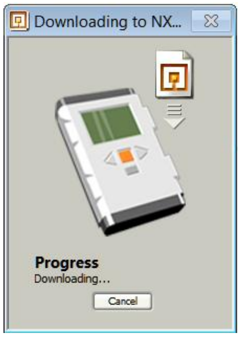
□ **15.** Ensure that your NXT Brick is hooked up to the PC and powered on. On the Controller select the Download button as shown:



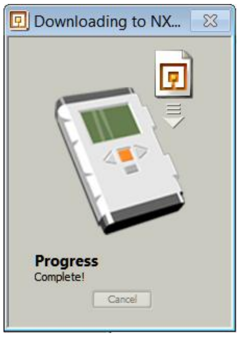
After pressing this button, if all is well you will see a sequence of three windows:



Progress
Compiling...



Progress
Downloading...



Progress
Complete!

Followed by a beep from your NXT Brick. This signifies that your program has been downloaded to your Brick. If this does not happen, check your cable. If necessary, seek help.

□ **16.** Press the orange button on your Brick until the screen displays "Sound OFF" with "Run" under it. Once you see this, press the orange button one more time. What do you hear? Hopefully it's what you expected!

□ **17.** For safe keeping (and later use) save your admittedly not-very-complex program so you can see how it's done. Select File then Save (or Save As) to save your program.

You will now see a "Save" dialog where you give your program a name and save it to a folder that your instructor suggests.

Hint: It's better to give names to your files that make it easy for you to find later. For example the file for this project would be easier to identify if named "Bill's Project2.rbt" rather than "This is really neat.rbt". Also FYI, all worksheet files need to have the ".rbt" extension.

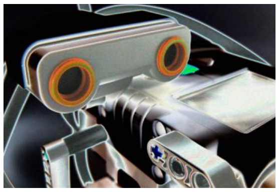
Questions for thought:

1. *Why did we add the Wait blocks between the sounds? How would it sound different if they were not there? Try it if you like.*
2. *Why did we select "Wait for Completion"? If you're not sure, uncheck it and try again. What you'll find is that if you don't wait for completion, the program will start playing a tone and then go on to the next block before it's even done playing. <u>This can be lots of fun if you want your robot to "dance" while it plays a tune.</u>*

Congratulations you just created, downloaded, executed and saved your first NXT program!

Take a deep breath, pat yourself on the back and get ready for…

Project 3, Straight on till Dawn



NXT Programming for Beginners

Project 3:Straight on till Dawn

At this point we will start to move more quickly. As always, if you get stuck refer back to earlier projects, try something, read the help system…eventually you'll figure it out. If you REALLY get stuck there's always the instructor or a lab neighbor.

Baby's first steps

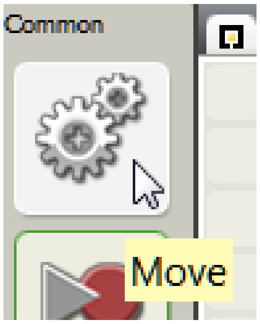
You've waited patiently; you've built your FlexBot, created a program and saved it. Now it's time to make this little bugger move. OK, it's only in a straight line for now, but that's a lot better than just sitting there.

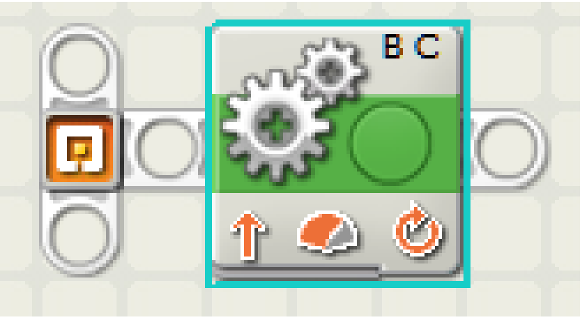
Before we jump right in, let's take a moment to look at the FlexBot to see what makes it able to move. A quick inspection shows there are two large wheels, one on each side. Turn it over and you see there's a third wheel or "caster" near the bump sensor. This allows the robot to sit like a tripod (three legged device), able to handle bumps and uneven surfaces pretty well. Take a closer look at the two bigger wheels; they have lots of rubbery tread, providing plenty of traction.

Driving each of these big wheels is a motor (there's other cool stuff inside the motor that we'll discuss later). Under program control you can tell each of these motors how fast to turn, how far (or for how long) and in what direction. This means you can make your FlexBot go straight, turn, spin, stop and anything else you can dream up. And just like when you programmed your robot to play music, you can tell your robot to move one way, then another and then another by just inserting the appropriate Action Blocks.

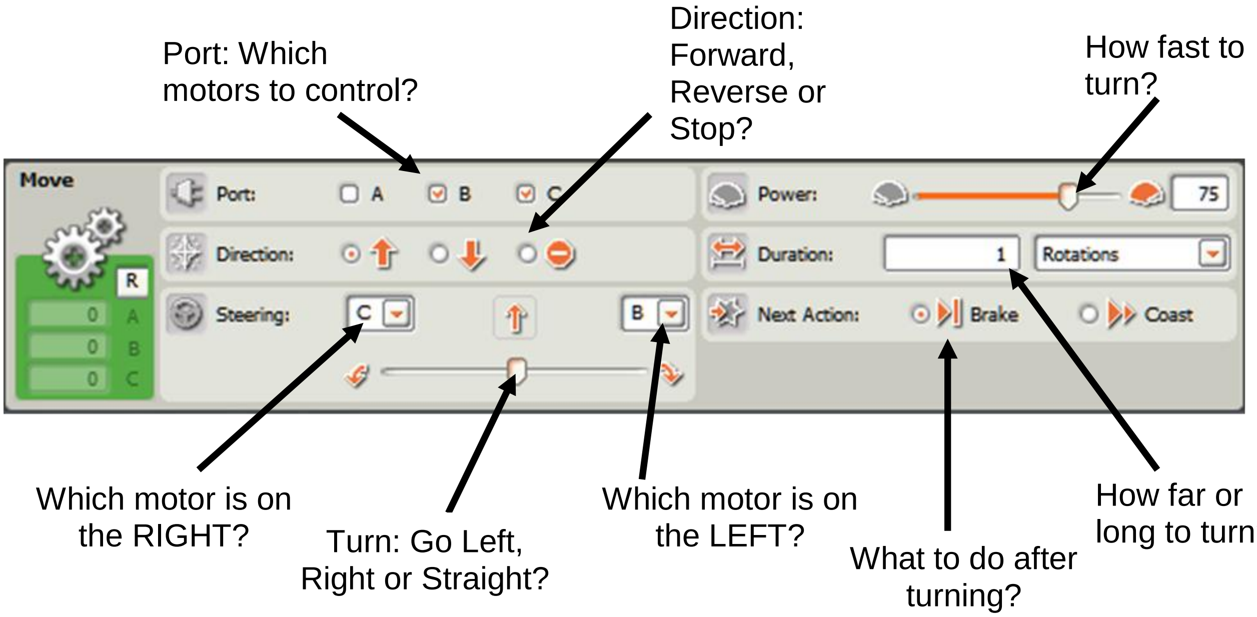
Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

☐ **1.** Starting with a clear worksheet, find and add a Move block. In the Programming Palette it looks like two meshing gears. Here are two screen shots – selecting a Move block, and the Move block in place in your program:





Immediately after inserting the Move block the configuration panel allows you to make quite a few settings. It seems to be pretty dense with information, let's review the parts that we'll use in this activity:



Move

Port: □ A ☑ B ☑ C

Direction: ● ○ ○

Steering: C B

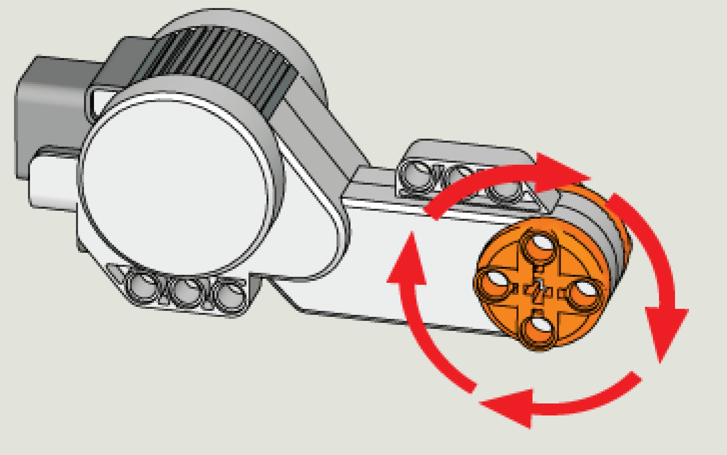
Power: 75

Duration: 1 Rotations

Next Action: ● Brake ○ Coast

R
0 A
0 B
0 C
Port: Which motors to control?
Direction: Forward, Reverse or Stop?
How fast to turn?
Which motor is on the RIGHT?
Turn: Go Left, Right or Straight?
Which motor is on the LEFT?
What to do after turning?
How far or long to turn?

There are three main functions here: how fast is each motor turning for how long (or far) and how should the motor stop when it's done. This picture (taken from the Lego help system) shows a motor turning in the Forward direction:

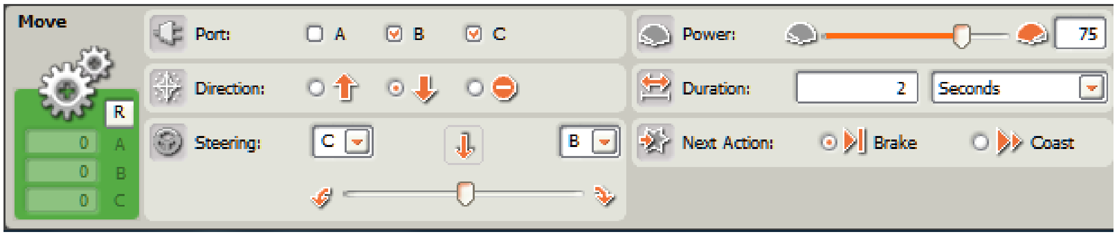


The best way to learn is to try something. Let's make this robot MOVE!

□ **2.** In general the end of your FlexBot with the light sensor is the FRONT. That means that the Motor on Port B is on the LEFT and the Motor on Port C is on the RIGHT. Follow the wires from the Brick and make sure.

Note: In the FlexBot moving **Forward** moves towards the Light sensor and **Reverse** moves towards the Bump Sensor!

□ **3.** In the Configuration panel for the Move block, set Ports B and C active (they may be already!), Direction: Reverse, Motor B on LEFT (C will automatically be on RIGHT), Steering: go Straight, Power: 75, Duration: 2 Seconds, Next Action: Brake. It should look like this:



Move

Port: □ A ☑ B ☑ C

Direction: ○ ● ○

Steering: C B

Power: 75

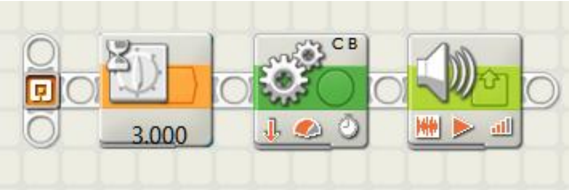
Duration: 2 Seconds

Next Action: ● Brake ○ Coast

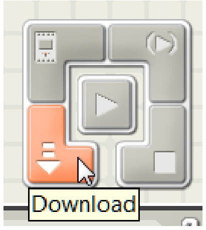
Note: The Steering controls can be confusing and are <u>*not*</u> *well documented. They are *fairly* intuitive but don't expect good accuracy or repeatability when turning.*

□ **4.** Insert a 3 second Wait block before the Move block.

□ **5.** Add a Sound block after the Move block, select the "Good Job" File. Your program should now look like this:



□ **6.** Download the program to your NXT by clicking the Download button on the Controller:



□ **7.** Disconnect the USB cable from your NXT, place the robot on a table where there is <u>plenty</u> of room to move. Using the Orange button, Run the program on your NXT.

Be ready to catch your robot if it heads for the edge of the table!

Questions:

1. Did it move as you expected? Run the program a few times.

2. How long did it wait (before moving) after you pressed Run?

3. When the robot stopped, did it stop smoothly or abruptly?

4. What might you do to make it stop more smoothly (Hint, look at the Next Action options)?

□ **8.** **Use a ruler or tape measure and check how far it travels.** Do this a few times, does it seem to be repeatable? *What you'll probably find is that controlling a Move block with time isn't very accurate!*

□ **9.** Modify the program to go for 2 **Rotations** (not seconds), set the Power to 50 (half speed). Download and run, measure as accurately as you can how far it travels.

□ **10.** Repeat the above, but move for 4 Rotations instead. Measure again. How does the distance traveled compare to distance in step 9 above? Is this more repeatable than using time to control?

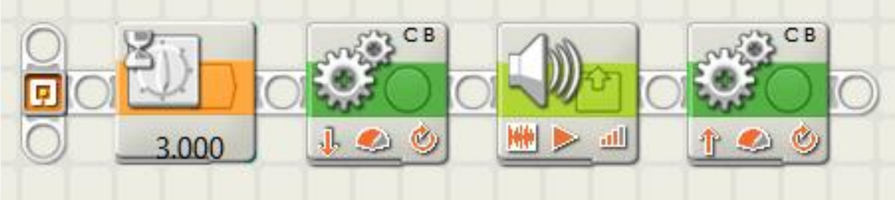
□ **11.** Repeat step 10 above but with a Power of 25 for each wheel. Measure the distance. What conclusions can you draw about expected behavior? Do you think you can predict how far it will move for other Powers and Rotations without having to run the program?

□ **12.** Based on what you saw above, how far (in inches) do you think the FlexBot will travel with a Power of 75 on each motor running for 3 Rotations? Write your answer here: _________

□ **13.** Perform the test described in step 12, was your prediction right? If not, how do you account for the difference?

> ***Note:*** *In the next step you will copy a block, and place the copy in a new location. An easy way to do this is to select the block (click on it with your left mouse button and release), press and hold the Ctrl key on your keyboard, and click and drag the block to the new location. Try this a few times, see how it works. You can delete an unwanted block by selecting it and pressing the Del key on your keyboard.*

□ **14.** Using the program from step 13, copy the Move block and drop the copy at the end of your program. Change the direction in the new block from Reverse to Forward. Your program should look like this:



Notice that the arrow in the second Move block points UP (Forward).

□ **15.** Download and Run this program, does it return accurately to the starting point?

Hint -- The NXT motors have an "encoder" (an optical sensor) that measures the number of wheel rotations; it's quite accurate as you're hopefully discovering. This allows you to create programs that move in very controlled ways, without having to worry too much about differences in motors, friction and battery condition.

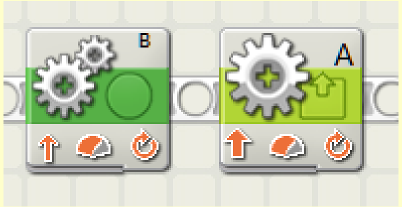
Question for thought -- Given the above, how accurate do you think the motion will be if one or both wheels slips or skids?

□ **16.** Save your program with a name that will make it easy for you to find later.

Important info about NXT Motors – Move blocks versus Motor blocks.

Experienced NXT programmers have learned (often the hard way) that NXT motors can be quite challenging to control, particularly if you care about the robot doing the same thing each time you run it.

You may also have noticed in the Complete programming palette, in the Action flyout there is a block called **Motor**. In this screen shot, you see a Move block on the left and Motor block on the right.



They both control motors, but there are important differences:

About Move blocks:

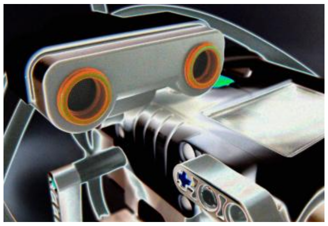
1. If you want to drive in a straight line, Move blocks (with two motors controlled) are the best – they keep track of the behavior of the two motors together to produce a straight run.
2. Using Move blocks to control multiple motors makes your program more compact – taking up less screen space and memory.

About Motor blocks:

1. If you want controlled, accurate arcs, Motor blocks are the best. The Motor block has the capacity for very precise control over the rotational speed of the motor, this makes your turns very pure (*as long as you enable **Control: Motor Power.** More about this in project 4).*
2. The Motor block controls only one motor at a time; you need one block per motor.
3. The Motor block lets you "ramp" the motor speed up or down in a manner you control, providing smooth starting and stopping.

In many of the projects to come, you may choose which one to use.

It's worth experimenting, you'll quickly find out which is better in your application.



NXT Programming for Beginners

Project 4:A”maze”ing Turns

Sometimes the only way to get ahead is by going sideways

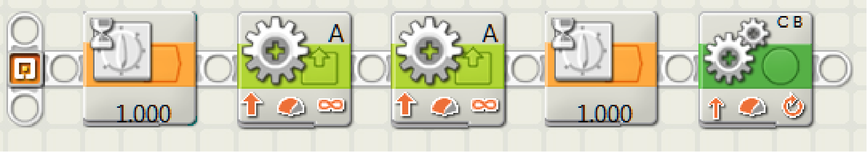
Now that you’re a pro at going straight (and predicting how far you’ll go) it’s time to do some turning. Thankfully your FlexBot has a couple of cool design features. The first is that the two driving wheels are arranged so that if they rotate in opposite directions at an equal speed, the robot will simply spin in place. The second feature is that there is a hole THROUGH the FlexBot at the center of that axis.

The combination of these features allows us to use the FlexBot as a sort of X-Y plotter. Just download your program, drop a pen through the hole, and start writing. Of course it’s not *quite* that simple, but almost.

> In this activity you make your FlexBot turn, go straight, turn again and so on. Hopefully by now you’re comfortable with the fact that your program is a sequence of steps which are performed one-by-one in the order in which they are placed from left to right.
>
> As you’ll see in this and future activities your NXT Brick is capable of making decisions, repeating (or ignoring) groups of statements and more. If your program is NOT behaving “right”, take the time to carefully review the action blocks and you’ll probably find that the robot is doing exactly what you told it to do.

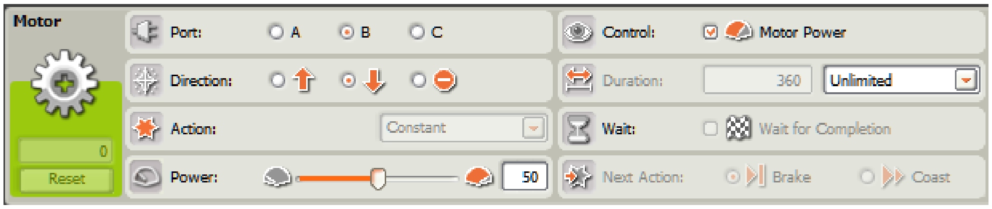
***Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil* ☑.**

□ **1.** Starting with a new program find and add five blocks: a Wait block, two Motor blocks (from the Action fly-out menu), another Wait block and Move block. Your program will look about like this:



□ **2.** The first Wait block is simply to give you time to get ready before the robot starts moving. Set the wait to 1 second (it may already be that value).

☐ **3.** Set the first Motor block (with the Configuration panel) to: Port B, Direction Reverse (down), Power 50, Duration Unlimited. **Under Control, check the Motor Power box (very important!).** The Configuration panel should look as shown below:



Motor

Port: ○ A ● B ○ C

Direction: ○ ↑ ● ↓ ○ ⊖

Action: Constant

Power: 50

Control: ☑ Motor Power

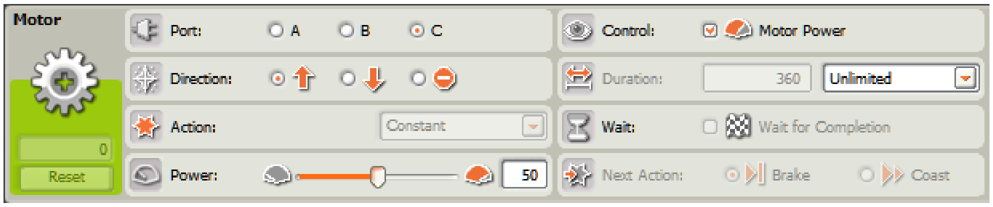
Duration: 360 Unlimited

Wait: ☐ Wait for Completion

Next Action: ● Brake ○ Coast

These setting will cause the motor on Port B to turn in Reverse indefinitely (or until we stop it!) at half Power (50 out of a possible 100). ***It also tells the NXT to monitor and control the Motor Power!***

☐ **4.** Set the second Motor block to: Port C, Direction Forward (up), Power 50, Duration Unlimited, Control Motor Power checked. The Configuration panel should look as shown below:



Motor

Port: ○ A ○ B ● C

Direction: ● ↑ ○ ↓ ○ ⊖

Action: Constant

Power: 50

Control: ☑ Motor Power

Duration: 360 Unlimited

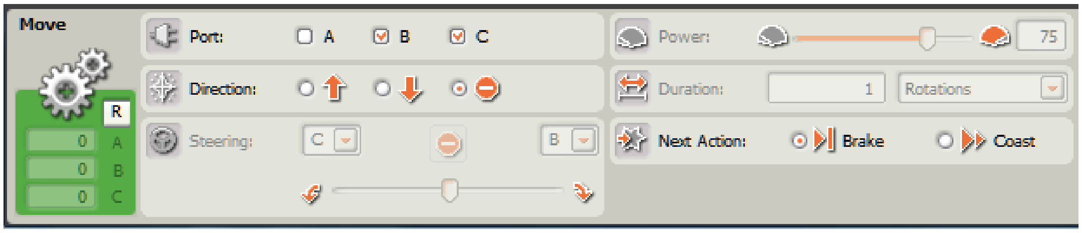
Wait: ☐ Wait for Completion

Next Action: ● Brake ○ Coast

These setting will cause the motor on Port C to turn Forward indefinitely at half Power.

☐ **5.** Set the Wait block to 2 seconds. This is the “Drawing Time” during which your motors turn and your FlexBot makes a mark on your paper/board.

☐ **6.** Set the final Move block to Port B and C, Direction “-“. This means “Stop”. Also be sure that Next Action is set to Brake. This will help the motors stop more quickly. The configuration panel for this Move block should look like:



Move

Port: ☐ A ☑ B ☑ C

Direction: ○ ↑ ○ ↓ ● ⊖

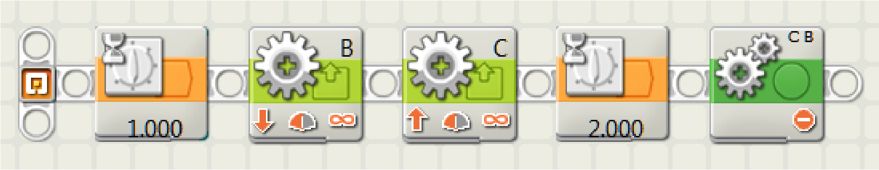
Steering: C B

Power: 75

Duration: 1 Rotations

Next Action: ● Brake ○ Coast

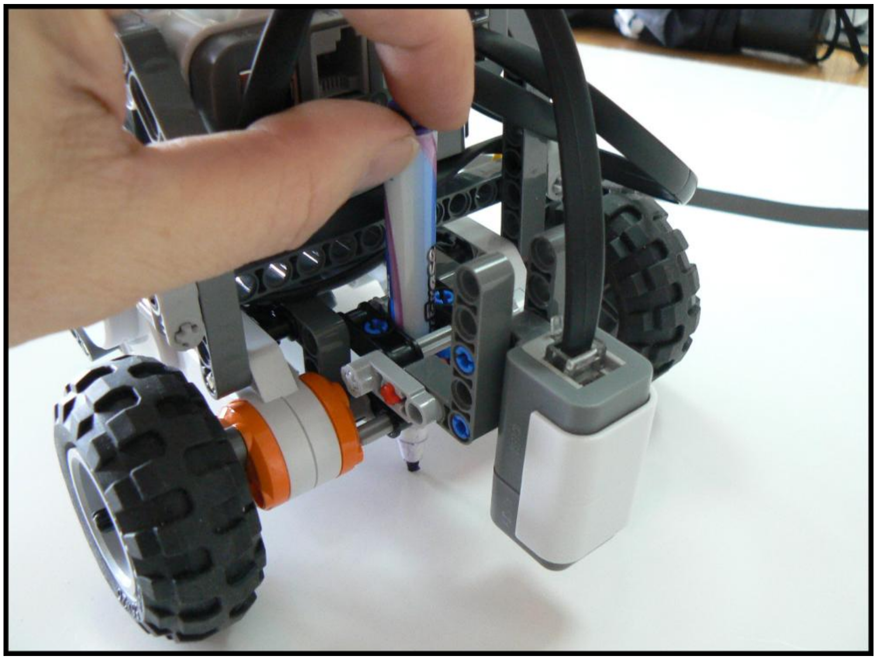
□ **7.** Your final program should look like this:



□ **8.** Download the program to your NXT, don't run it quite yet.

Caution – We're about to do some writing with your FlexBot. If you are using an erasable white board, BE SURE TO ONLY USE DRY ERASE MARKERS!!!! Otherwise you can permanently damage the board. Also please cap the pen after each run, that saves it from drying out.

□ **9.** Get a pen and a sheet of paper (or dry marker and white board). Place your FlexBot on the paper or white board, remove the cap from your pen and insert the pen through the hole tip-down. The pen should fit easily into the Pen Slot (remember, you tested this when you built the FlexBot?). This picture shows a pen being dropped into place, be sure the "nib" rests on the writing surface:



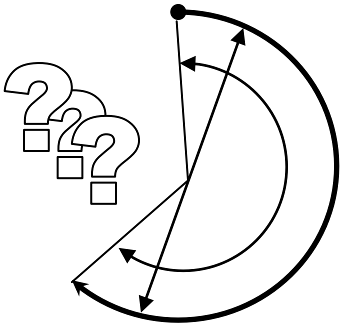
***You may find it easier to insert the pen from underneath before setting the FlexBot down*.**

□ **10.** Without moving the FlexBot run the program (using the Orange button). Once the program has finished, remove the pen first, **CAP IT!!!** then lift your FlexBot. Look at what was drawn.

If you entered the program correctly all you will see is a single dot, perhaps a very small circle or squiggle. OK, it's pretty dull but THIS IS GOOD NEWS, **it means that your FlexBot is able to spin (more-or-less) on its own center!**

□ **11.** Save this program, give it a useful name.

□ **12.** Now let's learn how to draw a circle. Change your program to have a power of 75 on Port B and 10 Port C (leave the Directions as they are). Set the Wait block to 1.5 (1 ½) seconds. Upload to your FlexBot, set up the pen and go again. You should now have an arc. What is the diameter of the arc in inches? ______



Approximately what portion of a circle was drawn (there are 360 degrees in a full circle)? _____ degrees

□ **13.** Now it's your turn. If your FlexBot is typical, you drew about ¾ of a circle. Hmmm… but we want a FULL circle, how do we get there? (Think about this BEFORE reading the next sentence!)
.
.
Time to think…
.
.
Did you realize that you can change the Drawing Time? Try it, download and test.

What did it do?

Did it come back (more or less) to the starting point? If not, change the time and try again.

Did you use math to calculate how much more (or less) time to give?

□ **14.** Adjust the Power values and times to produce as nearly a 7" diameter circle as possible, starting and stopping on the same spot. (Hint: you may want to change the direction of one motor)

Write down the final values here:

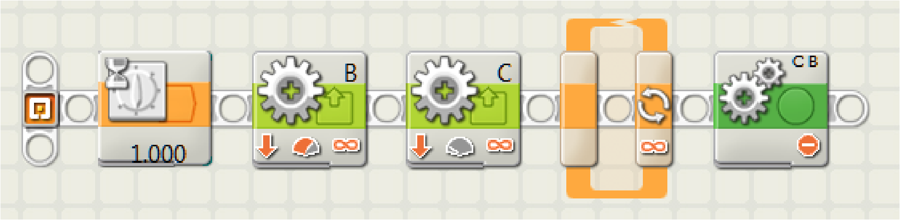
Port B Power _____ Port C Power _____ Drawing Time _______

Port B Direction (Forward/Reverse) ____ Port C Direction (F/R) ____

□ **15.** Save this program under a useful name.

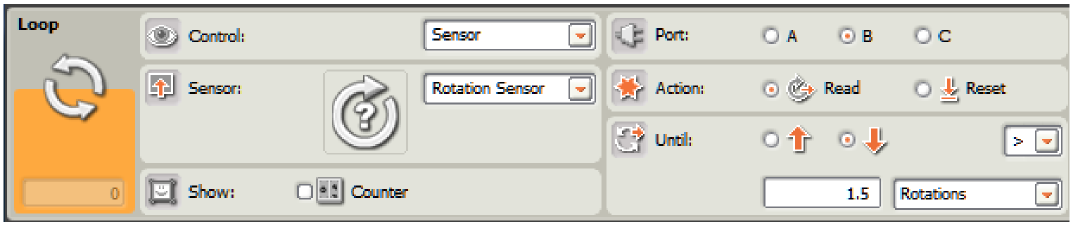
Now we will learn about a powerful action block, the "Loop" block. We will use this extensively in future projects, but let's see how it can be used in our simple program.

□ **16.** Starting with the program from Step 14, delete the second Wait block. Click on the Loop block in the Programming Palette. Place this block after the second Move block as shown:



> ***Loop blocks*** *are a special "structure" in a program that repeat whatever is inside the loop, usually until some condition is met. In this simple program you don't put anything inside the loop, but instead you use the loop to wait for one of your motors to turn a certain number of rotations.*
> ***Using rotation counts is <u>much</u> more accurate than measuring time for controlling your robot motion!***

□ **17.** In the Configuration panel for the Loop, set Control to "Sensor", Sensor to "Rotation Sensor" , Port to B, Action to "Read", and Until to Reverse (down), Greater than (>) and 1.5 Rotations. The Configuration for the Loop block should now look like this:



Loop

0

Control: Sensor

Sensor: Rotation Sensor

Show: □ Counter

Port: ○ A ● B ○ C

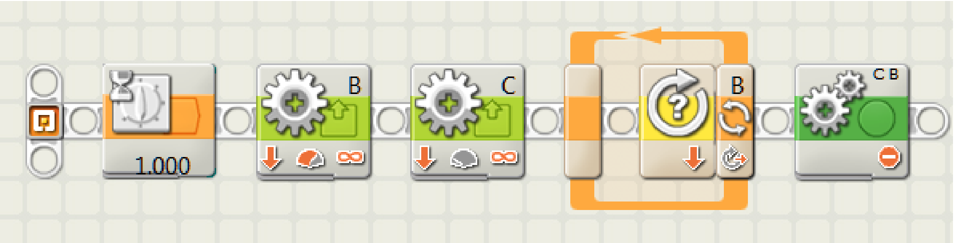
Action: ● Read ○ Reset

Until: ○ ↑ ● ↓ >

1.5 Rotations

Presuming you set everything right, this block will wait until the motor on Port B has rotated 1.5 rotations in Reverse, drawing as it goes. Once this has happened, the Loop block is finished, and then the next block to the right is given control. Since this is a Move block with a Stop function, the motors will stop.

Using a loop in this fashion is called an **external rotation counter**, since you're not using the built-in rotation sensing of a Move or Motor block. You could just as easily monitor rotations of Port C -- you get to choose which one is more convenient, *but be sure to set the direction of rotation appropriately*. Your program should look like this:



Perhaps you also noticed that you could have selected degrees instead of rotations -- remember there are **360 degrees in a circle**. *This is the amount that the specified motor rotates, not what portion of a circle it draws!*

□ **18.** Download and run this program. What portion of a circle did it draw? Modify the Rotation count value (in the Loop block) until it draws one full circle and no more. Record the values below:

Port B Power _____ Port C Power _____ Port B Rotation Count _______

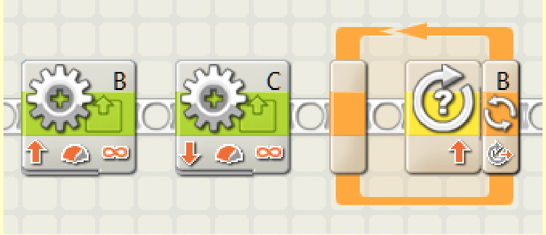
Port B Direction (Forward/Reverse) ____ Port C Direction (F/R) ____

□ **19.** Save this program under a useful name.

Why isn't my external rotation counter working?

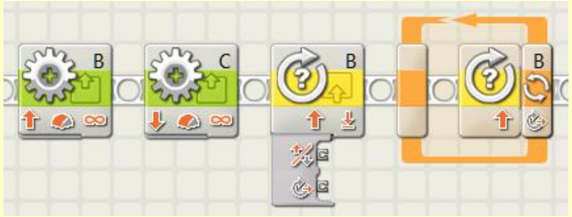
There's a problem that can show up when you use external rotation counting to control motion, particularly if you have a sequence of motions (move, turn, move again), but there's an easy work-around.

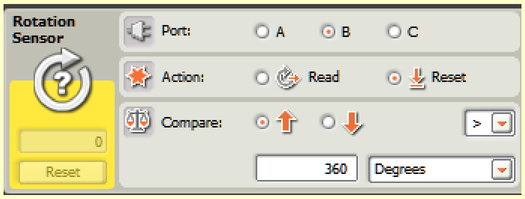
Say for instance that you are using external rotation counting of the B motor to control how far you turn, and that this is somewhere in the middle of a multi-move program:



If the B motor had recently turned, the loop might simply jump through rather than waiting for additional motion. ***This is because the NXT brick remembers that motor B already turned!***

The solution to this (and a good general programming practice) is to reset the rotation count for any motors you are sensing so that the rotation count starts fresh. To achieve this, insert a Rotation Sensor block (available from the Sensor item of the Complete Palette) before the loop, **set to the action to Reset**:




Rotation Sensor

0

Reset

Port: ○ A ● B ○ C

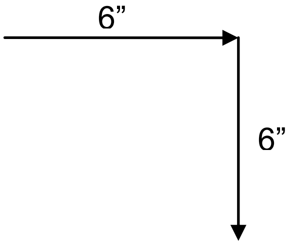
Action: ○ Read ● Reset

Compare: ● ○ >

360 Degrees

In the configuration panel don't forget to also set the Port to the appropriate motor port.

□ **20.** Using single Move blocks for straight motion (like you did in Project 3) and a pair of Motor blocks with external rotation counter for the turn, create a program that draws a 6” straight line, turns 90 degrees in place to the right and then draws another 6” straight line. The result should be two straight lines joined at a sharp right angle turn. ***Don’t forget to reset the sensor whenever you use external rotation counting.***



Write down the values for the Move blocks (for the 6” straight moves):

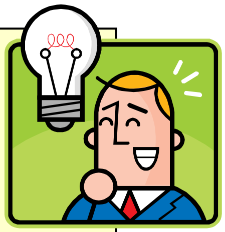
6” Straight Lines: Move block Power _____ Rotations _______

Spin 90 degrees to right (Motor blocks):

Port B Power _____ Port C Power _____ Port ___ Rotations ___

Port B Direction (Forward/Reverse) ____ Port C Direction (F/R) ____

Did you remember to enable (check) Control: Motor Power on the Motor blocks? Hope so!



Enrichment exercise 4.1:

You just made your robot move exactly 6” in a straight line.
Let’s use a bit of simple math to see if your results make sense.

Measure the Diameter of your driving wheel (d) = _____ inches

Multiply d by pi (3.14), this is the Circumference of the wheel (C) = ______ inches

For each turn of the wheel, your robot will move one circumference-worth of distance, let’s see how many rotations are needed to move 6”:

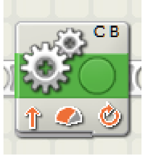
Calculated Rotations to move exactly 6” = ______ turns (Hint: divide 6 by C)

Now multiply YOUR Rotations (from your program) X Circumference

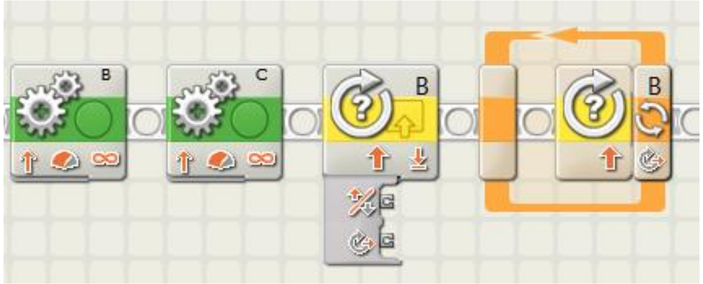
Your Rotations x C = _____________ inches

Question: Does it equal 6 inches (or pretty close)? *Hope so!*

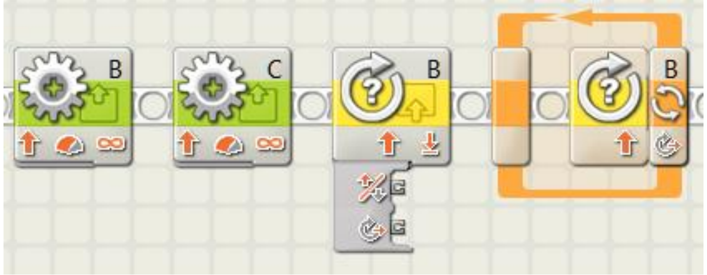
Hopefully by now you've realized that there's more than one way to make your FlexBot move. For example you want your robot to move a controlled distance, whether straight or arcing, you could try using a single Move block, with its automatic rotation control and steering:



Or you could use a pair of Move blocks with external rotation counting (notice the Reset before the loop):



Or you could use a pair of Motor blocks in a similar fashion, with Control Motor Power checked (as we've been doing in this project):



*But which one is the "**best**" for going straight or turning? Here's a "rule of thumb":*

To move **straight** a controlled distance, the best results generally come from the first example above, the simple Move block with built-in rotation counting. This is because when you use the **single Move block** command, the NXT brick "understands" that you want to go straight, and it makes automatic adjustments to keep it on course.

For the most accurate **turns and arcs**, you will get the best results using **two Motor blocks** (with external rotation counting) as shown immediately above.

This is a good moment to refer back to the end of Project 3 to review the chart that describes the differences between Move and Motor blocks.

□ **21.** *Let's put this to practice.*

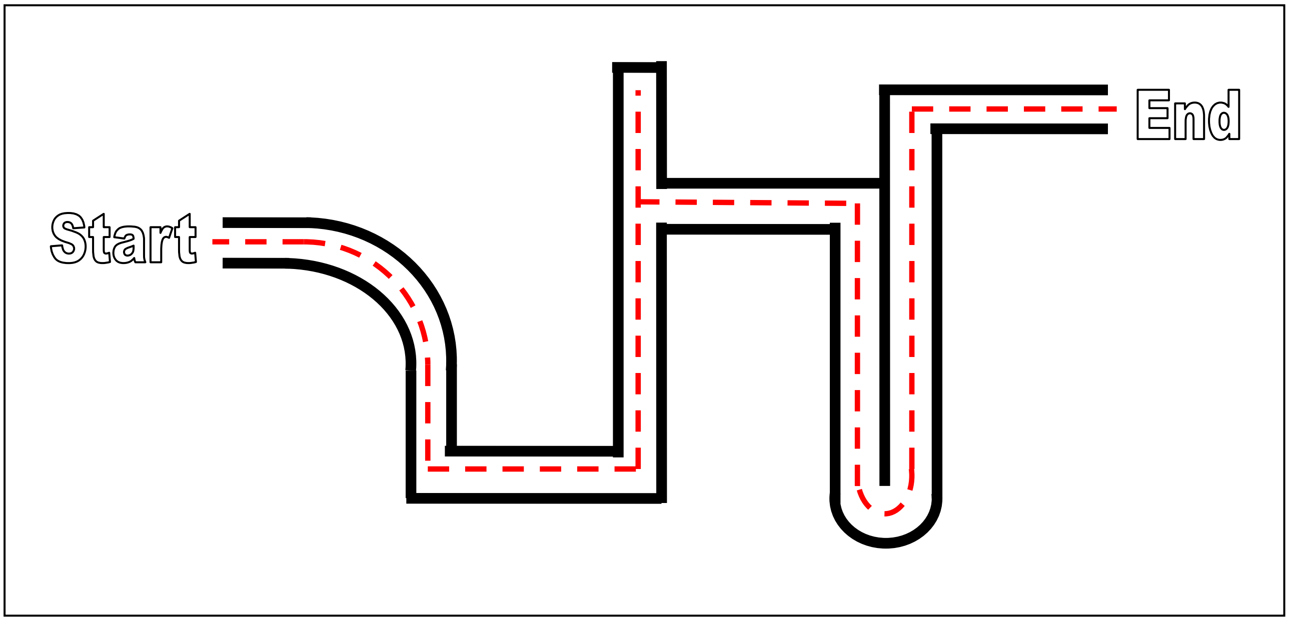
Your instructor will show you the Maze, a white board with two lines that describe the boundaries of a pathway.

Program your robot to turn and move (with a pen) so that the line is always between the lines. You MUST travel the <u>entire</u> path, *see the dotted line*. Be sure to use a dry-erase marker!

Ready, Set…Go! (Save your work along the way.)

Hint: If you reach what looks like a dead-end in the path, that means you are to stop, turn 180 degrees, and continue. Or of course, you could always travel backwards…

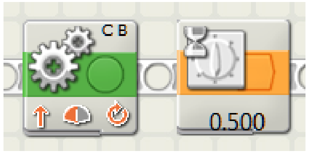
Here's an approximate picture of the maze. You are urged to measure the actual maze so you know how far to move and turn. ***Keep in mind that you <u>need</u> to use the SAME STARTING POINT for each run!***



Expert's Hints *– For the most repeatable runs*

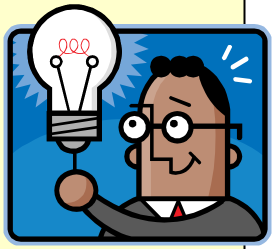
- Start your robot in precisely the same position and angle, and set your caster to the same orientation at the start of each run. This takes some discipline, but is worth the effort.

- Add a Wait block (.25 to .5 seconds) after each move (straight and turns) to give the robot time to settle:



Enrichment exercise 4.2:

For the more mathematically minded, here is a useful formula you can use to predict the approximate Diameter of the robot's turns:



D = b x (VR + VL) / (VR - VL)

Where

D = Diameter of robot turn
b = distance between drive wheels, center to center
VR = Velocity (Power) of Right wheel
VL = Velocity (Power) of Left wheel

Use the same units for b and D (inches). VR and VL can be negative or positive, depending on motor orientation in robot and Direction selected.

Note: if VR = VL, D is infinite. In this case the robot is moving in a straight line (think about it!)

Let's try an example:

Given that Port B is the Left wheel and Port C is the Right wheel, set the power of Port B to 50 and Port C to 10. If you measure the wheel center to wheel center dimension (b) you should find that it's about 5".

This yields:

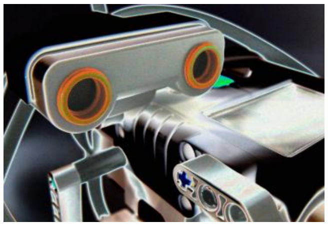
D = 5 x (10 + 50) / (10 – 50) = -7.5" (negative numbers mean turn to the Right).

What does your FlexBot do?

1. Try this experimentally with a few different powers, how far from calculated are the measured diameters? In a typical FlexBot the **TURNING DIAMETER ERROR** is often 5 percent or more from the calculated number due to a variety of factors.

2. Try reversing the direction of <u>one</u> motor and see what happens to the circles. What happens when the Power values are equal but the directions are opposite? (hint, see part 10 above).

3. What happens when one motor is stationary and other turns – what's the diameter of the turn? Try it!



NXT Programming for Beginners

Project 5:Simple Sensors

Detecting the world around the robot

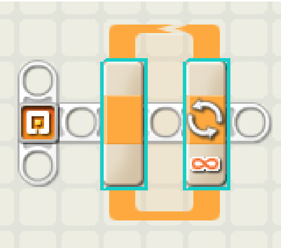
Just like your senses help you navigate and interact with your world, your FlexBot can also sense what's around it. Assuming you built your robot as described in Project 1, your robot has a vision sensor (looks down for dark/light areas), an ultrasonic sensor (detects how far to objects and walls) and a touch/bump sensor (detects when it is pressed).

To use these sensors your program needs to be able to read values from the sensor(s) and react accordingly. Of course, it's up to YOU to make the program, but it's really not that hard.

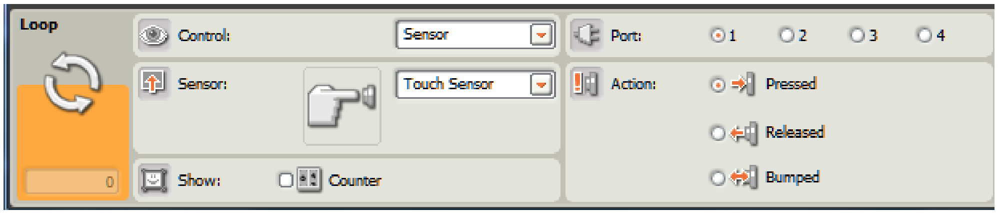
Touch/Bump Sensor

***Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil* ☑.**

□ **1.** Starting with a new program, add a Loop block as shown:



□ **2.** In the Configuration panel, set Control to Sensor, select the port that your touch sensor is plugged into (1 if you wired it right, now's the time to check), and make sure that "Pressed" is selected:



Loop

Control: Sensor

Sensor: Touch Sensor

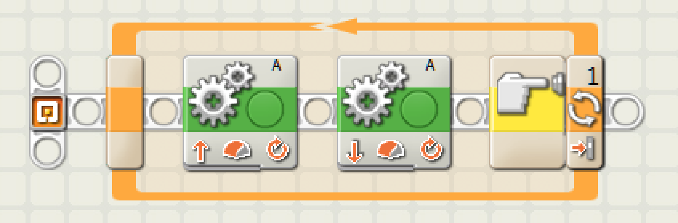
Show: □ Counter

Port: ● 1 ○ 2 ○ 3 ○ 4

Action: ● Pressed ○ Released ○ Bumped

With these settings, this loop will cycle repeatedly until the touch/bump sensor is pressed. Let's use this to control a motor action.

□ **3.** Insert the two Move blocks into the loop as shown. We will be using the "Arm" motor for this experiment, attached to Port A. Set the first Move block to .25 rotation (90 degrees) Forward, and the second Move block to .25 rotation Backwards, this will create a repeated up-down motion. Both should be set to Brake as the Next Action:



□ **4.** Upload this program to your FlexBot and run it. Does your Arm begin to "wag"? Press in and keep holding in the touch/bump sensor (don't allow it to push out). Does the Arm stop?

□ **5.** Run the program several times, this time with short pushes of the touch/bump sensor (push and release quickly). Does the robot seem to miss some of your pushes? Why do you think this is happening?

□ **6.** Change the loop to respond to "Bumped" instead of "Pressed", download and repeat step 5 above. Does your robot miss your pushes now?

□ **7.** Run the program in step 6, **but this time press and hold in the touch/bump sensor** (don't let it release). Does the wagging stop? While it's still running, release the sensor (allow it to push out). Does it stop now?

If things are working normally, you should be finding that there are some pretty important differences between "Pressed" and "Bumped". With Pressed, the loop will only notice pushes if the button is pushed at the precise moment when the loop logic is checking. Since the loop logic (in this program) only checks when a Motor block <u>isn't</u> busy, there's a good chance it will not see a short press of the button.

If you instead select "Bumped", the NXT Brick stores the fact that the push/release happened, and the loop logic will find out about it when it goes to check the next time around. However, as you'll see, Bumped has a pretty important limitation – *it requires both a press <u>AND</u> release to trigger your program.*

Here's are the options and what they do:

1. **Pressed** watches for a push of the button, but ignores releases.
2. **Released** watches for a release of the button, but ignores presses.
3. **Bumped** watches for press and then release of the button --it <u>only</u> acts on release <u>after</u> a press.

Depending on the purpose of your program, you need to make a logical choice about which option to choose.

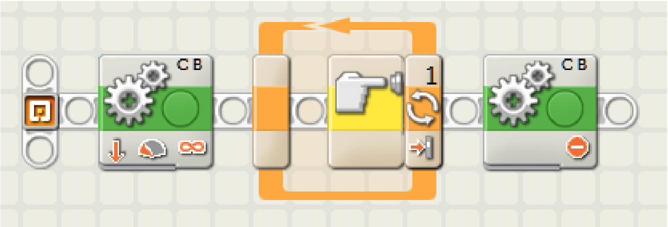
□ **8.** Save your program, give it a useful name.

□ **9.** Create a new program to make your FlexBot drive in the direction of its bump sensor and **stop** as soon as it hits something. Set the Power to 50 so that you're not moving too fast. Based on what you saw before, decide whether you want to use "Pressed", "Released" or "Bumped". Think about what happens when the robot pushes into an immovable object.

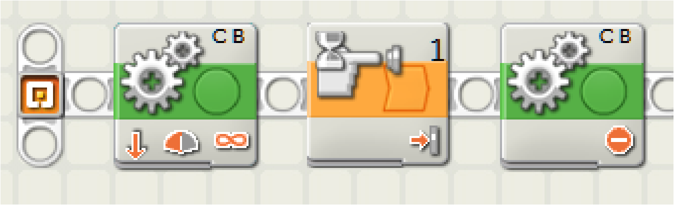
□ **10.** Download this program to your robot, place it in a location where going straight will cause it to hit an obstacle (like a wall) and run the program. Does it stop as you expected? Does it try to keep pushing even though it has hit the wall?

> If you want to stop a robot from going over the edge of a table, your sensor has to detect it early (and reliably) enough for your robot to react in time.
>
> You're now in the world of "**Real-Time Programming**" -- if you watch your robot carefully you can often get clues that tell you why your program is failing. For example, as you have already seen, you shouldn't use the "Bumped" option to protect from collision, *since the sensor will never* ***release*** *once the robot hits the wall.*

Here is an example program that can do step 9 above. Is it different from yours?



Here is yet another version that has the same effect -- it uses a **Wait Touch block** (in the Wait flyout). Much like the loop version above, it sits and waits for the appropriate trigger to occur before proceeding on with the next block:



There are Wait blocks for ***Time****,* ***Touch****,* ***Light****,* ***Sound****,* ***Distance*** *(using the ultrasonic sensor) and* ***Color*** *(using the light sensor). Sadly, there is NOT a Wait block for rotation sensors.*

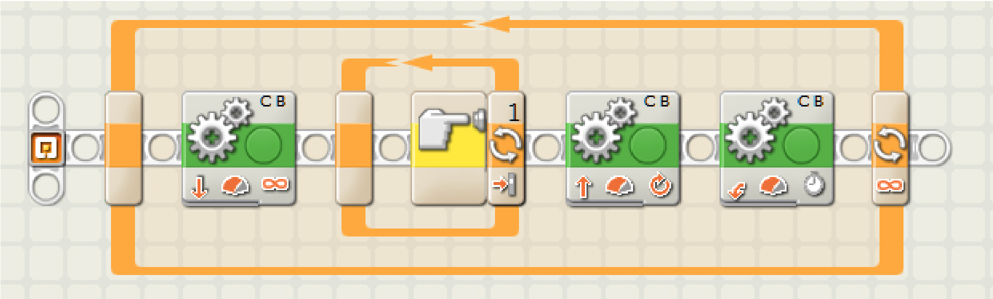
□ **11.** Modify your program from step 9 to back up a few inches and turn around and go the other way when it collides with a wall or object. (Hint: You may want to put a loop within a loop.)

□ **12.** Download and run your program.

Does your robot adapt to changing situations and never get stuck?

Do you need to adjust the times, powers, etc. to make it work better? Try it now!

If you get stuck, here's an example program that does the above:



Look at the above program to figure out what each step does. What happens when we have a loop within a loop?

Is it different from how you programmed your robot?

□ **13.** Save this program, give it a useful name.

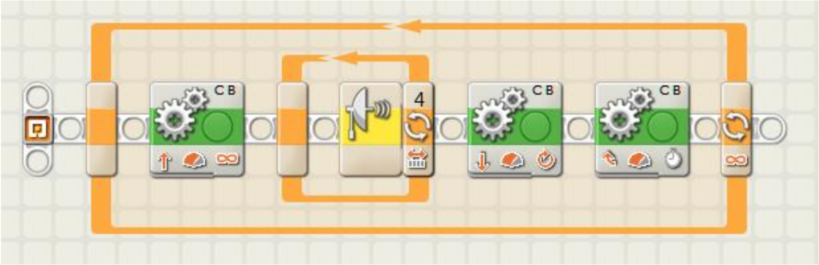
Ultrasonic Sensor

Now let's learn how to use the ultrasonic sensor. This sensor sends out a short burst of sound (a "ping") and waits for the reflected sound to return. The sensor then measures how much time it took for the reflection to come back. The longer the time, the further away the nearest object is.

Starting with the program in step 12 above, reverse the motor directions in the Move blocks, change the inner loop to use the "Ultrasonic Sensor", select the appropriate Port (follow the wire), and set it to "Until: Distance < 20 cm".

□ **14.** Download and run the program. Does your robot successfully detect and avoid walls without contact? Hope so!

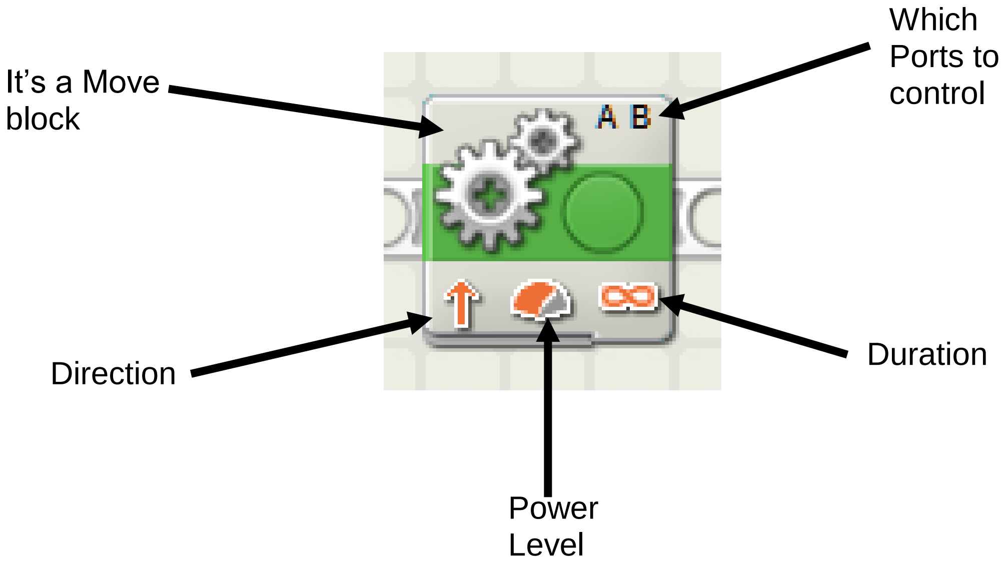
Here is a version of a program you might use to do the above:



□ ***15.*** Save this program under a useful name.

Have you noticed that the blocks in your programs often have little pictures that tell what they are set to do? Let's have a look at one of them, a Move block, to see what it's up to.

Here is a typical Move block as shown in your program, with notations:



□ **16.** Answer the following questions, write in the spaces provided:

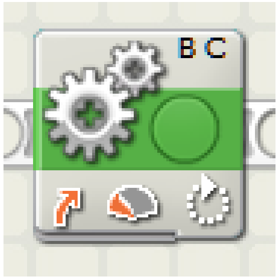
Which motor ports are controlled by this Move block? _____ *and* _____

What direction will it go (Forward or Backwards)? _______________

What is the approximate power level, from 0 to 100? ____________

For how long will it move (Duration)? ________________________

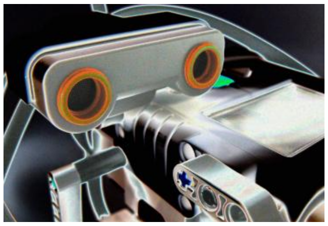
□ **17.** Use the Configuration panel on a Move block. Are you able to make it look like this:



Edit as many options as necessary to exactly match the above picture.

Were you able to do it?

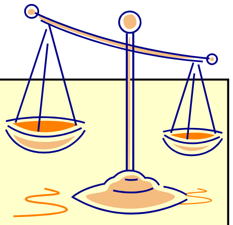
What options did you need to select?



NXT Programming for Beginners

Project 6:Stop at the Line

First "exposure" to using a line sensor in your programs



You are about to use the **Light Sensor** on your FlexBot to detect a line. Before you can use the sensor, it needs initial calibration. *For best results, do this in an area that is lighted similarly to how it will be when you're using your programs.*

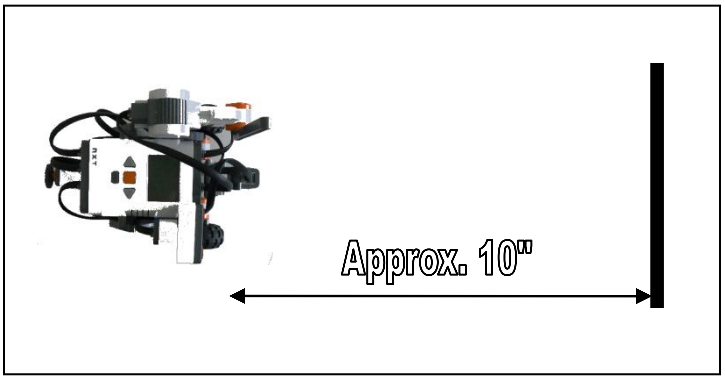
1. Hook up your FlexBot to your PC and turn it on.
2. Place your FlexBot on the table or board, with some portion of it covered in dark tape.
3. In the NXT software, select Tools – Calibrate Sensors.
4. Choose Light Sensor, and choose the correct sensor port (in the FlexBot it's Port 3)
5. Click on "Calibrate", the software will download a program called "Calibrate" to the NXT and run it.
6. Read the on-screen instructions, you will place the Light sensor over the dark region (tape) and press the Orange button, then place it over the light region and press the Orange button again.
7. Once you see the confirmation message, your light sensor is calibrated. You can repeat this again at any time.

In this project you program your robot to stop when it sees a dark line. There are <u>many</u> ways to do this, from simple to more complex. In the first part of this project you will use a few of the very simple approaches.

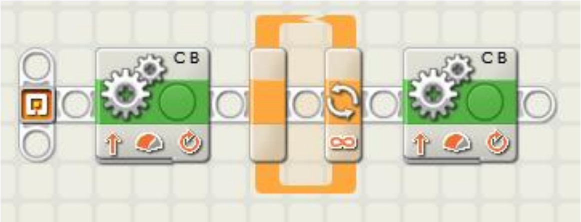
You will then learn to do it in a way that gives you greater understanding of process control and handling numbers. This is good preparation for Project 9 and beyond, where you will need these skills.

***Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil* ☑.**

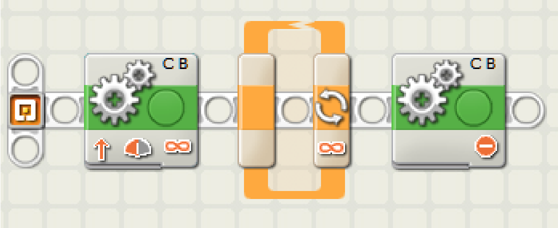
□ **1.** Prepare your work area as shown below. You will need room for your FlexBot to travel in a straight line, and there should be a line across its path.



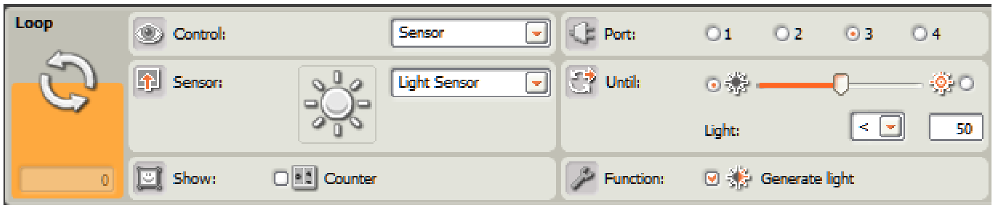
□ **2.** Starting with a new program, add a Move block, a Loop block, and another Move block as shown:



□ **3.** Set the first Move block to Duration - Unlimited, Power - 50 and the second Move block to Stop:



□ **4.** In the Configuration panel for the Loop block, set it to Control - Sensor, Sensor - Light Sensor, and Until - Light < 50. Make sure that Function - Generate Light is checked and that you've used the right Port number (3):



Loop

0

Control: Sensor

Sensor: Light Sensor

Show: □ Counter

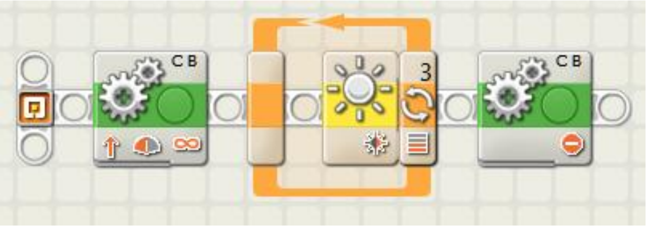
Port: ○ 1 ○ 2 ● 3 ○ 4

Until:

Light: < 50

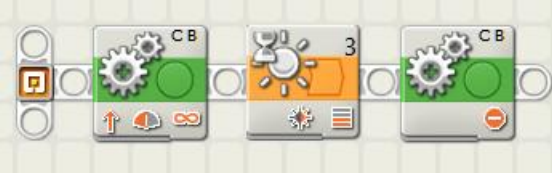
Function: ☑ Generate light

□ **5.** Your program should now look like this:



□ **6.** Download and test. Does your robot stop at the line? If not, debug your program and re-test. Save the final program under a sensible name.

□ ***7.*** Modify your program, delete the Loop block and replace it with a Wait – Light block. Make sure the Port number is correct, and set the Wait – Light block to trigger when the light is < 50:

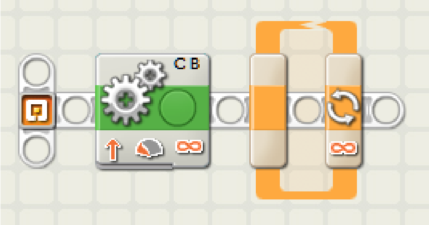


The Wait block does essentially the same thing as the Loop block in the prior example -- your program should behave the same. *The Wait block waits for the specified trigger and then proceeds to the next step.*

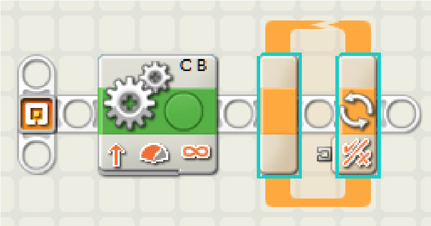
□ **8.** Download and test. Does your robot stop at the line? If not, debug your program and re-test. Save the final program under a sensible name.

As promised, we're about to get a bit more advanced. In this next exercise you will read from your light sensor, compare it with a numeric value and send the result into a special decision block, called a "Switch block". This is not the simplest way to do this specific task, but these are important skills. You will use this technique in Projects 9 and beyond.

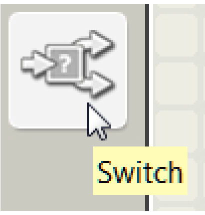
□ **9.** Create a new program; add blocks to it as shown (a Move block and a Loop). Set your move block to move forward indefinitely, using the usual B and C ports, and set your power to 35.

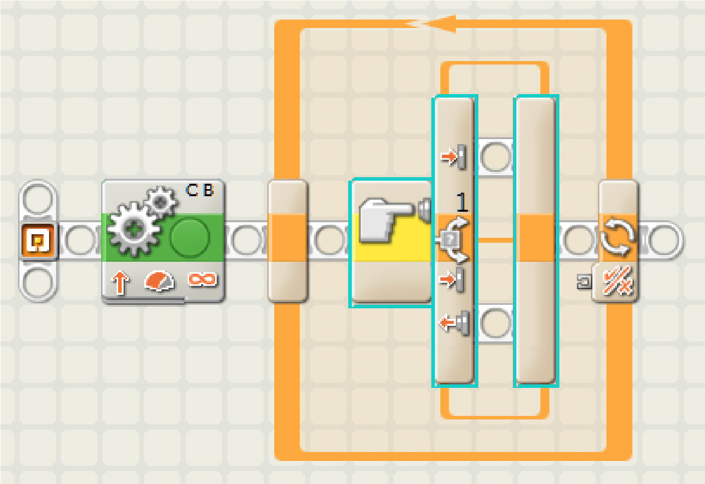


□ **10.** With the Configuration panel set the Loop block to Control: Logic. This will allow you to exit the loop when necessary. The program now appears as follows:



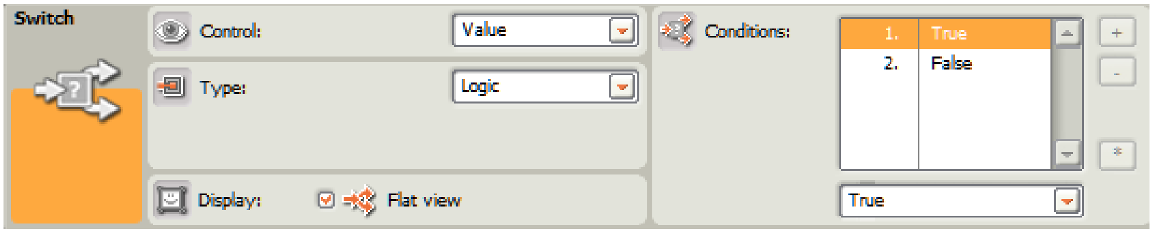
□ **11.** From the Programming palette select the Switch block, place the block into the Loop Block in your program as shown.





Note that the Switch block also has the ability to read values from a sensor directly. While this is useful, in this exercise we're going to give the Switch block a value that we provide.

□ **12.** In the Configuration panel of the Switch block, set it as shown below. Notice that making the settings changes the appearance of the Switch block icon in your program.



Switch

Control: Value

Type: Logic

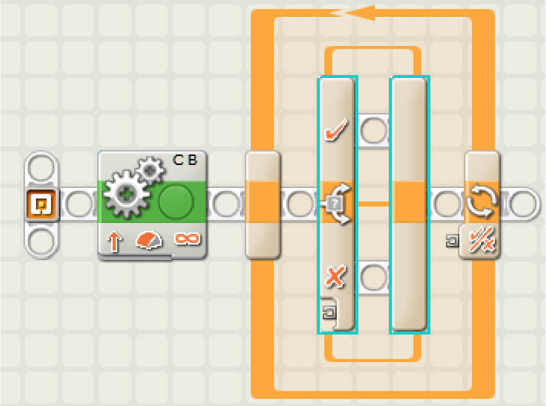
Display: ☑ Flat view

Conditions:

1.	True
2.	False

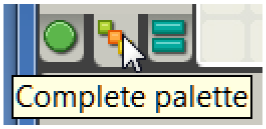
True

Once you have made these changes, your program looks like this:

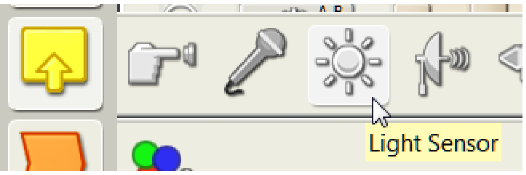


The Switch block (inside the Loop) will execute one of its groups of action blocks (or "rows"), based on its input. In this example there are two Conditions: True and False. Notice that there are two rows in your switch statement – the upper one next to a check mark ✓, and the lower one next to an X mark ✗. These rows are where you put the program blocks for True and False inputs respectively.

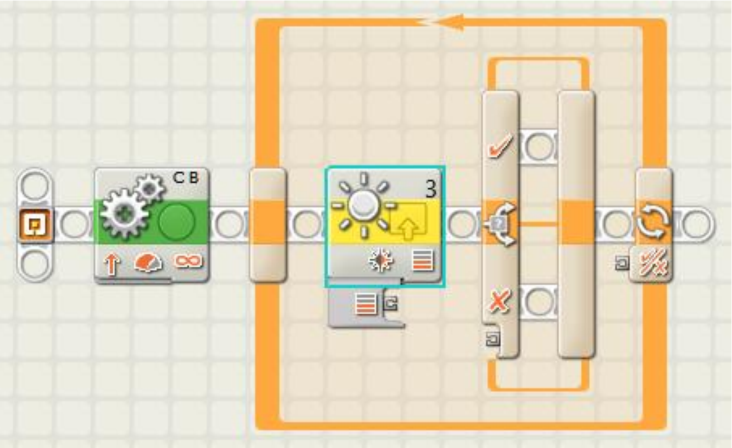
□ **13.** In the Programming palette select the “Complete Palette” as shown. This will allow you to select more action blocks:



□ **14.** In the Sensors flyout, select Light Sensor as shown:

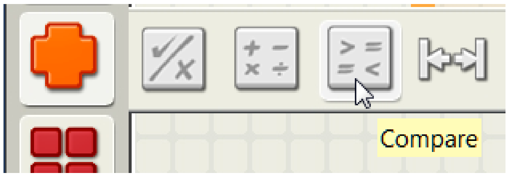


□ **15.** Place a Light Sensor block into your program as shown:

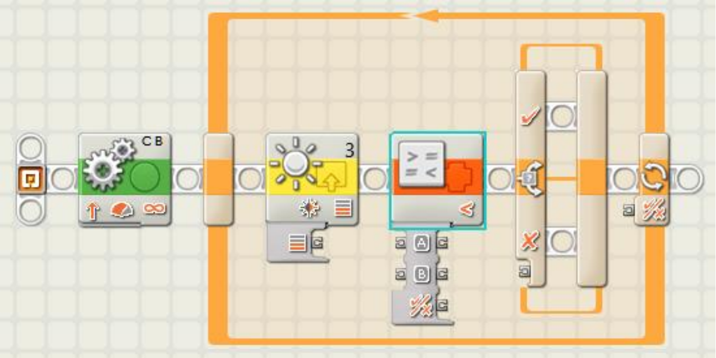


□ **16.** In the Configuration panel for the Light Sensor block, make sure that it is set to the Port to which your light sensor is attached, and that Generate Light is selected. We will not be using the Light Sensor’s built-in comparison, so you may ignore the Compare section.

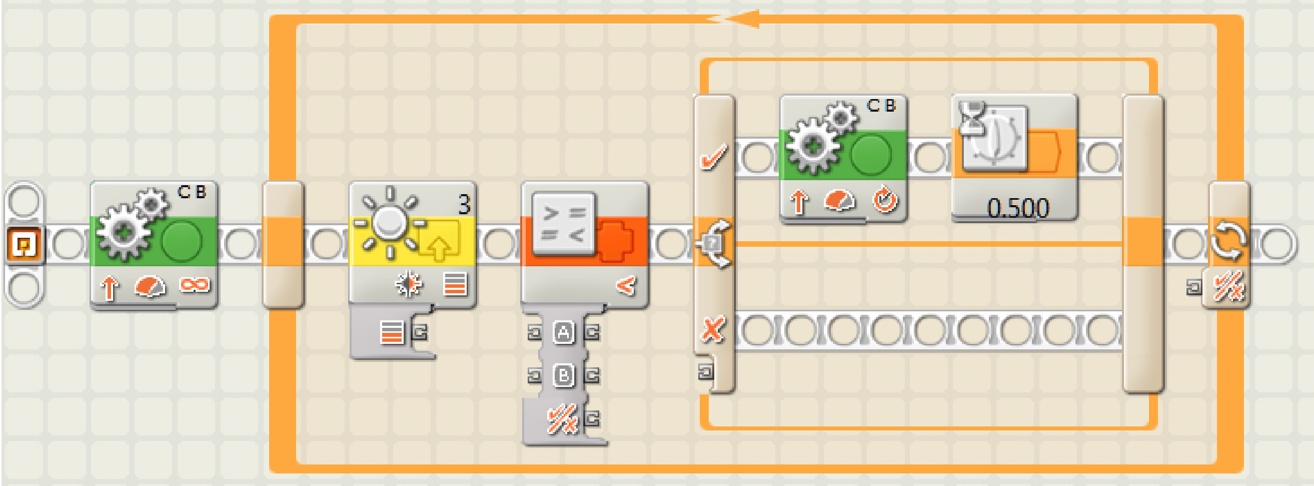
□ **17.** From the Data palette flyout menu, select the Compare block:



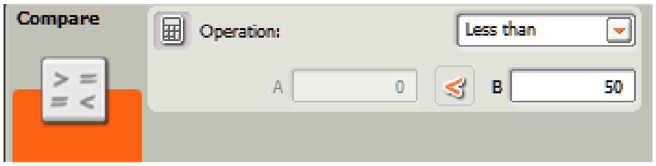
Add a compare block to your program as shown:



□ **18.** In the Switch block **row** next to the check mark ✓ you place action blocks you want executed if the input to the Switch block is True. Keeping in mind that you want your robot to stop when it reaches a dark line, place in the True row a Move block, with Stop as the operation. To allow the robot to settle, place a short wait (.5 sec) immediately after it as shown:



□ **19.** In the Configuration panel for the Compare block, set the Operation to “Less than” and enter a value of 50 in the B entry field as shown:



Compare

Operation: Less than

A 0 B 50

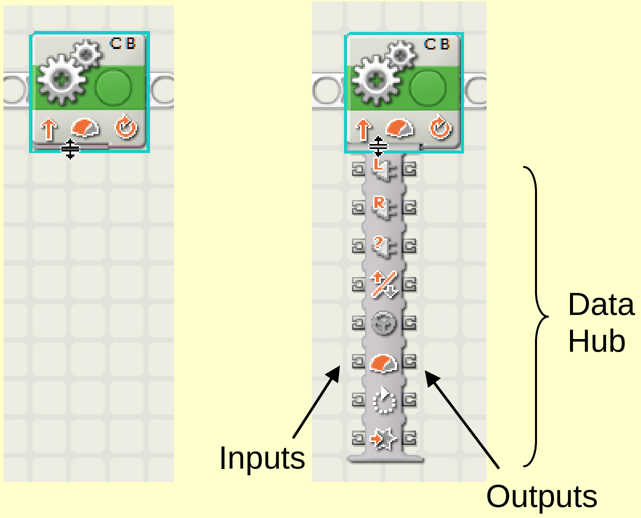
This will cause the Compare block to output a True value only if the incoming A value (from the light sensor) is less than 50 (dark). Otherwise the Compare block will output False.You now have a loop that can read from the light sensor, do a comparison, and perform different operations based on the light reading. **However, they are not connected yet** -- we will now “wire” the blocks together, so that data can be passed from one to another.

We'll be running some wires in the next few steps.

More about Data Hubs

That odd looking thing hanging down from some action blocks is the Data Hub; it's how **data** (numbers, logical values, strings, etc.) flows in and out of the block. As you'll soon see, you connect wires to specific plugs on the Data Hub – *each plug has a different meaning.*

You may find times when you want to show or hide a Data Hub -- this is pretty easy to do. Shown on the left below is a typical Move block, notice that the mouse pointer is above a small indentation, or "tab", in the lower left of the block, and that it has changed appearance. If you click on that tab, the Data Hub opens up (down actually) as shown in the right picture. Click again on the tab and it closes.

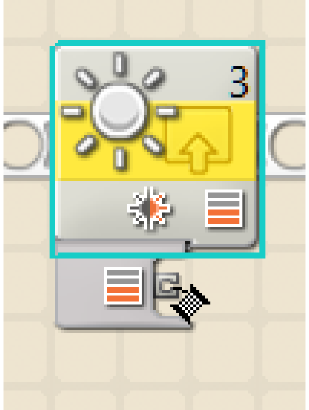


The plugs on the **left** of the Data Hub are the **Inputs** (data going into the block) and the plugs on the **right** are the **Outputs** (data coming out of the block). *If you're not using the Data hub, it's usually best to close (or minimize) the Data Hub so that your program appears as simple as possible. You can always open it again later.*

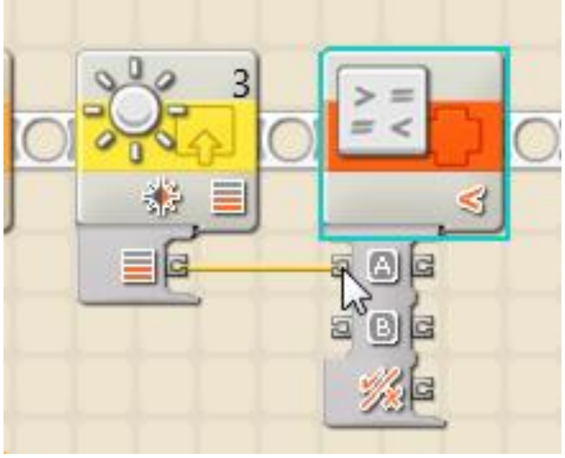
If you have a wire hooked up to one or more plugs, they will remain visible when you minimize the Data Hub view. Practice opening and closing the Data Hub on any block, see what happens.

To read more about the function of each plug on the Data Hub, check out the context-sensitive help system (click on More Help).

□ **20.** Hover your mouse over the output “plug” of the Light Sensor block as shown (don’t click yet). As you do this notice your mouse pointer will change appearance as shown. This means that it is ready to run a wire from the plug on this “**Data Hub**” to some other location:

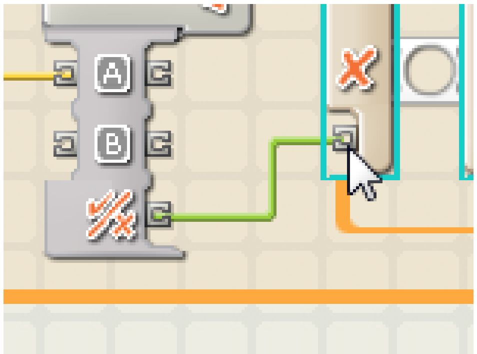


Insert a wire as shown to from the output plug of the Light Sensor to the “A” input plug of the Compare block by clicking and dragging a wire from one plug to the other:

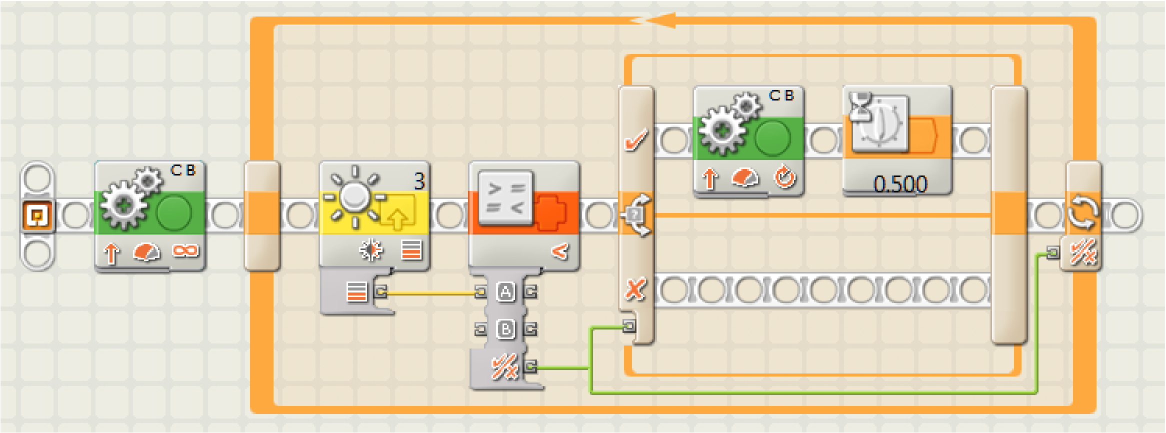


This will send the light readings to the A input of the Compare block.

□ **21.** Wire the output of the Compare block to the input plug of the Switch block as shown:



□ **22.** Run another wire from the same output of the compare block to the logic input of the loop. This will allow the loop to exit when the condition TRUE is met. You have now finished the wiring of the logic. Confirm that your program appears as shown below, download and test.



Does the robot stop at the line?

If not, carefully review your program to find the problem. You may need to adjust the value in the Compare block, this is your “trigger” point for detecting the line.

□ **23.** Save this program under a useful name.

□ **24.** Write a program of your own design that will go to a line, stop, turn around to face the other way, and travel six inches away from the line.

□ **25.** Download and test your program.

Were you able to get the robot to point accurately the other way (180 degrees turn)?

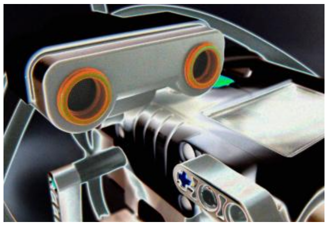
What changes did you need to make to your program to add the new logic?

□ **26.** Save this program under a useful name.

Congratulations!

You have now used three different methods with your line sensor to detect and react to a line.

Let’s move on to Project 7, “Using MyBlock”.



NXT Programming for Beginners

Project 7:Using MyBlock

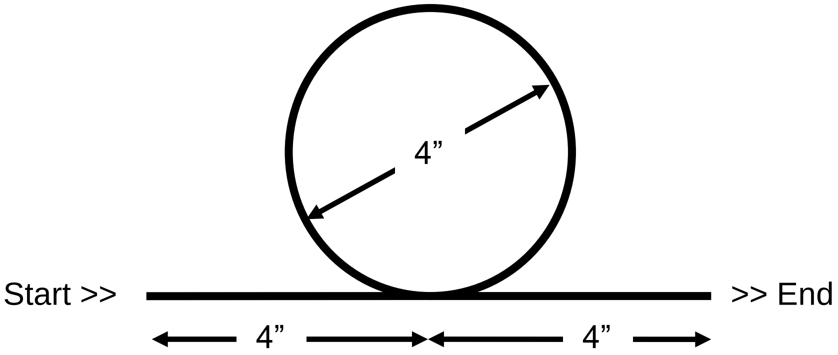
How to keep your programs small and simple

As your programs get longer and more complicated, you might find that there are certain groups of blocks that are repeated. You probably saw this in project 4 where you navigated the maze.

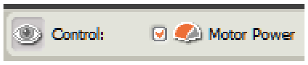
Wouldn't it be great if there was a way to make these groups into re-usable "modules" that you can use, instead of having to re-write them again and again. Thankfully, there is a way to do this through the "**MyBlock**" feature of the NXT language. In this project you will write a program that uses your own custom MyBlocks to repeat actions in an efficient way. In other languages this might be called a "subroutine". It might also be considered a LabView "Virtual Instrument" or "VI", since Labview is at the heart of the NXT software.

Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

□ **1.** Starting with a new program and using what you learned in Project 4, add code blocks to draw the following figure. Download and test your program, using a pen. Fix as necessary to get the measurements within plus/minus ¼" from the dimensions shown. Also, do your best to make the Start and End line create a straight line:

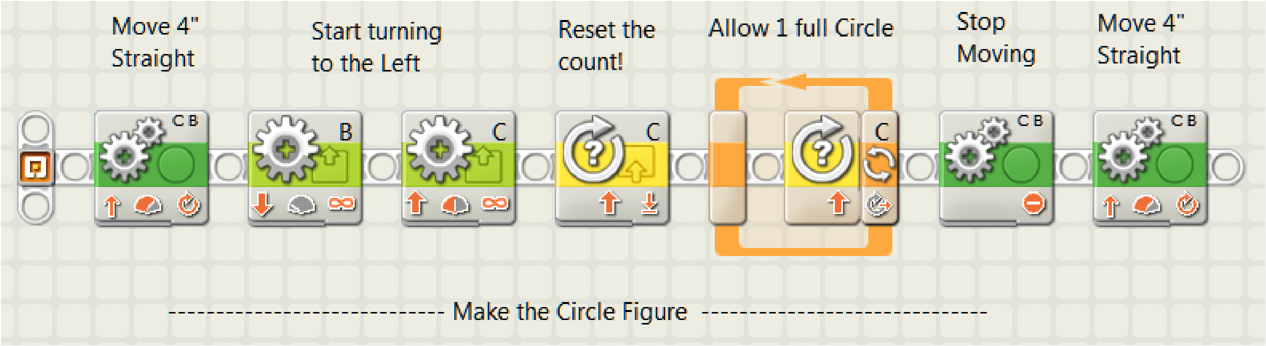


***A general rule**: For the most accurate turns, use **Motor** blocks with Control: Motor Power **CHECKED**:*



Control: ☑ Motor Power

□ **2.** Here is a possible circle routine for the above. ***Notice the text comments that have been added above and below the blocks. This is easy for you to do (just double click somewhere and start typing), and it's a <u>big</u> help in explaining what's happening!***



Move 4" Straight

Start turning to the Left

Reset the count!

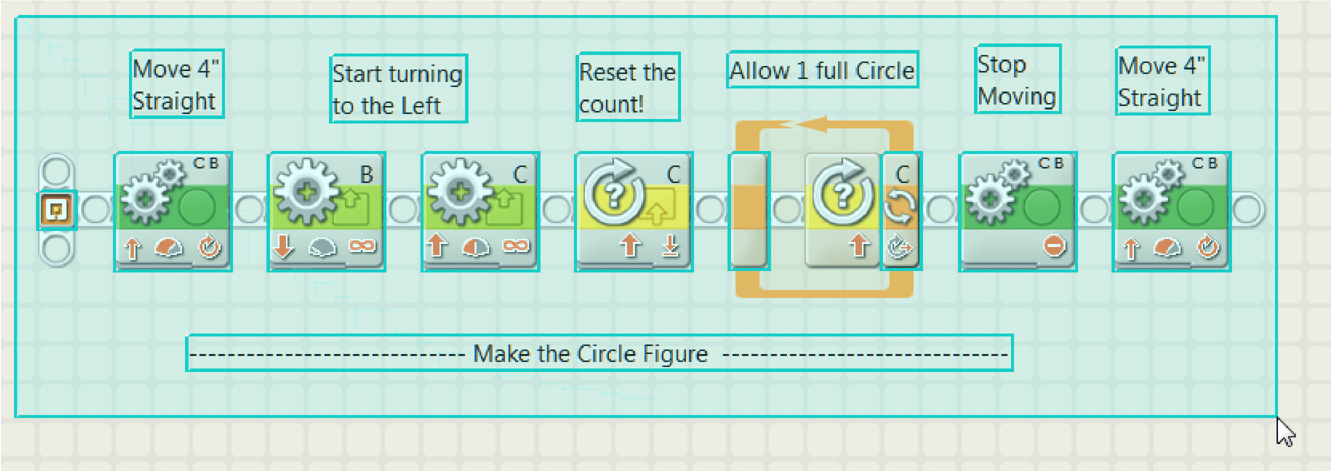
Allow 1 full Circle

Stop Moving

Move 4" Straight

---------------------------- Make the Circle Figure -----------------------------

□ **3.** Click on the left mouse button and drag your mouse around the group of blocks that draw the first line, circle and final line. Also enclose the text comments above and below the blocks. We are going to convert this whole section of our program into a **MyBlock** that we can use again and again.



Move 4" Straight

Start turning to the Left

Reset the count!

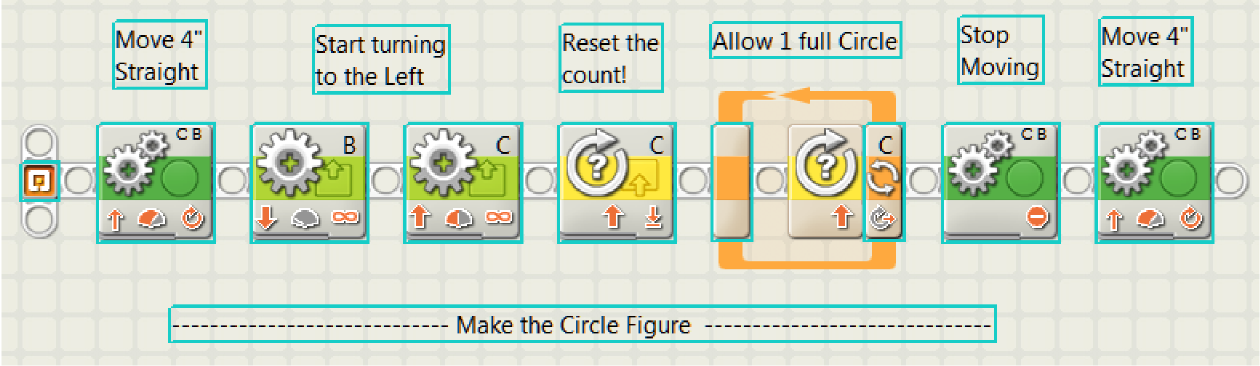
Allow 1 full Circle

Stop Moving

Move 4" Straight

---------------------------- Make the Circle Figure -----------------------------

□ **4.** When you release your mouse button the screen should look like this, with all of the selected items surrounded by a blue rectangle:



Move 4" Straight

Start turning to the Left

Reset the count!

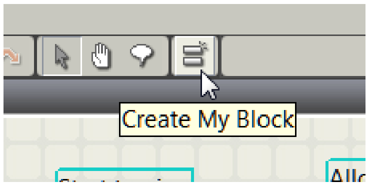
Allow 1 full Circle

Stop Moving

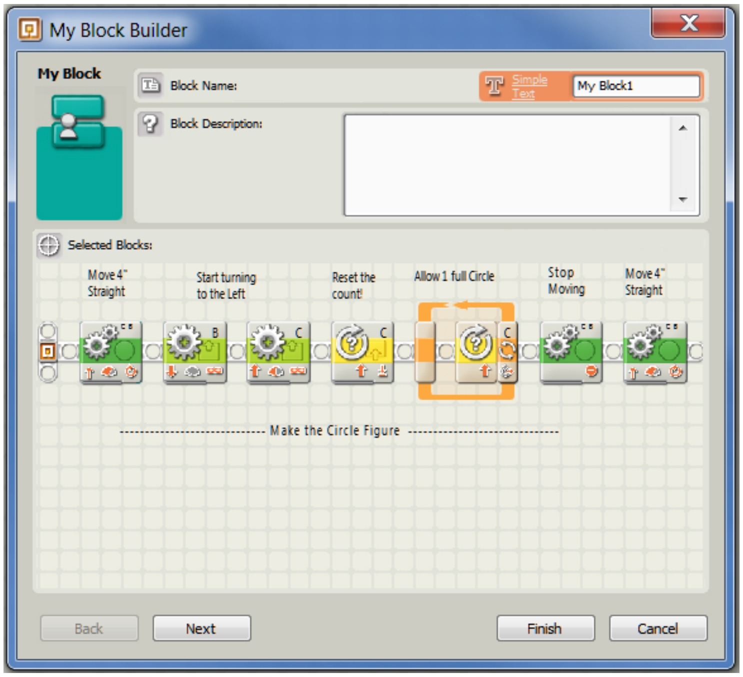
Move 4" Straight

---------------------------- Make the Circle Figure -----------------------------

□ **5.** Click on the “Create MyBlock” icon in the toolbar as shown (you can also select *Edit-Make a New Myblock*):



□ **6.** A window will appear that looks like this, displaying all of the blocks and text that are now part of this MyBlock:



Block Name: My Block1

Block Description:

Move 4" Straight

Start turning to the Left

Reset the count!

Allow 1 full Circle

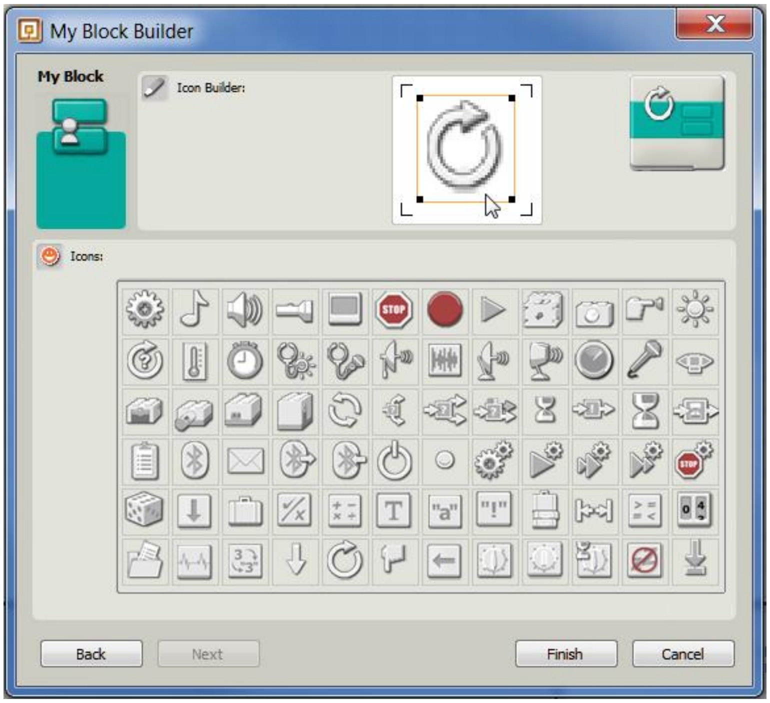
Stop Moving

Move 4" Straight

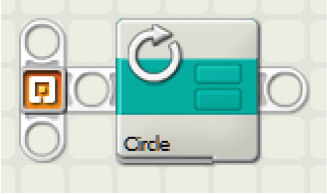
Make the Circle Figure

Replace the words “My Block1” with a name that describes what the block does, such as “Circle”. Press Next.

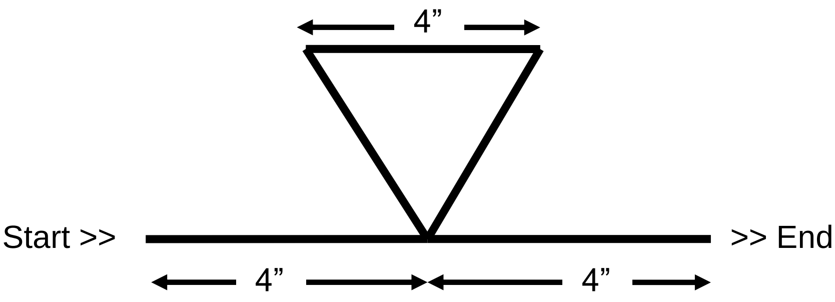
□ **7.** You now see the Icon Builder, where you can select an Icon to use with your new MyBlock. Since this is a Circle drawing routine, select an icon that is circular and drag it to the workspace near the top of the window as shown. Press Finish when done:



□ **8.** Your program now looks like this, where a single MyBlock has replaced all of those other blocks:

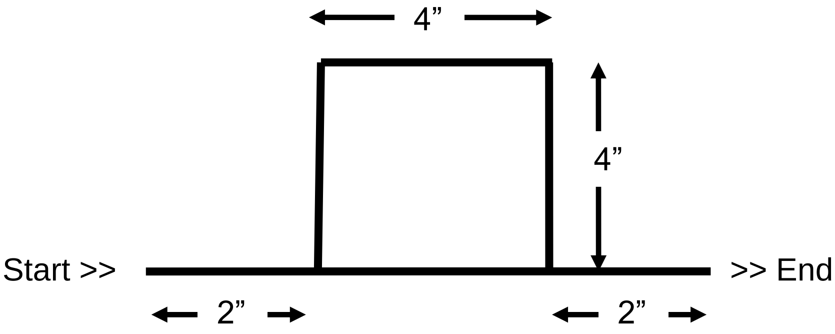


□ **9.** In your program, add blocks after the Circle block to draw the following figure (each side of the triangle is 4”). Download and test your program, using a pen. Fix as necessary:



□ **10.** As before, create a MyBlock for your triangle code. Give it an icon that suggests what it does.

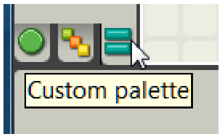
□ **11.** In your program, add blocks to draw the following figure. Download and test your program, using a pen. Fix as necessary:



□ **12.** Create a MyBlock for your square code. You now have three custom MyBlocks to use in your programming!

□ **13.** Save this program under a useful name.

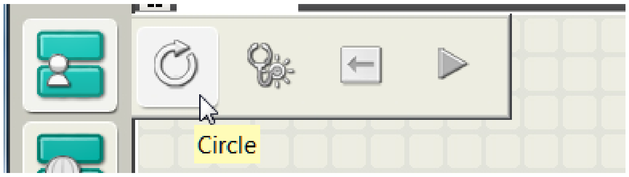
□ **14.** Click on the rightmost tab of the Programming Palette Selector as shown:



You now have access to your newly created MyBlocks. You may also see MyBlocks from other programs, ignore them for now.

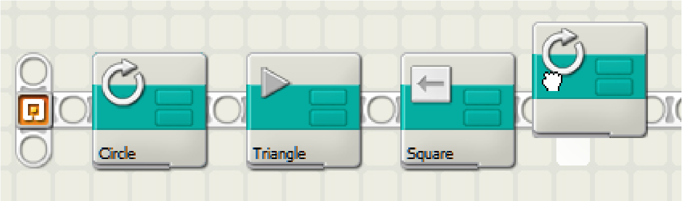
□ **15.** Click on the top item in the Programming Palette, this is your MyBlocks selector – a flyout menu will appear with all of your MyBlocks, presented in alphabetic order by name.

Notice that in this picture there's a fourth MyBlock (the second one), this was developed for another program. As you develop more MyBlocks, they ALL show up in your menu. ***This means that you can create MyBlocks and use them in multiple programs!***



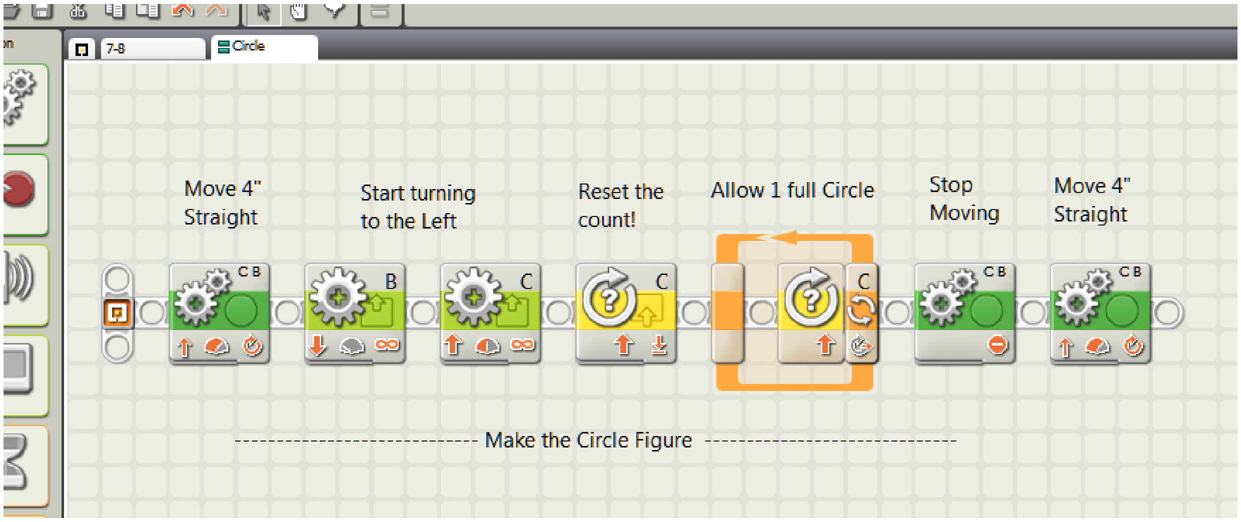
If you click on a MyBlock you can now drag it into your program.

□ **16.** Since your program has only three blocks (why?), drag a Circle MyBlock into place as shown:



□ **17.** Once you are satisfied with the order, download and run the program. Does it draw the shapes you expected? Hope so!

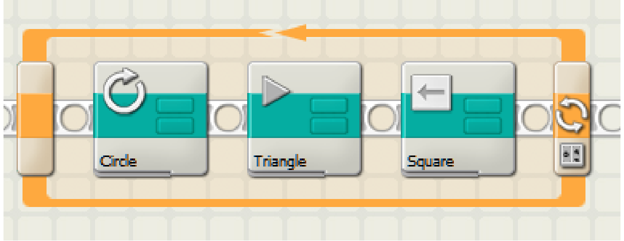
□ **18.** If you are not satisfied with how one of your MyBlocks is working and want to change it, double click on one of the "instances" in your program (for example, click on a Circle MyBlock). A new tab opens in which you see the original blocks and comments that you selected for this MyBlock.


7-8
Circle
Move 4" Straight
Start turning to the Left
Reset the count!
Allow 1 full Circle
Stop Moving
Move 4" Straight
C B
B
C
C
C
C B
C B
---------------------------- Make the Circle Figure ----------------------------

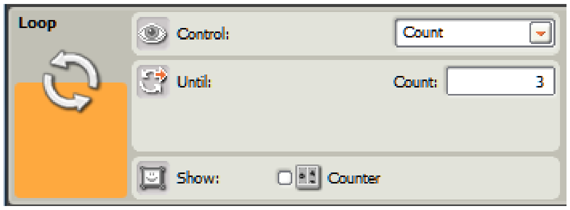
You can edit this code – and **here's a bit of magic**. *If you change the code in a MyBlock, every place it is used in your program(s) will automatically get the same changes! In other words, be CAREFUL!*

□ **19.** Save this program under a useful name.

□ **20.** Starting with a new program, add MyBlocks to draw three groups of Circle, Triangle, Square in a row. Instead of entering 9 MyBlocks, create a loop that will repeat three times as shown:



In the Configuration panel for this Loop block, select Count and set the Count to 3. This loop will now repeat everything inside it three times before moving on:



Loop

Control: Count

Until: Count: 3

Show: □ Counter

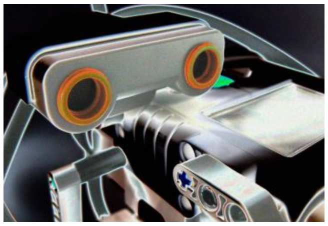
□ **21.** *Now it's your turn.* Create a new MyBlock that will draw a Hexagon (six sides), and then use it in a new program that draws five groups of Hexagon then Square groups of figures.

Your program is only allowed three action blocks!

Congratulations you have now learned the basics of using MyBlocks.

Let's really put it to work in…

Project 8 – Running Ragged (some call it "suicide")



NXT Programming for Beginners

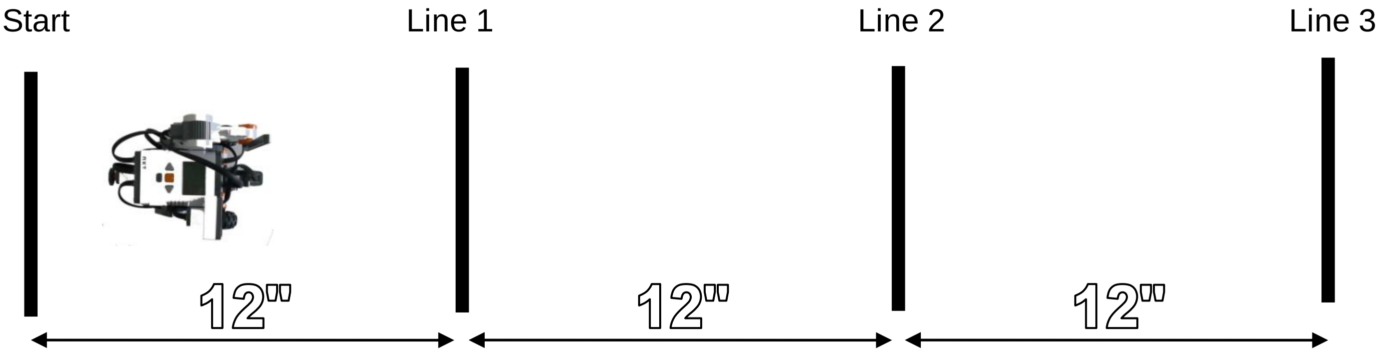
Project 8:Running Ragged (some call it "suicide")

More structure to your programs

Let's put your knowledge of line detection and MyBlocks to good use. In this project you will be moving from line to line, repeating actions as necessary. You will also learn about "**pseudo-code**", where you write out what you want to do in non-computer-language, then convert it to programming steps.

Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

□ **1.** Prepare your test track as shown below.



□ **2.** Here are the steps your robot will follow. Study them to understand what's required:

A. Beginning in front of Start, travel to Line 1.
B. At Line 1, turn around and return to Start.
C. At Start, turn around and travel to Line 2.
D. At Line 2, turn around and travel to Start.
E. At Start turn around and travel to Line 3
F. At Line 3, turn around and travel to Start.
G. At Start, stop.
H. Do a little dance while playing music (not you, silly, your robot!)

□ **3.** Write out a series of simple-language steps describing what you want to do. We have started the process for you here. Each item in this list is called a “simple command”, ***notice how they repeat:***

1. Move forward to line and turn around
2. Move forward to line and turn around
3. Move forward to line and keep going far enough to pass line
4. Move forward to line and turn around
5. Move forward to line and keep going far enough to pass line

.
.
.

Write out your “pseudo-code” on a separate piece of paper. Make sure you do it a way that repeats simple commands such as shown above.

□ **4.** Walk through your pseudo-code in your mind, making sure that it achieves everything in step 2 above. If it helps, put your finger on the picture in step 1 and move it as if you are a robot.

□ **5.** Create a program that achieves one of the simple commands from your pseudo-code using NXT action blocks. Download and test that simple command on your FlexBot to confirm that it works.

□ **6.** Convert the blocks in that program to a MyBlock, give the MyBlock a useful name and icon. Save the MyBlock.

□ **7.** Create a new program that achieves the next unique simple command, test and convert to MyBlock. Again be sure to save the MyBlock.

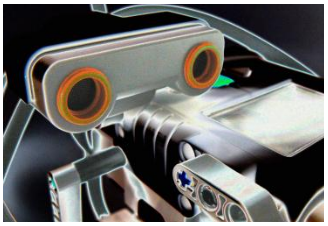
□ **8.** Repeat this process (if needed) until you have a MyBlock for each unique simple command from your pseudo-code. Once this is done you have a “library” of MyBlocks that you can use to complete the project.

□ **9.** Create a new program that strings together calls to your MyBlocks to fully perform the routine from step 2 above. Download and test, fix if necessary.

Think about how many action blocks this would have taken if you didn’t use MyBlocks. (Answer: A WHOLE BUNCH!!!)

Congratulations, you have now done a nice bit of “structured programming”.

Now let’s move on to Project 9 where you will learn about
Automatic Sensor Calibration



NXT Programming for Beginners
Project 9: Automatic Sensor Calibration

More advanced use of data

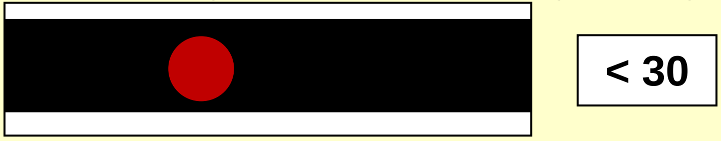
Sometimes you need to save information, do calculations and make decisions based on a combination of factors. You'll be glad to know that the NXT software is able to do all of this and more, and with ease.

But first, let's review how a Light sensor can be used to find the EDGE of a line.

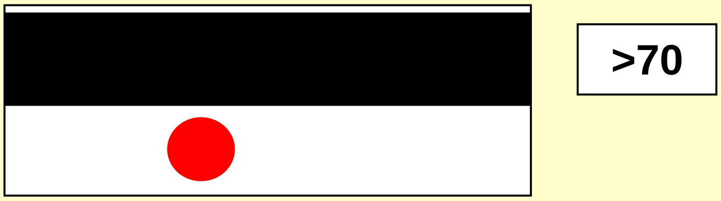
Your **Light sensor** can project a beam of light and measure how much light is reflected back. The light forms a circle -- the size of the circle is controlled by how FAR the sensor is from the surface. The amount of light reflected back tells you how dark or light the surface is.
Low numbers are dark, high numbers are light.

Note: You must have performed the initial calibration of your Light sensor as described in Project 6 for this project to work. If you haven't done so, please do it now.

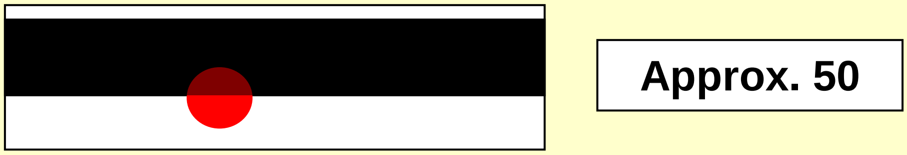
In this illustration the Line sensor is directly over the middle of a dark line. Notice how the circle of light is completely within the line. Since the circle of light is on a dark area, the light reading will be a low number.



In this next illustration, the Light sensor has moved completely off the line, now projecting on the white area. Now with the circle of light on a light area, the reading should be much higher.



Lastly, if the circle of light is exactly on the edge of the line, the reading is an equal combination of light and dark (the **average**). The light reading at that point will be mid-range:



In this project your program seeks the edge of the line -- ***This is how you follow lines in Project 10.***

In this project you use the robot to finely **calibrate** your Light sensor. This involves taking light readings, performing calculations, saving the values and making decisions based on the results. You will also learn how to display information on the panel of the NXT -- useful when debugging a program.

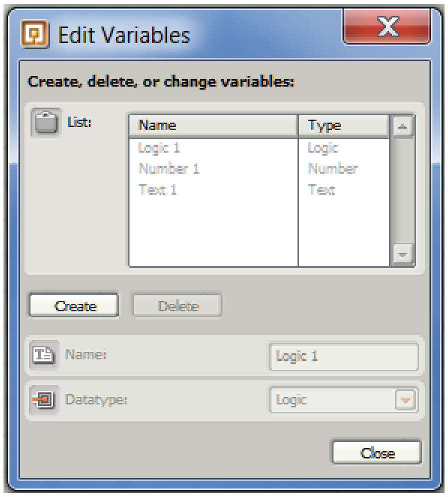
> "**Variable**" -- This is how your program stores information, such as light values and other numbers. Each Variable has a name and a type (number, text, logic). Your program can **read from** and **write to** variables.

In this project you use four variables:

1. **Min –** the light value of the darkest area discovered
2. **Max –** the light value of the lightest area discovered
3. **Avg –** the light value that represents ½ way between Min and Max (the average)
4. **Light –** the most recently read light value

Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil ☑.

□ **1.** Create a new program. Click on Edit/Define Variables in the menu bar. This window appears:



Create, delete, or change variables:

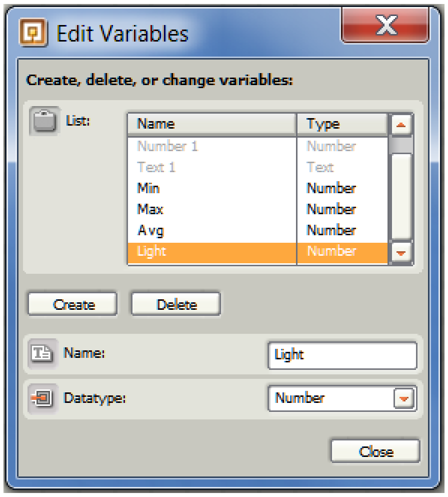
List:

Name	Type
Logic 1	Logic
Number 1	Number
Text 1	Text

Name: Logic 1

Datatype: Logic

□ **2.** Click on Create, in the Name field enter "Min", select a data type of "Number". Repeat this to create three more variables named "Max", "Avg" and "Light".



Create, delete, or change variables:

List:

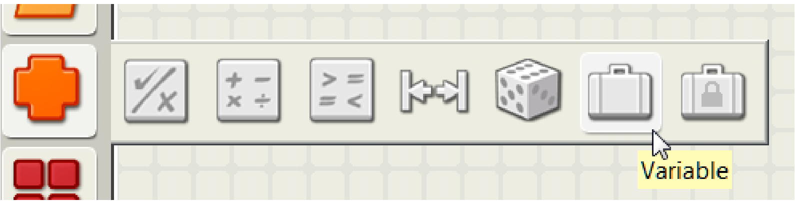
Name	Type
Number 1	Number
Text 1	Text
Min	Number
Max	Number
Avg	Number
Light	Number

Name: Light

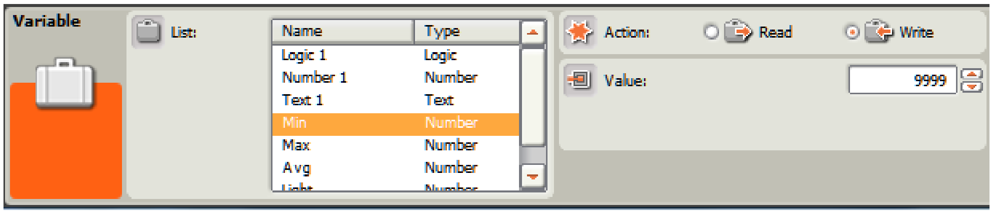
Datatype: Number

When satisfied, click Close.

□ **3.** You will use the Min variable to store the darkest (least) light value that we read during the calibration run. Place a Variable block in your program, you will use this to pre-set its value to a high number that will be changed once your program starts taking actual readings. The Variable block is in the Data flyout menu as shown:



In the configuration panel for the Variable block select Min from the list of variables, click on Write, and enter a value of 9999 (an impossibly high reading):



Variable

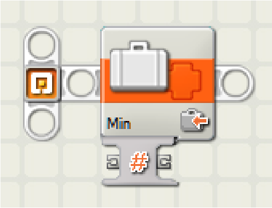
List:

Name	Type
Logic 1	Logic
Number 1	Number
Text 1	Text
Min	Number
Max	Number
Avg	Number
Light	Number

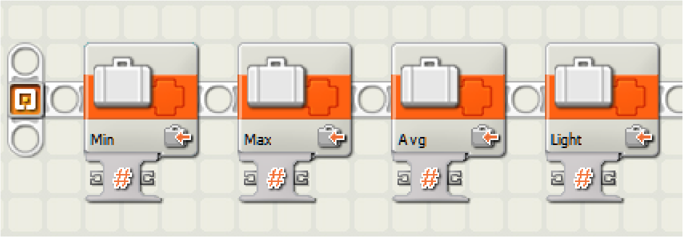
Action: ○ Read ● Write

Value: 9999

Your program should now look like this:



□ **4.** Repeat this, adding Variable blocks to set Max to 0, Avg to 0 and Light to 0. Your program should now look like this:



You will now create the rest of the program that uses the light sensor to calibrate the Min, Max and Avg variables.

The automatic calibration routine you are about to create does the following:

1. Starting with the light sensor pointing at the **darkest** region (pointing directly at the tape), your robot turns in place gently to the right for about ¼ turn. This is far enough for the sensor to move completely off the tape, to point at the white part of the board.

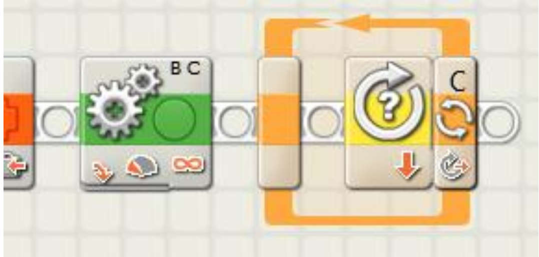
2. While the robot moves it constantly takes light readings and writes them to the Light variable.

3. Each time it takes a reading, it compares Light with the Min and Max values.

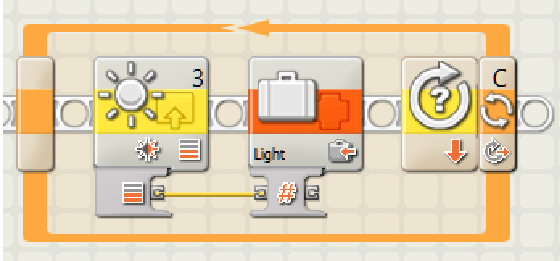
4. If Light is brighter (greater) than the current Max value, the Max value is set to Light.

5. If Light is darker (less) than the current Min value, the Min value is set to Light.

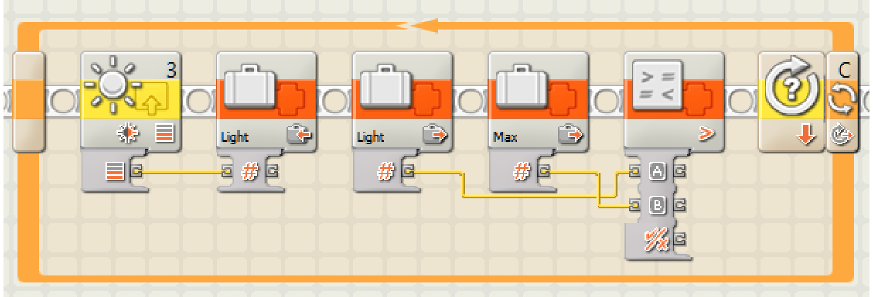
☐ **5.** Add code to your program to rotate the robot slowly to the right about ¼ turn. Use the rotation sensor on the Port C motor to control the amount of robot turn. Download and test the code to see that it turns an appropriate amount:



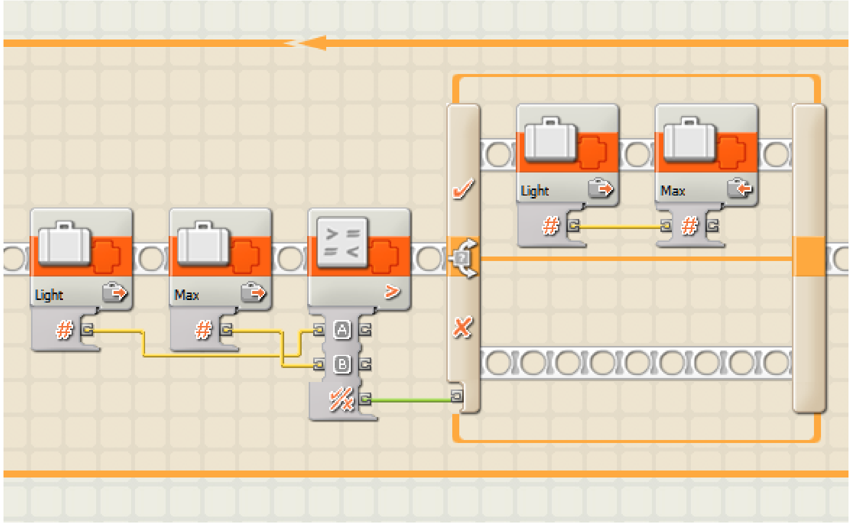
☐ **6.** Insert code in the loop to read from the Light sensor to your Light variable. Be sure to wire from the output of the Light sensor block to the input of your Variable block:



☐ **7.** Insert code to compare Light with Max. Wire it as shown, use the Compare block to see if Light is greater than Max (A > B):

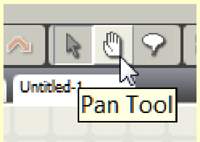


☐ **8.** Insert a Switch block that is controlled by the logic output of the Compare block, where the True case puts the value of Light into Max (think about why!). The False case does nothing:

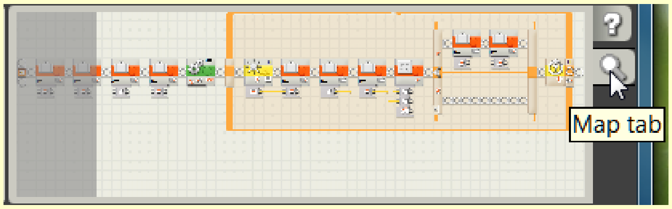


Don't forget to wire the output of the Compare block with the input of the Switch block!

As your programs get larger, you'll need to "scroll" around to see different parts. This can be done by selecting the "Pan" tool in the toolbar and dragging your program:

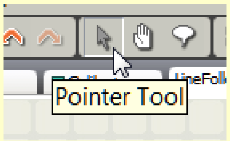


You can also use the "Map tab" in the lower right of the screen and drag the current viewport to the part of the program you want to see:

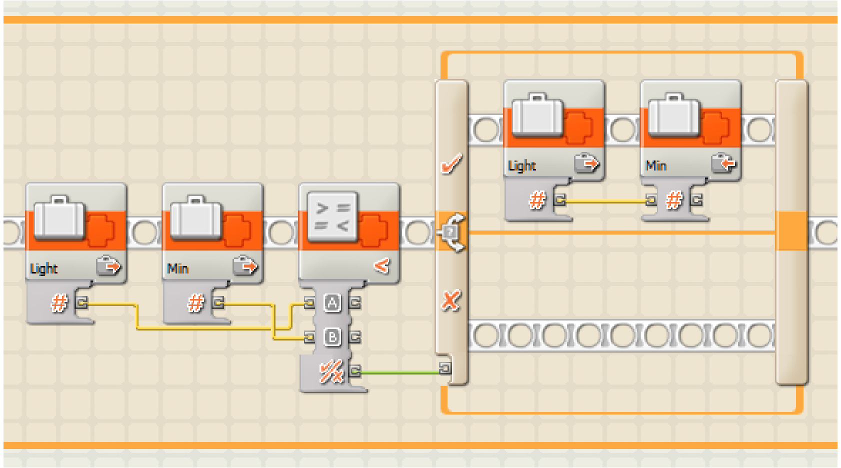


It's a good idea to practice this to become comfortable with moving around in large programs.

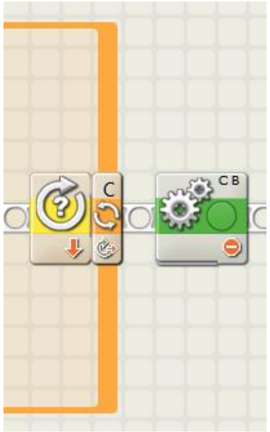
Don't forget to re-enable the Pointer tool when you're done:



□ **9.** Similar to how you update the Max variable, add code to your program to update the Min variable if the Light value is less than the current Min value. The new section of code will look like this:

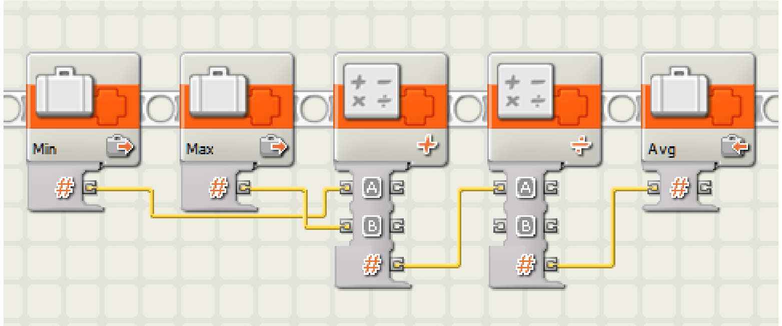


□ **10.** This loop is now complete. Make sure that the motors come to a full stop by adding a Move block set to Stop just after the loop:



□ **11.** Now let's do a calculation to get the Avg (average) of Min and Max. This will be used later as the threshold for our line detection "algorigthm".

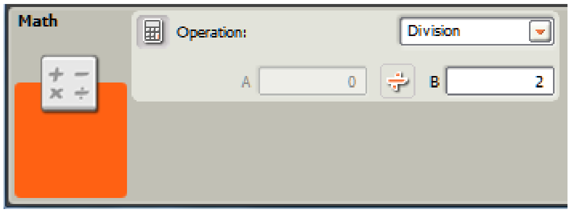
After the loop insert blocks to add Min plus Max, divide the result by two, and place that final value into Avg:



Note: If the term "Average" is new to you, here's what it means: The average of two numbers is a third number that's exactly half-way in between. To calculate an average, add the two numbers and divide the sum by two. That's what you're doing in your program.

The first calculation is pretty simple, it adds inputs A and B (Min and Max) and outputs the sum. Notice the "+" symbol on the block.

Your second calculation is a bit more complex, it divides the A input (the sum of the prior calculation) by the B value, but it looks like there's no B input. If you check out the Configuration panel for this block you see that the B value (2) is typed in directly by you:



Math

Operation: Division

A 0 ÷ B 2

You now have the three values you wanted – Min, Max and Avg. Let's display them!

□ **12.** At the end of your program you will add blocks that display the three variable values on the LCD panel of your NXT brick.

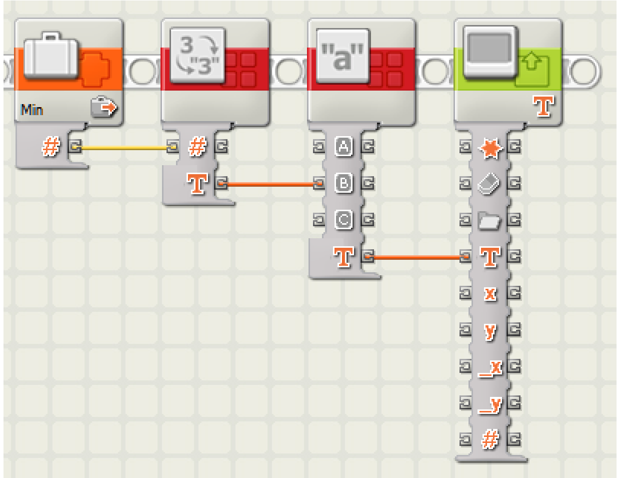
First, let's "build" a text "string" to be displayed. We want it to read "Min =" followed by the current value of the variable.

Insert a Variable block, select Read and Min.

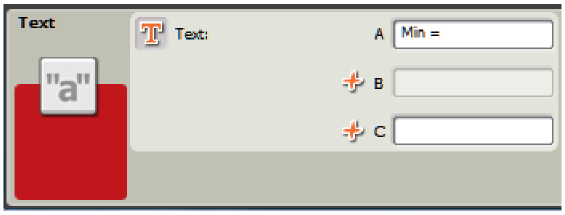
Then insert a "Number to Text" block and a Text block, both are located in the Advanced flyout. You will use the Number to Text block to convert the Min variable to its text equivalent, and the Text block to combine "Min =" with that text in preparation for display.

Add a Display block and drop down the Data hub for the Text block and Display block as shown.

Finally, add wiring to this section of code, the result should look like this:



□ **13.** In the Configuration panel for the Text block, set it as follows:



Text

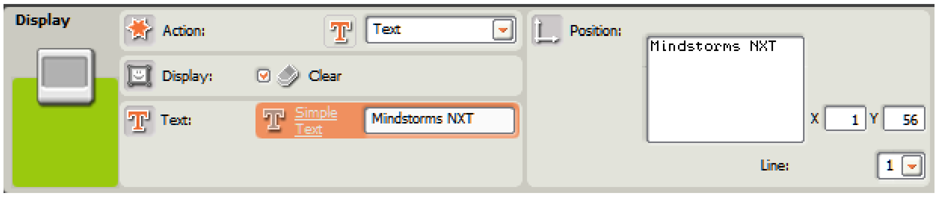
Text: A Min =

B

C

The output of this Text block will be the combination of the A input "Min = " and the B input, which is the current value of the Min variable (converted to text).

□ **14.** In the Configuration panel for the Display block, make these settings:



Display

Action: Text

Display: ☑ Clear

Text: Simple Text Mindstorms NXT

Position: Mindstorms NXT

X 1 Y 56

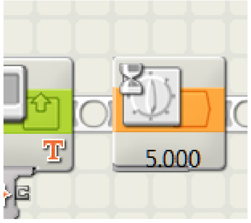
Line: 1

Notice that "Clear" is checked, this will clear the display before showing the data. Also notice that the text is to be displayed with an X position (horizontal) of 1, on Line 1.

□ **15.** Repeat steps 12 to 14 to display similar text, such as "Max = xx" and "Avg = xx", where xx is the current value of each of these variables.

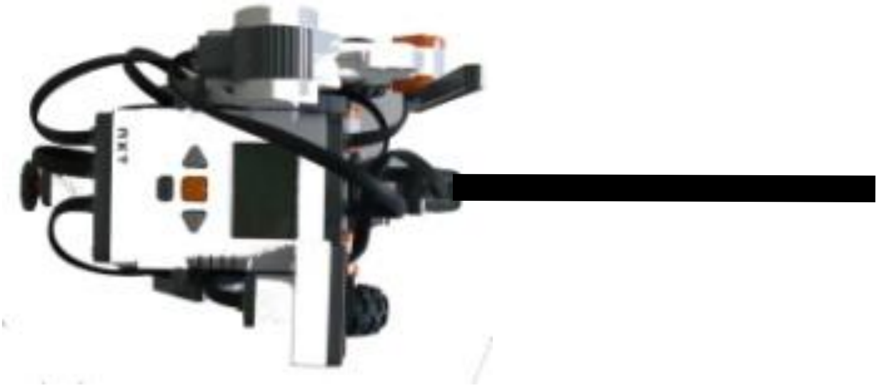
For the two new Display blocks, **DON'T set Clear**, otherwise you will only see the last one. Also set the Line number 2 and 3 respectively so that they won't over-write each other.

□ **16.** Insert a 5-second Wait block to the end of the program, this gives you time to read the display before it is cleared when the program ends. You will delete this wait block later as you add more code to your program:



Save the program under a sensible name.

□ **17.** Download and test the program. Before you run the program, place the robot as shown so that the line sensor is <u>directly</u> above the tape:



Once the program has finished, you have 5 seconds to observe the text on the display of your NXT Brick.

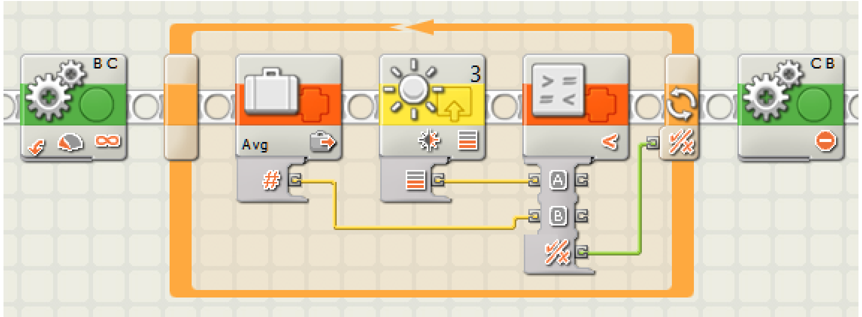
Do the values seem sensible? If not, look carefully at your program, find the problems and fix them. Don't stop until you have correct output!

□ **18.** Save the finished program again.

□ **19.** To make this routine really usable, let's add code to turn the robot back to re-find the edge of the line using the newly-created Avg threshold.

Delete the Wait block from above. *You can always add it back in if you want to see the values.*

Add the following blocks to the end of your program:



Examine this code, what does it do?

1. The initial Move block starts the robot turning slowly to the left (the opposite direction that it was moving). This allows it to start scanning for the edge of the line. What the program is looking for is the moment when the Light sensor is returning just less than the Avg value of light. **This happens when the sensor is directly over the right edge of the line.**

2. Next, within the loop it repeatedly reads from the Light sensor, and compares it with the Avg variable. Only when the Light sensor's value is less than Avg, will the loop stop.

3. After the loop, the motors are both stopped by the final Move block.

□ **20.** Download and test this final version of the program. Fix any problems, and be sure to save as well.

When your program is working properly, the light sensor should be directly over the edge of the line when the robot stops moving.

Let's make this program usable in other programs by converting it into a MyBlock.

□ **21.** Select all of the action blocks in the program. You can do this by holding the Shift key while you click from block to block (or surround a group of blocks). Scroll as necessary.

Note: To select a Loop or Switch and all its contents, you merely have to select the Loop or Switch block, not the individual blocks inside.

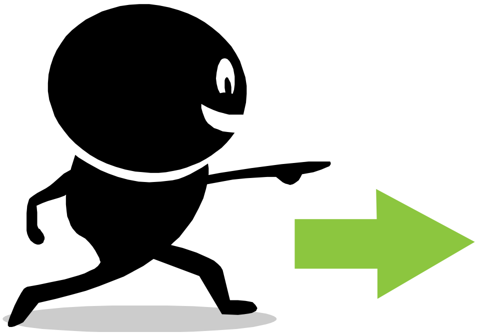
Make sure that all blocks are selected before continuing to the next step. Look for the blue line around each action block (except those within loops and switch statements)!

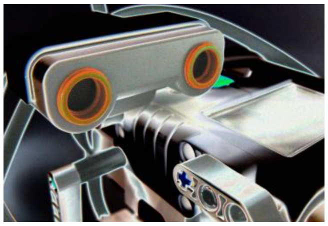
□ **22.** Convert the ENTIRE program into a single MyBlock, name it "Calibrate" and choose an icon you like.

Congratulations, you have now learned many new skills

Now let's move on to Project 10 where you will find out about

Line Following





NXT Programming for Beginners

Project 10: Line Following

Follow the Black Brick Road (sorry Dorothy)

In this project, you will create three programs that can follow the edge of a line as it wanders left and right. There are limits to how well a one-sensor robot can perform, but you may be happily surprised at how well it does, and without a lot of code.

In **Trial 1** of this project you will use the stock calibration of your Light sensor, and simple two-choice steering.

In **Trial 2** you add more steering options, but still use the stock calibration.

In **Trial 3** trial you add the automatic Calibrate MyBlock you created in Project 9 to improve accuracy.

Trial 1 - Very basic

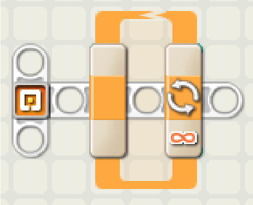
Here is pseudo-code that describes a very simple way to follow the <u>right</u> edge of a line:

In an infinite Loop get a reading from the Light sensor and based on the value do the following:
 a. If it is DARK, that means the robot has moved into the line. Turn right.
 b. If it is LIGHT, that means the robot has moved off the line. Turn left.

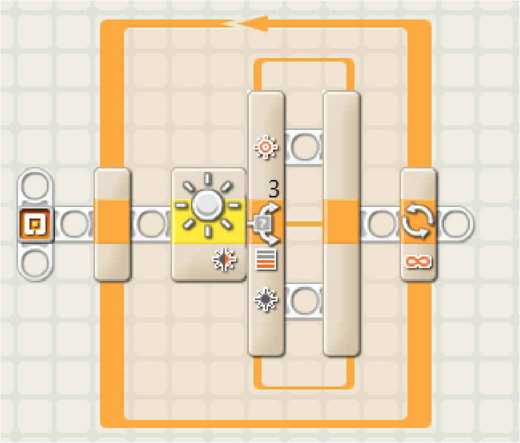
Let's create this simple program:

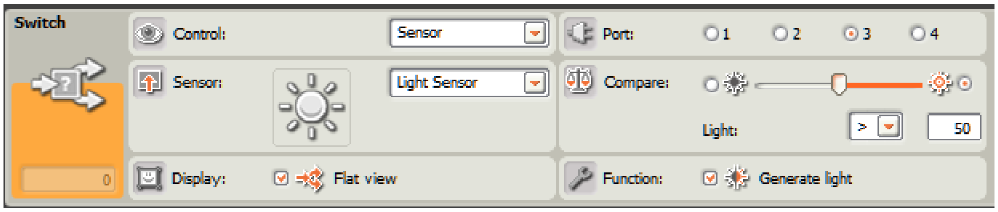
***Please do the following in step-by-step fashion. When you have finished each step please check it off with a pencil* ☑.**

□ **1.** Create a new program. Add an infinite loop as shown:



□ **2.** Inside the loop add a Switch block, set it as shown in the Configuration panel:





Switch

0

Control: Sensor

Sensor: Light Sensor

Display: ☑ Flat view

Port: ○ 1 ○ 2 ● 3 ○ 4

Compare: ○ ●

Light: > 50

Function: ☑ Generate light

Look at the Configuration panel:

What controls the Switch state? ____________________ What Port is it on? _____________

What will the True condition indicate (dark or light)? __________________

What will a False condition indicate (dark or light)? ________________

□ **3.** Place Motor blocks into the True and False path of the Switch block, based on the pseudo code above. Download and test on your track.

If needed, fix the program and re-run. (Hint: Keep your power at 50 or less!)

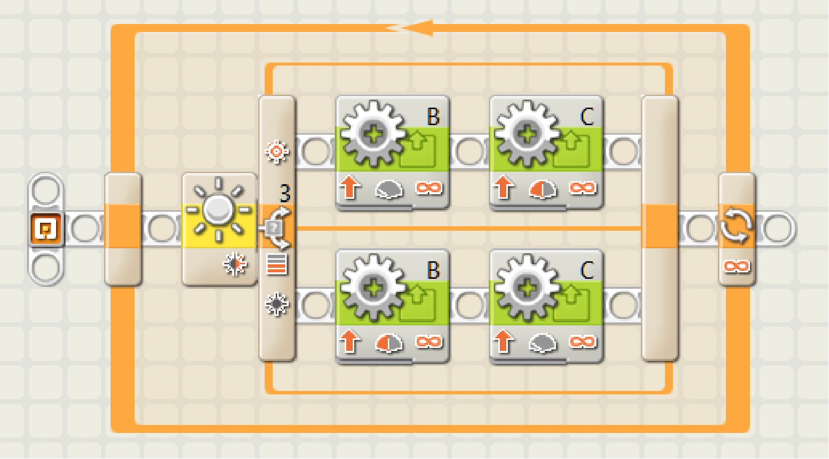
Does your robot stay on the line? __________________________

Is it following the right or left edge of the line? ___________________

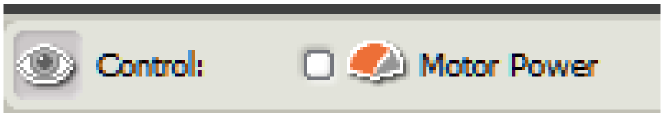
How well does it handle tight curves? ___________________________

Does your robot move smoothly forward, or jerk side to side a lot? ______________________

Here is one possible version of the program for this first trial:



A general rule*: For smoother transitions in Line Following, use Motor blocks with Control: Motor Power **<u>NOT</u>** checked:*


Control: □ Motor Power

Trial 2 – Better

For the second trial, you will use a more flexible "algorithm", one that has more options for motion. Here is the pseudo-code:

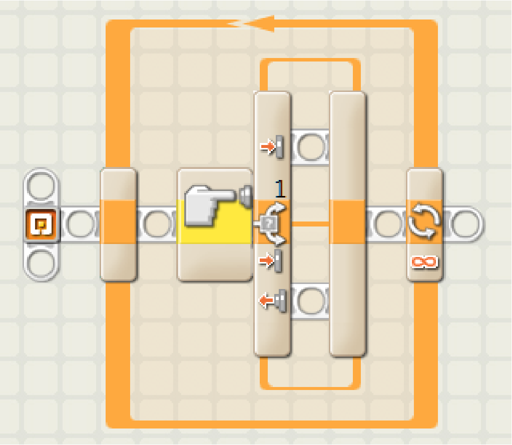
In an infinite Loop get a reading from the Light sensor and based on the value do the following:

a. If it is really dark, that means the robot has moved totally into the line. Turn hard right.
b. If it is somewhat dark, that means the robot has moved a bit into the line. Turn right.
c. If it is close to mid-range, keep going straight
d. If it is somewhat light, that means the robot has moved a bit off the line. Turn left.
e. If it is really light, that means the robot has moved totally off the line. Turn hard left.

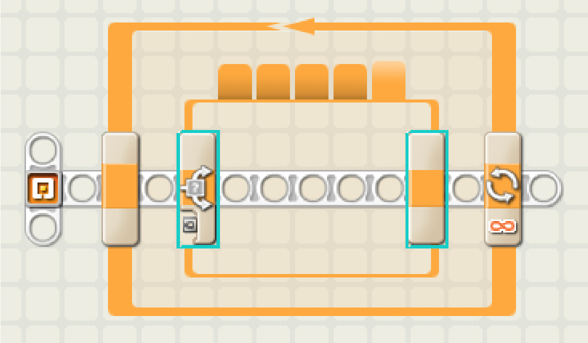
□ **4.** Fill in the following table (the first value is done for you):

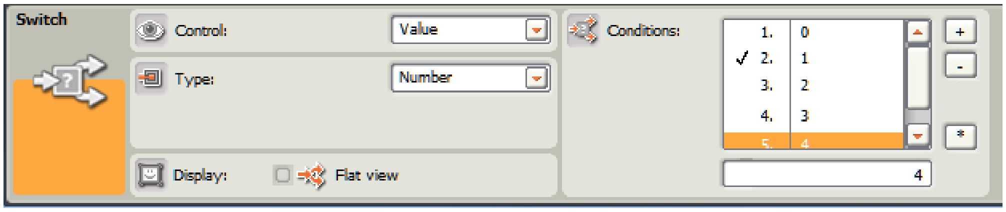
Light Value from Sensor	Action
0-19 (dark)	Turn hard to the right.
20-39	
40-59	
60-79	
80-99 (light)	

□ **5.** Create a new program with an infinite loop. In the loop place a Switch block:



□ **6.** In the Configuration panel for the Switch set Control to Value, Type to Number, deselect Flat view. In the Conditions area click the Plus (+) button three times, to have a total of five (5) Conditions as shown:





Switch

Control: Value

Type: Number

Display: □ Flat view

Conditions:

	1.	0
✓	2.	1
	3.	2
	4.	3
	5.	4

4

What this means is that there are five possible paths in this Switch block, based on the incoming number. If a 0 is the incoming value, it will do the first path, if it's 1, the second path and so on.

The check mark next to the 2. Condition says that this is the "Default" action. This means that if a control value <u>other</u> than the listed conditions (0 through 4) comes in, the Switch block will do the Default action.

You will set the Default action in a later step by clicking on a Condition and pressing the * *button.*

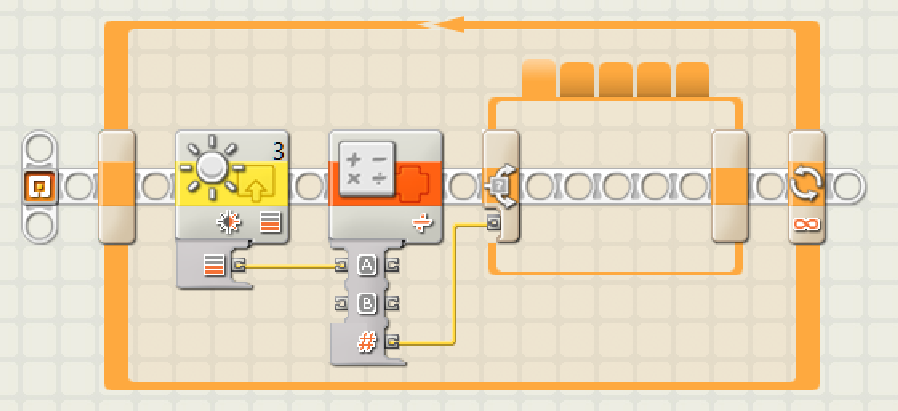
□ **7.** If we have a light value from the Light sensor from 0 to 99, how do we convert this to a simple number from 0 to 4 (five possible values) to use with the switch block? Here is a table that may suggest a good way to do this. Copy your answers from the table in step 4 above to finish it:

Light Value from Sensor	Desired number	Action
0-19 (dark)	0	Turn hard to the right.
20-39	1	
40-59	2	
60-79	3	
80-99 (light)	4	

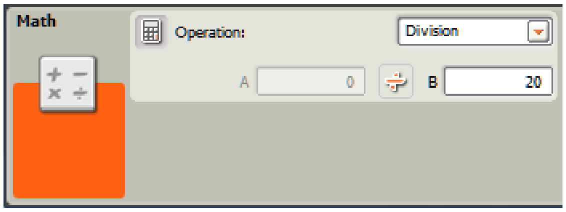
□ **8.** A simple mathematical way to do this is to divide the Light value by 20 (and throw away the remainder).

For example if we have a light value of 65 what will that produce when divided by 20? ________

□ **9.** Let's add a few blocks to our program to retrieve the light value, divide it by 20 and provide the result to the switch statement:



Here is the Configuration panel for the Math block:



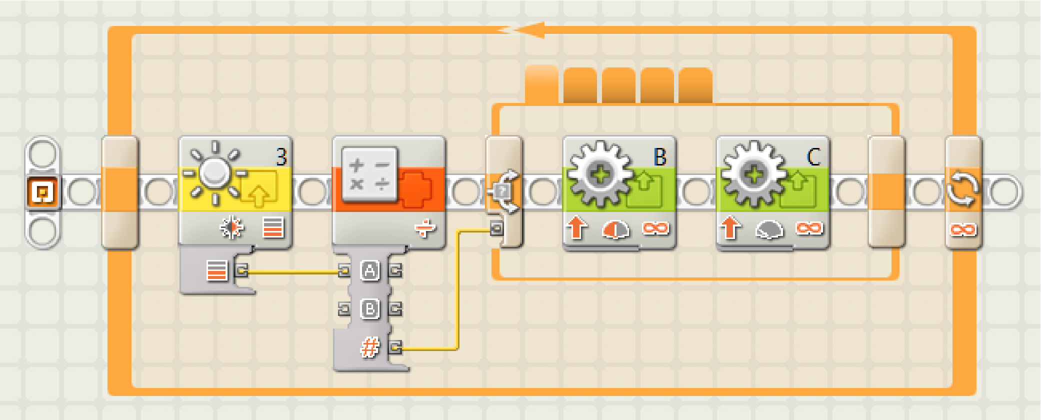
Math

Operation: Division

A 0

B 20

□ **10.** Notice that the Switch block has 5 tabs! There's one for each of the values, 0 at the left, 1 the next and so on through 4. Looking at the table in step 7, the path for the first tab wants code that turns hard to the right (it's over a very dark area).



Notice that the power for the left motor (B) is <u>much</u> greater than the power for the right motor (C).

□ **11.** Using similar logic, fill in code for the remaining four tabs according to the table in step 7. Be sure to keep your power to 35. Hopefully you know to put code to move straight ahead in the path for the middle tab! **(Hint: Set the Default to be tab 5, just in case the light value is greater than 100!)**

□ **12.** Download and test your program, fix if necessary.

□ **13.** How well does this perform, compared with your line follower from the first trial?

Is your robot able to move more smoothly? ______________________________

Does it follow the line, whether sharp or smooth turns? _______________________________

Does it stay on track better when the track turns one way versus the other? _____________________

What happens if the light value is greater than 100? _________________ (remember the Default tab?)

□ **14.** Experiment with your program, change the sharpness of turns and the power for the different Move blocks.

Are you able to get it to navigate more quickly? __________

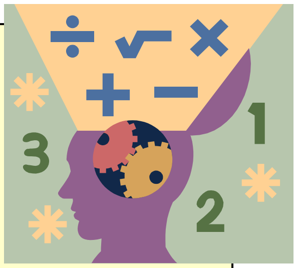
Are you able to get it handle turns more reliably? __________

Are you able to make the motion smoother? ____________

□ **15.** Save your program under a logical name.

Trial 3 – Getting pretty good

For the third and final trial of this project, you will use the MyBlock that you created in Project 9 to calibrate the Light sensor. Instead of relying on pre-measured light values, your robot will now calibrate itself each time the program is run, making it compensate better for variations in room lighting, and the specific light and dark properties of the track.



Trial 3 – The Math

In **Trial 2** you used a simple range of light readings (0 to 100) that assumed that your light sensor had been calibrated. This yields *ok* results, but doesn't adapt to changing conditions.

In **Trial 3** we use the actual measured range (Min, Max) to control our FlexBot's motion. Let's take a look at how we can do this without major re-writing of our code.

Let's say, for example, the values we gather during the Calibrate MyBlock are:

Min = 14
Max = 89

To support the code we already have, we need a way to break this range into 5 equally spaced ranges, much like we divided the arbitrary range 0 to 100 in Trial 2. How can we do this?

To calculate the range from Min to Max we subtract Min from Max. In our example, this is:

89 – 14 = 75

If we now divide 75 by 5 (the number of ranges), we get:

75 / 5 = 15 Save this number into a variable named "Step"

When we're ready to take light readings and create a number for the Switch block, we do the following:

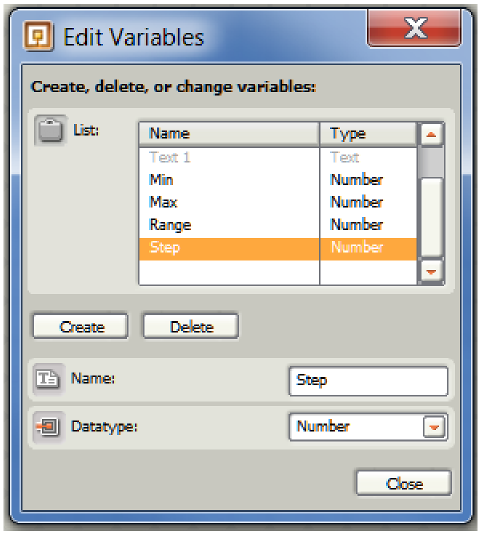
1. Take a light reading
2. Subtract Min from that reading, this makes it zero-based, suitable for division.
3. Divide the result by 15 (our Step value from above). This number controls the Switch block.

For the more mathematically oriented, here are the equations:

Step = (Max – Min) / 5

Switch control value = (Light – Min) / Step

□ **16.** Starting with the program from Trial 2, add number-type variables named Min and Max. These are the same variables you used in the Calibrate MyBlock. This will allow your program to access these values when necessary. Before closing the window, add two more number-type variable named Range and Step. The Edit Variables window should look like this:



Edit Variables

Create, delete, or change variables:

List:

Name	Type
Text 1	Text
Min	Number
Max	Number
Range	Number
Step	Number

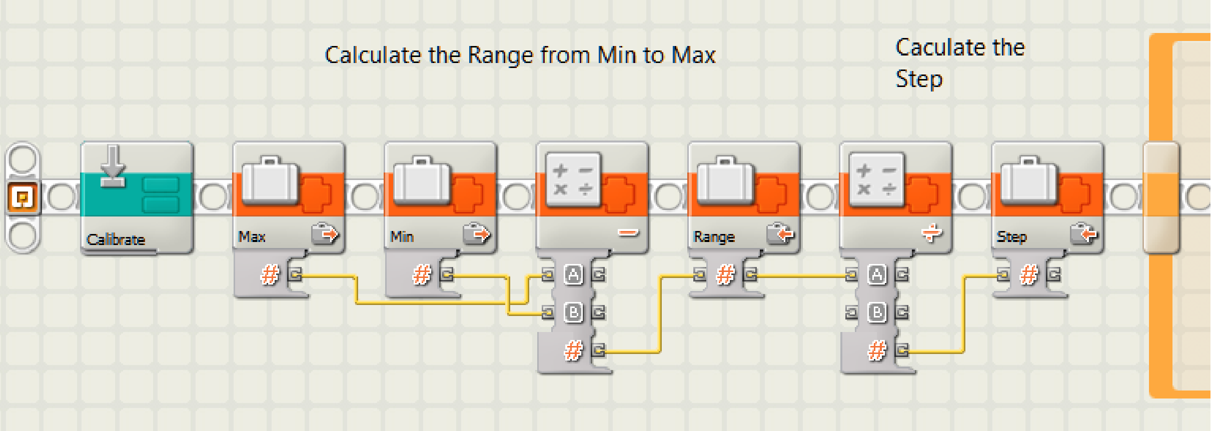
Name: Step

Datatype: Number

□ **17.** Insert a Calibrate MyBlock at the beginning of your program, before the main Loop.

□ **18.** Just after the Calibrate MyBlock, add code to calculate the Range and Step values as per the description above. Don't forget the wiring!

Notice that comments have been added above the blocks to describe what's happening. Do this in your own program:



Calculate the Range from Min to Max

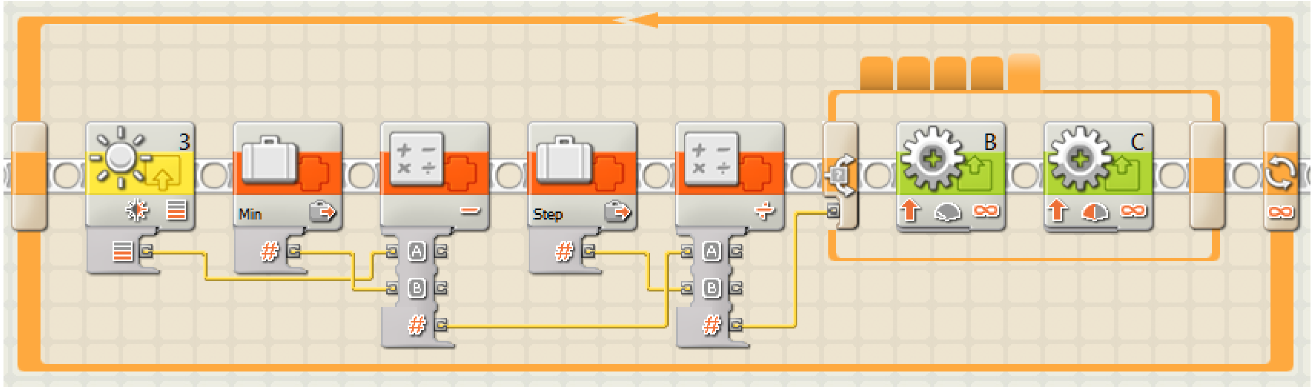
Caculate the Step

Make sure that the second math block is set to Divide, and the B value is 5 (why 5? Think about it?).

□ **19.** Inside your loop you need to add a bit more calculation. First you need to subtract Min from your light reading to make it zero-based, and then divide the result by Step to produce the control value for the Switch block.

Go ahead and do this on your own, take your time to understand the logic (don't peek at the picture below just yet).

Refer to this example only **AFTER** you try on your own:



□ **20.** Test and fix your program, if necessary add Display blocks to show what your program is doing.

□ **21.** Do you find that the performance has improved?

Depending on how well your light sensor was pre-calibrated, you might not see any real difference. Where this version really "shines" is if the lighting conditions have changed since you did your calibration.

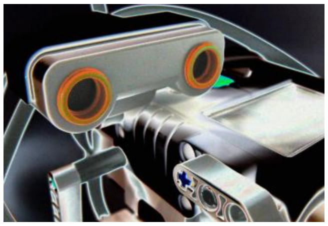
□ **22.** Save your program under a useful name.

Note: There are even better ways to follow lines, but they generally involve more math. For examples of good line following methods for the NXT visit this web site:

http://www.nxtprograms.com/line_follower/steps.html

Congratulations you now have three different nicely working Line Follower routines.

You're ready for Project 11, the **Final Challenge**.



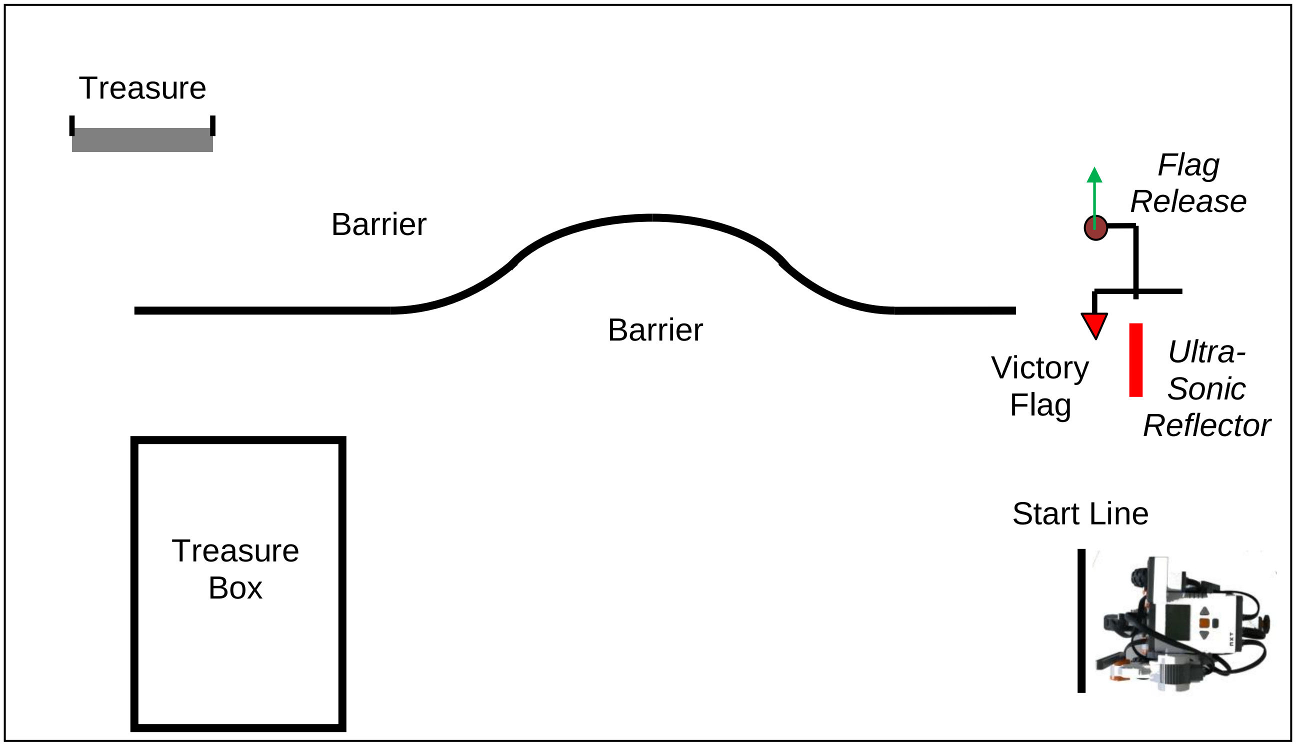
NXT Programming for Beginners

Project 11: The Final Challenge

Putting it all together!

In this final project you program your FlexBot to perform a series of challenges, each requiring a variety of skills. This is YOUR chance to show what you can do -- there will be very little explanation or guidance!

The Challenge Track looks like this:



The challenges:

1. Starting behind the Start Line, travel to the Treasure and retrieve it.
2. Bring the Treasure to the Treasure Box and deposit it. No part of the Treasure should extend beyond the outside edge of the Box.
3. Travel to the Victory Flag, trigger the Flag Release (push the release to the Left). **You MUST travel between the Barriers without touching them.**
4. Do a victory dance!

Be sure to study the actual track and challenges.

Points:

Retrieve the Treasure:	5 points
Place the Treasure fully in the Box	10 points
Raise the Victory Flag	20 points

Penalties:

Robot touches a Barrier	-5 points per Barrier
Treasure extends outside Box	- 3 points
Robot must be rescued	-10 points

If the robot goes off the edge of the track, the run ends at that point.

Good Luck!