To GSOC Troop Leaders:

Thank you for checking out the Robotics Badge “Program in a Box”!

On the following pages, we break down the badge requirements for Daisy, Brownie, Junior and Cadette levels that can be completed using this Program In a Box. Note that this “Program in a Box” will not complete all of the badge requirements, so pay special attention to where you will need to supplement this program to complete badges.

In addition to the level badge requirements, this document includes a discussion of ***What Is a Robot?***, a discussion of what sensors, and how sensors are critical to the very definition of “robot”. In addition, a detailed ***Parts of a Robot*** review is included based upon the Gigglebot robot included within the program box. There is also a discussion of ***What does a Robotics Engineer do?*** as well as an associated brainstorming activity for the girls to consider a useful robot application.

This discussion is immediately followed by ***How can we make our Gigglebot robot do something?***, which is very important to follow as it explains and supports testing the basic “flash and run” steps that are required to program the Gigglebot and thereby complete the subsequent ***Lessons***.

In other words, it is important to follow the documentation leading up to the lessons; you need to read this documentation to understand how to perform the subsequent Lessons, and this part of the documentation ties to many of the Girl Scout badge requirements (e.g. discussing ***Parts of a Robot***).

Further, it is important that you complete the Lessons in order, or at minimum complete Lesson 1 before moving forward onto any of the other Lesson sections within this document.

Finally, the underside of the Program in a Box lid contains an inventory of the contents of the box; kindly be sure that you return all parts into the boxes prior to returning the boxes to council.

Thank you,
Volunteer STEM Patrol

2018 DAISY ROBOTICS BADGE REQUIREMENTS

Key:
Y = Box program fulfills badge requirement
P = Box PARTIALLY fulfills badge requirement
N = Leader is required to follow the program guide to fulfill requirement

Daisy Badge #1 What Robots Do
Y - Learn about Robots
Y - Find out what Robots will do (***Brainstorming*** section within)
P - Team up Design your own robot (as a complement to the ***Brainstorming*** section, have Daisies draw their own robot design)

Daisy Badge #2: How Robots Move
Y - Learn about the parts of a robot (***Parts of a Robot*** within)
Y - Find out how robots move (complete at least ***Lesson #1*** within)
Y - Make a robot move (complete at least ***Lesson #1*** within)

Daisy Badge #3: Design a Robot
This badge is not covered within the materials here, but it would be a good follow up meeting to a meeting using this “Program in a Box”

2018 BROWNIE ROBOTICS BADGE REQUIREMENTS

Brownie’s Badge #1: Programming Robots
P - Build a Simple Machine (leader should add discussion of simple machines)
Y - Test your robot senses (discussion of sensors within)
Y - Learn about programming (complete any of the ***Lessons*** within)
Y - Try a simple program (complete any of the ***Lessons*** within)
Y - Code a robot (complete any of the ***Lessons*** within)

Brownie’s Badge #2: Designing Robots
N - Explore how robots imitate nature (leader needs to lead discussion on biomimicry)
Y - Learn about the parts of the robot (within, ***Parts of a Robot***)
Y - Plan your robot (within, ***What Does a Robotics Engineer Do*** and ***Brainstorming***)
P - Create a prototype (suggest creating a paper step by step instructions of any of the ***Lessons*** within)
N - Get feedback on your prototype (follow Brownie Badge Guide)

Browie’s Badge #3: Showcasing Robots
N - Create a presentation to share how you designed your robot (could potentially take pictures or video of one of the ***Lessons*** within as the basis of a presentation)
N - Tell others how you designed your robot (in a nutshell, give the presentation you created in the prior step to another troop, at school, or to your family)
Y - Learn about Robotics Competitions (covered within the Appendix of this document)
Y - Learn about Robotics Teams (covered within the Appendix of this document)
N - See Robots in Action (tour suggestions covered within the Appendix of this document)

2018 JUNIOR ROBOTICS BADGE REQUIREMENTS

Key:
Y = Box program fulfills badge requirement
P = Box PARTIALLY fulfills badge requirement
N = Leader is required to follow the program guide to fulfill requirement

Junior Badge #1: Programming Robots
Y - Learn about how robots work (cover the initial material and any ***Lesson***)
Y - Discover the robot brain (cover the initial material and any ***Lesson***)
Y - Learn about programming (cover the initial material and any ***Lesson***)
Y -Try a simple program (cover the initial material and any ***Lesson***)
Y - Code a Robot (cover the initial material and any ***Lesson***)

Junior Badge #2: Designing Robots
P - Discover the Future of Robots (within, ***What Does a Robotics Engineer Do*** and ***Brainstorming***) Leader should supplement with a discussion of biomimicry and artificial intelligence.
P - Determine your Robots Expertise (Leader: brainstorming session is required, could be based on one of the ***Lessons*** within)
P - Plan your Robot (Leader: brainstorming session is required, could be based on one of the Lessons within)
P - Create a prototype (suggest creating a paper step by step instructions of any of the Lessons within)
N - Get feedback on your prototype (follow the Junior Badge Guide)

Junior Badge #3: Showcasing Robots
N - Create a presentation to share how you designed your robot (could potentially take pictures or video of one of the Lessons within as the basis of a presentation)
N - Tell others how you designed your robot (in a nutshell, give the presentation you created in the prior step to another troop, at school, or to your family)
Y - Learn about Robotics Competitions (covered within the Appendix of this document)
Y - Learn about Robotics Teams (covered within the Appendix of this document)
N - See Robots in Action (tour suggestions covered within the Appendix of this document)

2018 CADETTE ROBOTICS BADGE REQUIREMENTS

Key:
Y = Box program fulfills badge requirement
P = Box PARTIALLY fulfills badge requirement
N = Leader is required to follow the program guide to fulfill requirement

Cadette Badge #1: Programming Robots
Y - Learn about how robots work (cover the initial material and any ***Lesson***)
N - Build a robot part: simple sensors (can be accomplished with a laptop, the micro:bit included within this kit, and by using this instructional link: https://microbit.org/guide/temperature/)
N - Make a box robot with sensors (can be accomplished using the instruction link above + brainstorming on robot designs that would utilize this temperature sensor)
Y - Learn about programming (cover the initial material and any ***Lesson***)
Y - Write a program for a robot (cover the initial material and any ***Lesson***)

Cadette Badge #2: Designing Robots

Y - Pick a challenge (within, ***What Does a Robotics Engineer Do*** and ***Brainstorming***)
Challenge girls to team up into small teams and present their brainstormed robot concepts!
Y - Explore possible solutions (within, have the girls describe their brainstormed robot within the context of the points listed within the ***What Does a Robotics Engineer Do*** section)
P - Plan your prototype (extend the points above using a hand drawn paper design)
P - Build a prototype (provide cardboard, Lego or other materials to build some simulated or working component of the robot prototype)
N - Get feedback on your prototype (follow the Cadette Badge Guide)

Cadette Badge #3: Showcasing Robots

Y - Learn about robotic events and organizations (covered within the Appendix of this document)
N - Create a presentation about your robot (could potentially take pictures or video of one of the Lessons within as the basis of a presentation)
N - Present your robot pitch to others for feedback (in a nutshell, give the presentation you created in the prior step to another troop, at school, or to your family)
Y - Learn about robotics opportunities for teens (covered within the Appendix of this document)
N - See robots makers and robots in action (tour suggestions covered within the Appendix of this document)

What is a Robot?

"A robot is a self-running machine capable of sensing its environment, carrying out computations or commands to make decisions, and performing actions in the real world."

Self-running machine: A robot doesn't necessarily need a human operator controlling the robot at all times, the robot can operate independently either all or part of the time.

Sensing its environment: An import distinction of robots from other machines; a robot contains sensors, usually many different types of sensors, that the robot's artificial brain can use to detect the robot's environment and perform resulting actions. Humans use our vision, hearing, smell and touch senses; robots use electrical and mechanical sensors in a similar manner to our human senses.

The coding (programs) installed on the robot will use the data values returned by the sensors to help the robot make decisions, like how or where to move, or what to do when the robot encounters an obstacle.

Here are some examples of **sensors** that could be found on a robot:

- **Light sensors:** detect if there is light, how much light and what color the light is.
- **Reflective light sensors:** detect how much light is reflecting from an object, normally the surface the robot is operating on. Also can determine the light color.
- **Touch sensors:** detect if the robot is touching something at the moment (or not), similar in function to an insect's feelers.
- **Ultrasonic sensor:** Determines how near or far the robot is to an object, just like a bat senses objects within its flight path.
- **Radio transmitter / receiver:** A robot may send or receive a radio signal, for example from the robot's charging base, so the robot will know where to find it. Robots can communicate with each other using radio signals as well.
- **Motor sensors:** Determine how many degrees (360 degrees = 1 turn) the motors on the robot have turned. This will allow the robot to accurately determine how far it has traveled or how high it has lifted or gripping something.
- **Gyro sensor:** Determines the side to side or up and down movement of the robot and can be used to have the robot make very precise turns.
- **Strain sensors:** Used to determine "how hard" a robot is gripping or pushing against something.
- **Cameras:** Used to record video, sometimes for humans to view in real time, or to be recorded for viewing later. A robot can also use a camera to help it identify where it is within its environment, or to identify items of interest it its environment, to support navigation or item identification.
- **Microphones**: Record sounds that a human can listen to in real time or recorded for listening later. Microphones can also be used to listen for voice commands.
- **LIDAR:** A form of radar using a laser that helps a robot understand its environment, normally used to help a robot move around and avoid obstacles.

What does a Robotics Engineer do?

A robotics engineer is a really fun career! If you like puzzles and finding creative ways to solve problems then you may want to become a Robotics Engineer!

A robotics engineer might design a robot for a specific task or need, for example, for exploring the inside of a pipeline, searching for buried items in the sand on a beach (or on the moon!), or for use in a search and rescue mission.

Based on the specific task requirements, the robot engineer will need to:

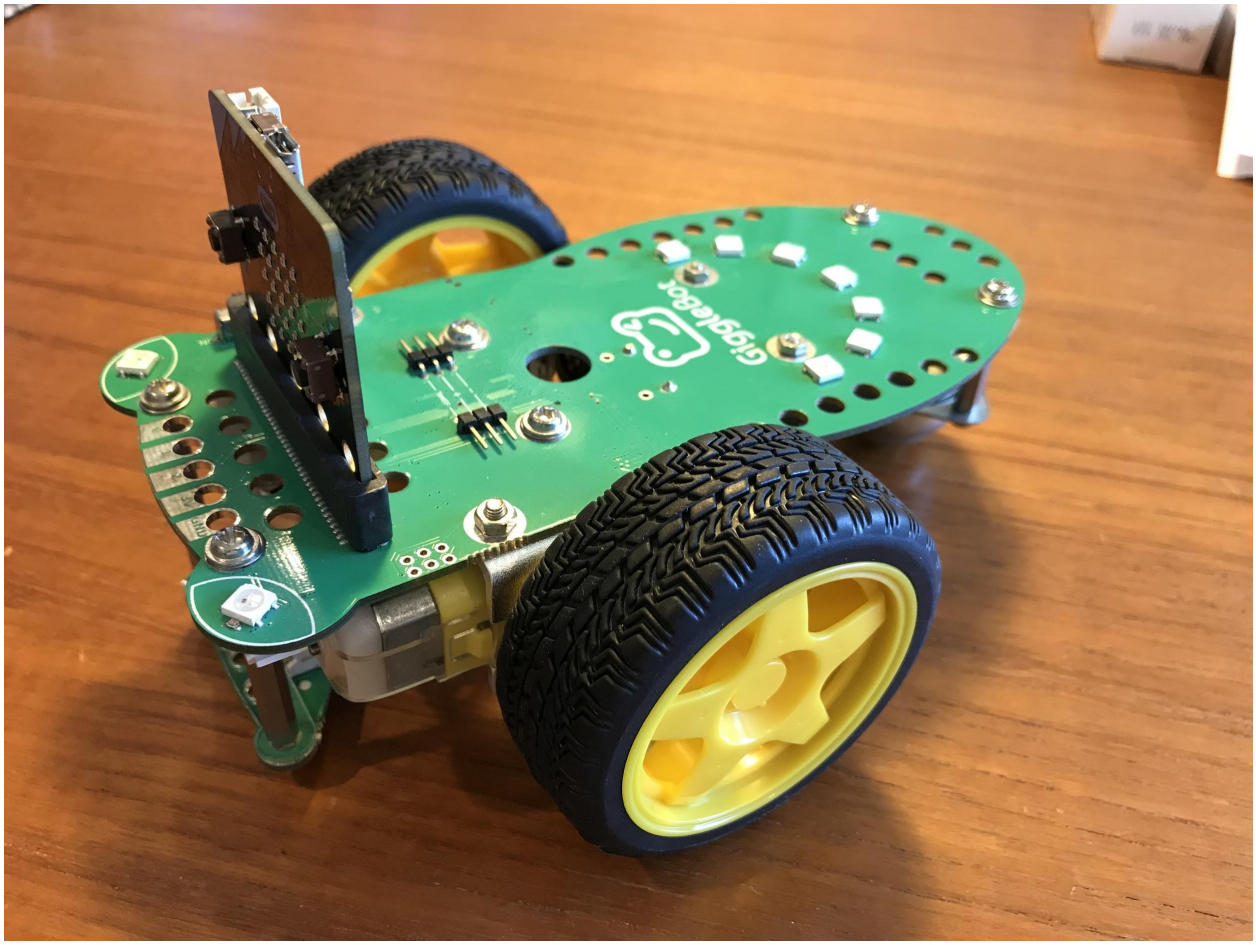
- Determine how the robot needs to move (does the robot need wheels, tracks, legs, arms, have the ability to fly, etc.)
- Determine what sensors the robot will need to perform all the required task(s).
- Determine what computer / microprocessor brain and energy source the robot will need.
- Design and build the physical structure of the robot.
- Program the robot.
- Test, redesign, reprogram and test again!

Brainstorming - take time out to think of some cool robot applications!

How about a robot that will roam around your house and squirt some air freshener when it enters into a room? Maybe a robot that tows a rake behind it and creates beautiful sand art on a beach. What sensors would you need to make a fun robot pet? ***What other cool robotic applications can you think of?***

Parts of a Robot

Now that we know what a robot is, let's look at our teaching robot, which is called a **Gigglebot**.



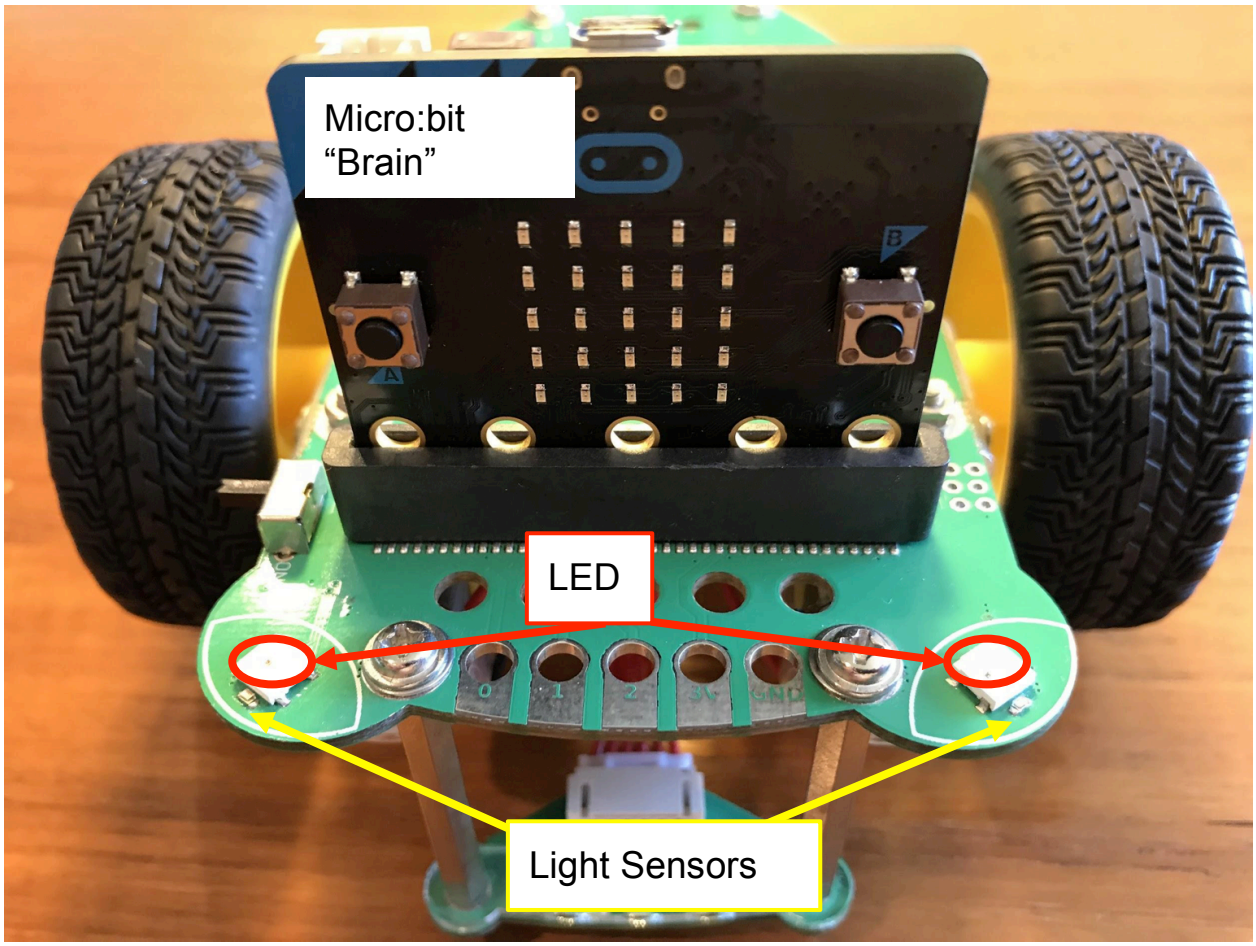
The Gigglebot robot, with micro:bit microprocessor "brain" attached on top.

If you would prefer a video overview, the Dexter Gigglebot video overview is here: https://youtu.be/lwgK3IE4jeA

Our Gigglebot has many of the same components of any robot, which we list below. See if you can identify the parts of the Gigglebot listed below!

- Energy source - (3 AA batteries)
- Two motors
- Two driving wheels/tires (driven by the motors)
- Rear caster wheel (to make turning easy)
- Microprocessor "brain" (like a computer, explained below)
- Sensors
  - Two top mounted light sensors
  - Two bottom mounted reflective light sensors

Here are some pictures to help you identify the parts of the **Gigglebot** !

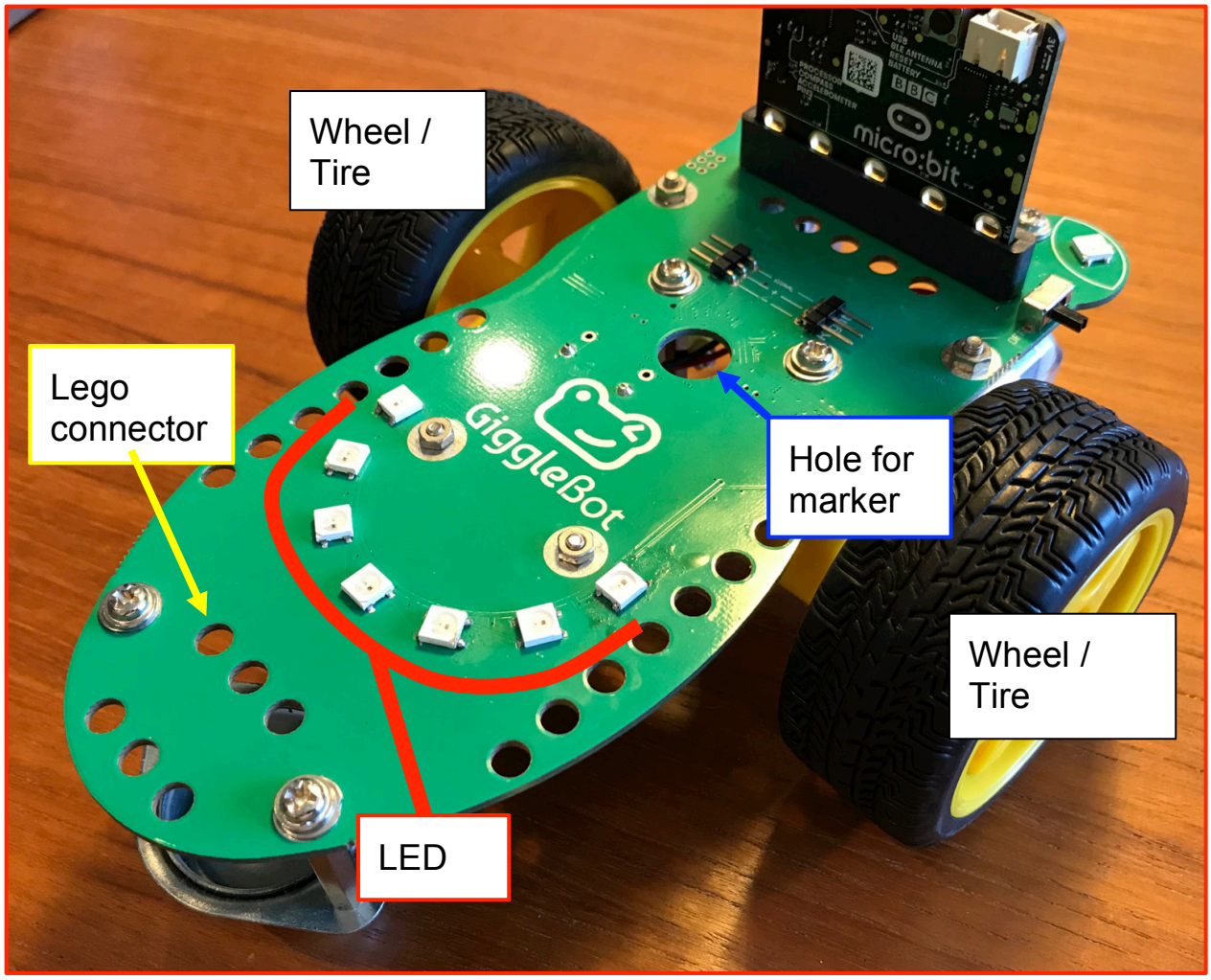


The LED eyes shown above will light RED when the batteries are low. We can also light these two LEDs with our code. The tiny light sensors just in front of the eyes detect light, and we can use these sensors to make the Gigglebot follow a flashlight or a bright light.

The **micro:bit** is a microprocessor used for teaching maker and coding skills. The micro:bit can be used independently of the Gigglebot (see http://www.microbit.org for project ideas) or as the Gigglebot’s artificial “brain” by attaching the micro:bit onto the Gigglebot as shown.

A microprocessor is not a computer; in fact you need a separate computer to program (code) a microprocessor. The microprocessor is connected to the computer, typically with a USB cable, and code written on the separate computer is then downloaded from the computer onto the microprocessor. ***This process is called “flashing” the microprocessor.***

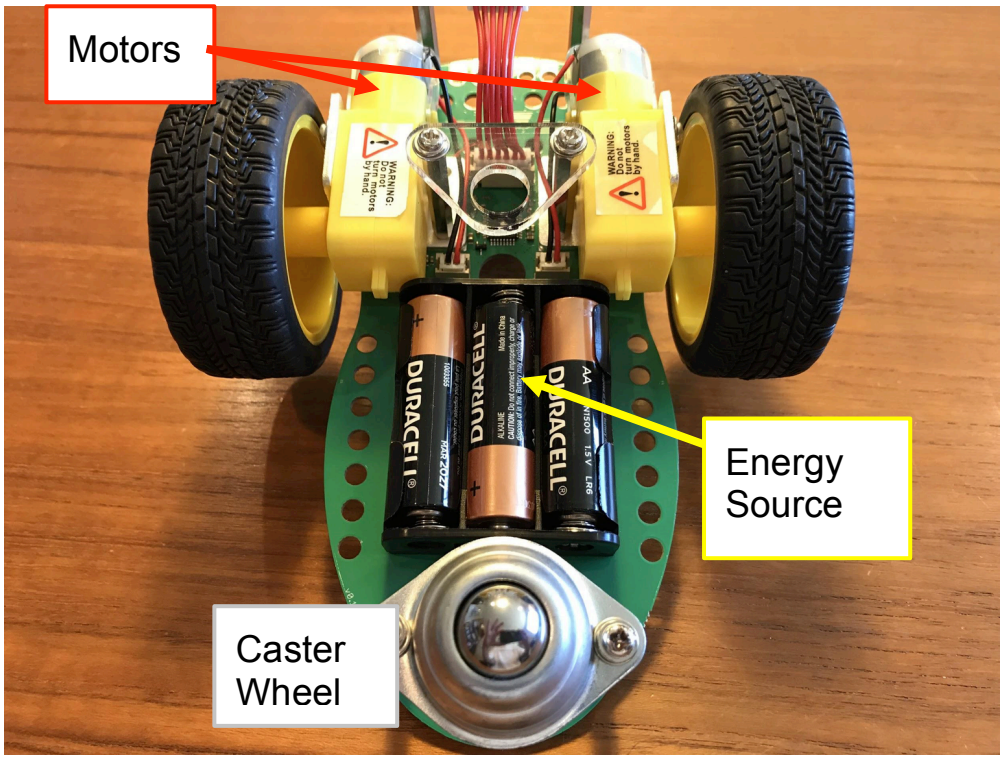
When the microprocessor is disconnected from the computer and powered on, the code begins to execute or “run”. We will be programming our micro:bit with a laptop computer (not supplied within the GSOC Program in a Box) using the Microsoft MakeCode software.



Our Gigglebot has a “hole” through the center of the green chassis platform which accepts a marking pen, as well as smaller holes arranged around the green chassis platform which are sized to accept Lego blocks. You could use these holes to build you own creative Lego “robot body” right on top of the green chassis!

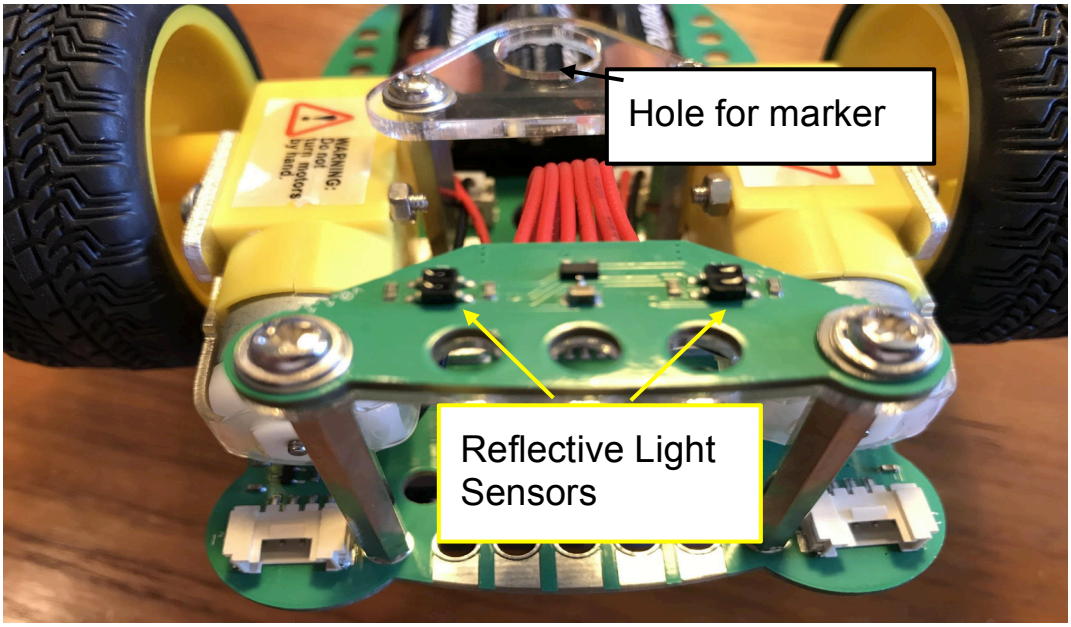
We can place a marker in the “hole for marker” shown above, and have the marker trace the movement of the robot on a piece of paper. What a fun way to make robot art!

We can light up the LED smile with our code to help us know when parts of our code are executing.



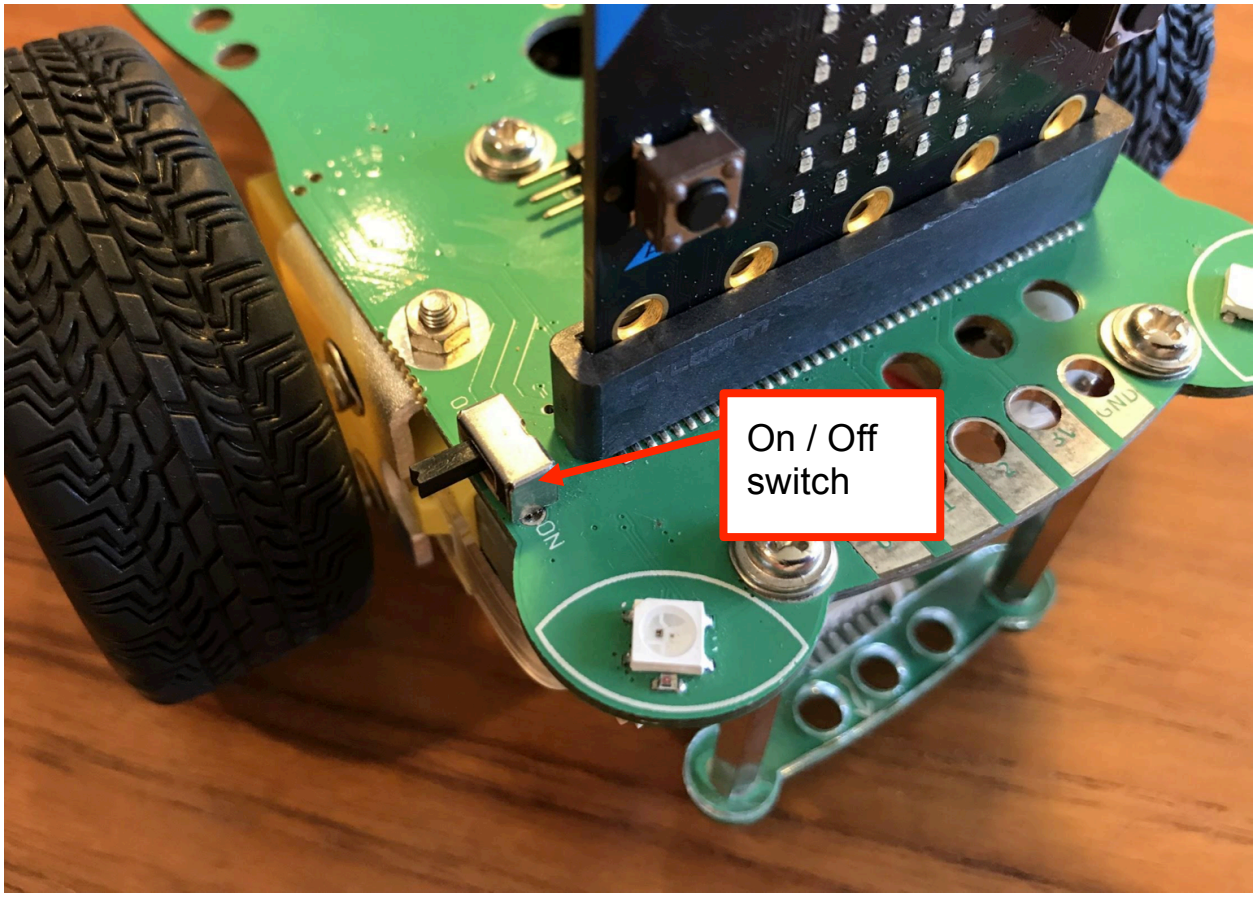
Every robot needs an energy source. Our Gigglebot uses 3 AA batteries as its energy source which powers the micro:bit, the two motors, the LED lights and the sensors of the Gigglebot. Sensors require power in order to read their data or the information the sensors return.

The caster wheel, found at the very back bottom of the Gigglebot, makes it easy for the robot to make tight turns in any direction.



On the front underside of the Gigglebot we can see tiny black reflective light sensors which are positioned close to the tabletop. The reflective light sensors detect how much light is reflected back from the tabletop. We can use these sensors to determine if our Gigglebot is overtop of a

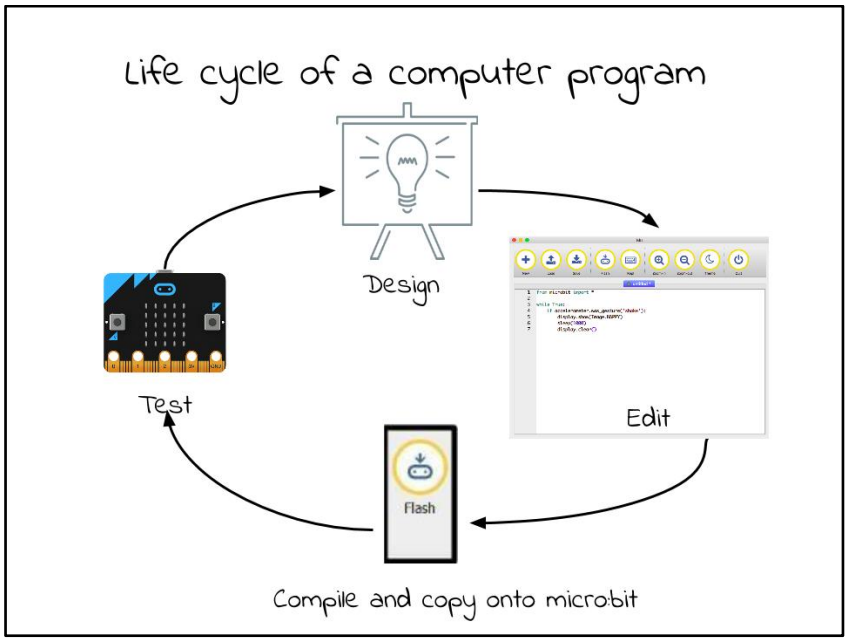
glossy black line on the tabletop or “mat”. We can then write code that will use the sensors to follow the glossy black line.



The Gigglebot’s on/off switch is on the front right side of the Gigglebot as shown here. We use this switch to turn the Gigglebot on (or off if it starts to run away from us!). ***IMPORTANT: Always switch the Gigglebot OFF before replacing the micro:bit card, and place the Gigglebot down in a safe place (like an open floor) before turning it back ON.***

How can we make our Gigglebot robot do something?

The life cycle of programming the Gigglebot:



The Gigglebot uses a micro:bit microprocessor as its artificial “brain”. Using software installed on a separate laptop or tablet, we create programs (i.e. code) which are then copied or “flashed” onto the Micro:bit.

In this guide we will use **Microsoft Makecode** (https://makecode.microbit.org) for creating our programs. There are other (more advanced) coding options which are included in the appendix of this document.

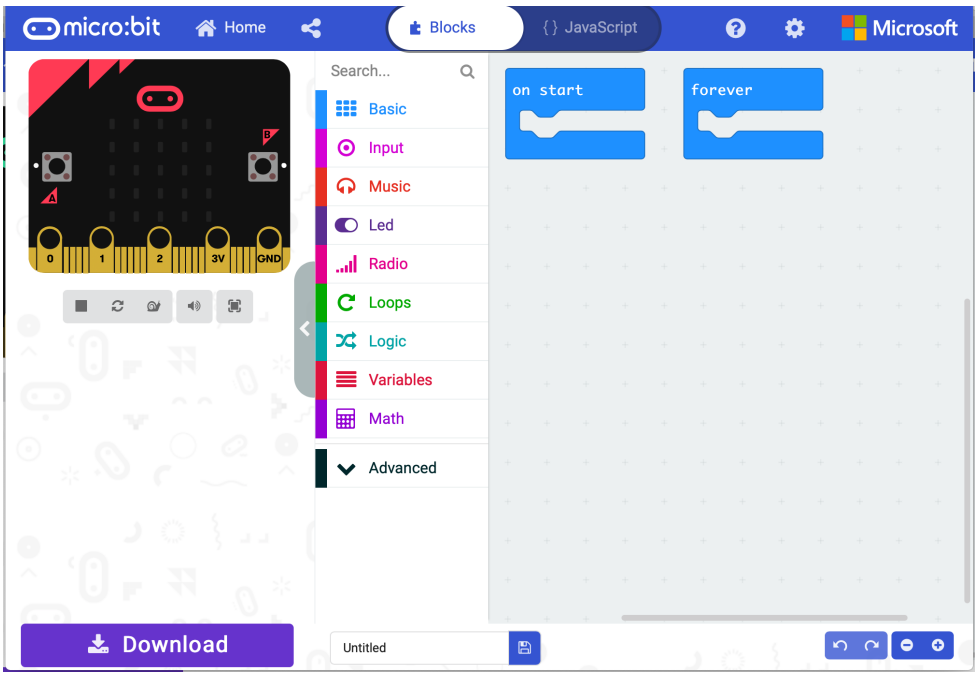
Here’s a video overview from Dexter, the maker of the Gigglebot, if you’d prefer to follow a video of the operational life cycle: https://youtu.be/rLuS1BsOelA

Getting started - coding your first Gigglebot program
PLEASE DO THIS FIRST!

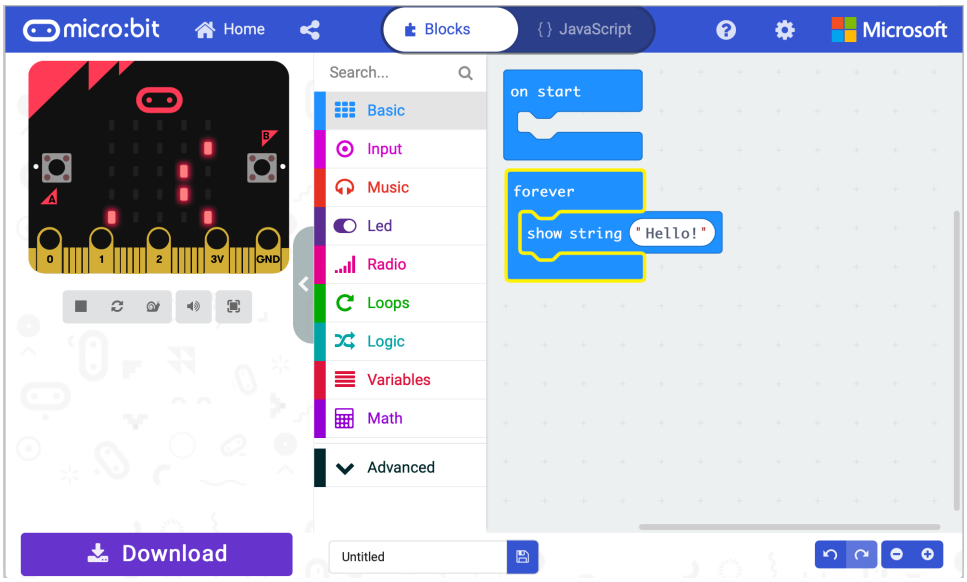
We’ll start with a very simple program to illustrate the “code / flash / test / repeat” life cycle illustrated in the “Life Cycle of a Computer Program” image above. You’ll use this life cycle for each of the subsequent robotics projects within this document.

1) From your laptop, open your web browser and access the website: https://makecode.microbit.org.

2) Click the **+ New Project** button to start a new coding project. Two blocks will appear on the ride side of the screen; “on start” and “forever” as shown below.

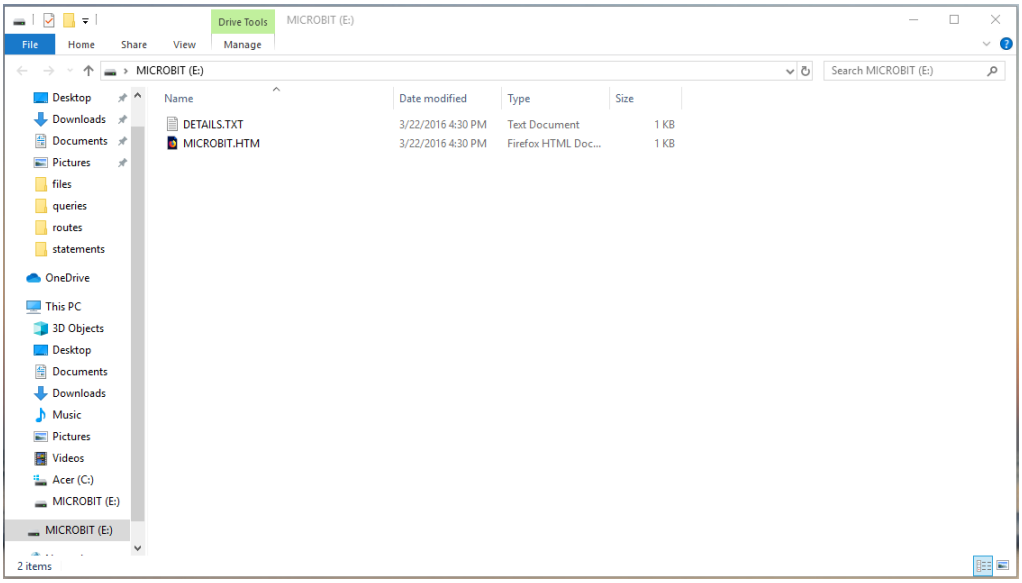


3) Click on the Basic block folder, then drag the “show string “Hello!” block into the “forever” block as shown below. Note that micro:bit emulator on the left side of the screen begins to scroll “Hello!”. Our objective is to make this same scrolling happen on the Gigglebot’s micro:bit.



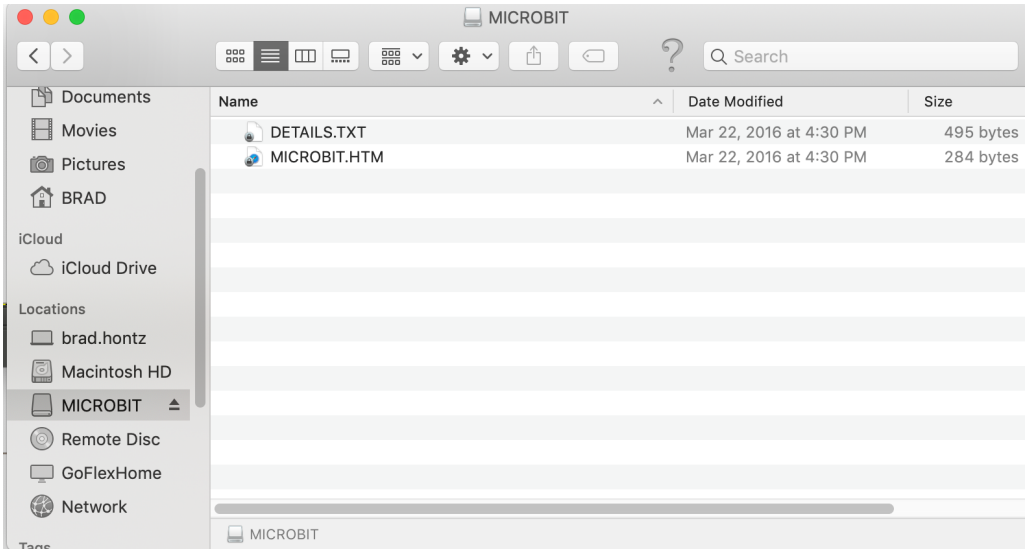
4) Be sure the Gigglebot’s switch is OFF, then carefully remove the micro:bit by pulling it out of (pull up) the Gigglebot’s connector plug.
5) Using the short USB cable provided in the box, plug the micro:bit into a USB port of your laptop.
6) Similar to connecting a USB flash drive, your computer will recognize the micro:bit as a new drive that files can be copied to. Examine your laptop to be sure you understand how the micro:bit drive appears in your file explorer.

Windows computers:



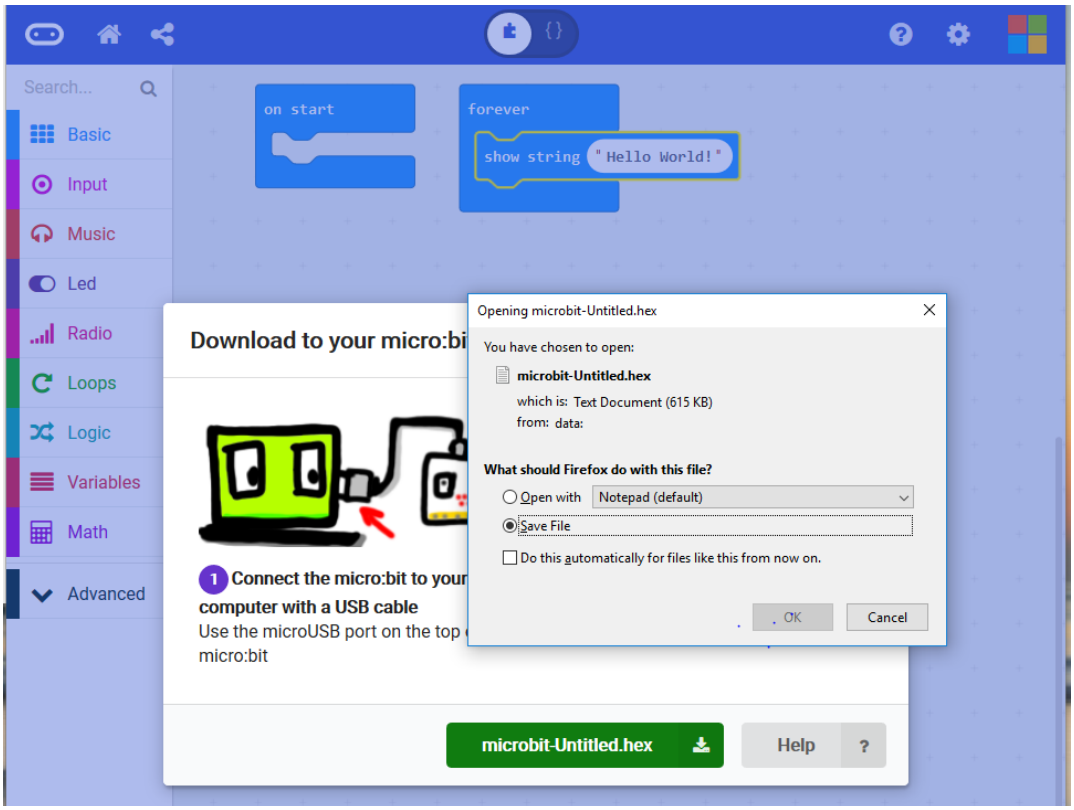
Windows Explorer showing a connected micro:bit as “drive E:”

Mac computers:



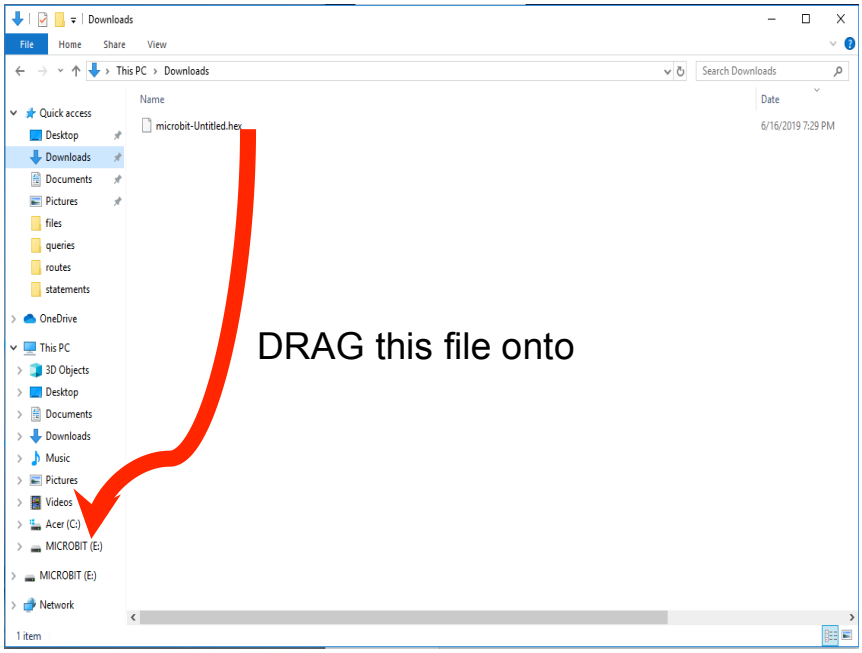
Mac Finder showing a connected micro:bit

7) Click the Microsoft Makecode’s Download button found at the bottom right of the screen. A new file will be saved to your laptop’s Downloads folder, the filename will end with a HEX file extension (example: microbit-Untitled-1.hex).

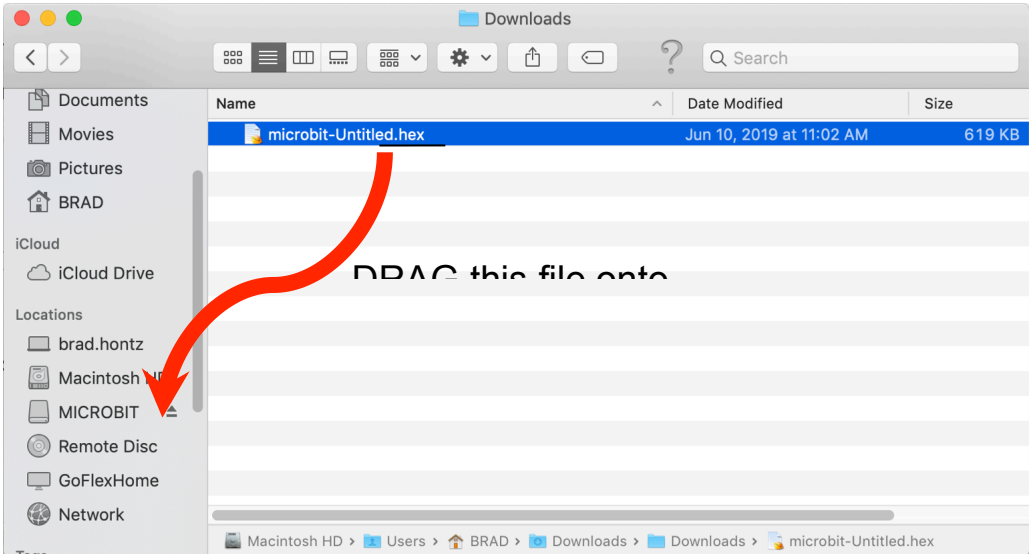


8) Drag (i.e. copy) the HEX file onto the micro:bit. The micro:bit has a yellow light which will flash repeatedly while the file is being copied. This process is called “flashing” the micro:bit. Windows and Mac pictures for this process are provided below.

Windows computers:

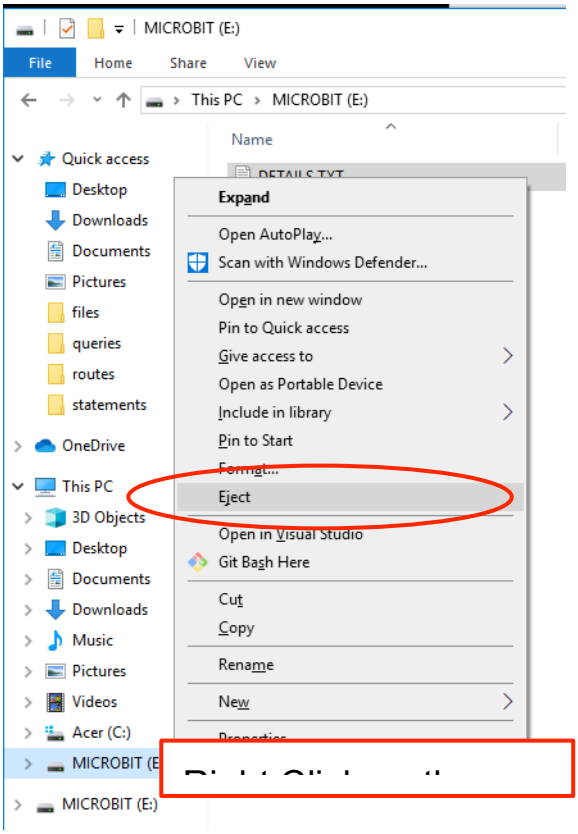


Mac computers:

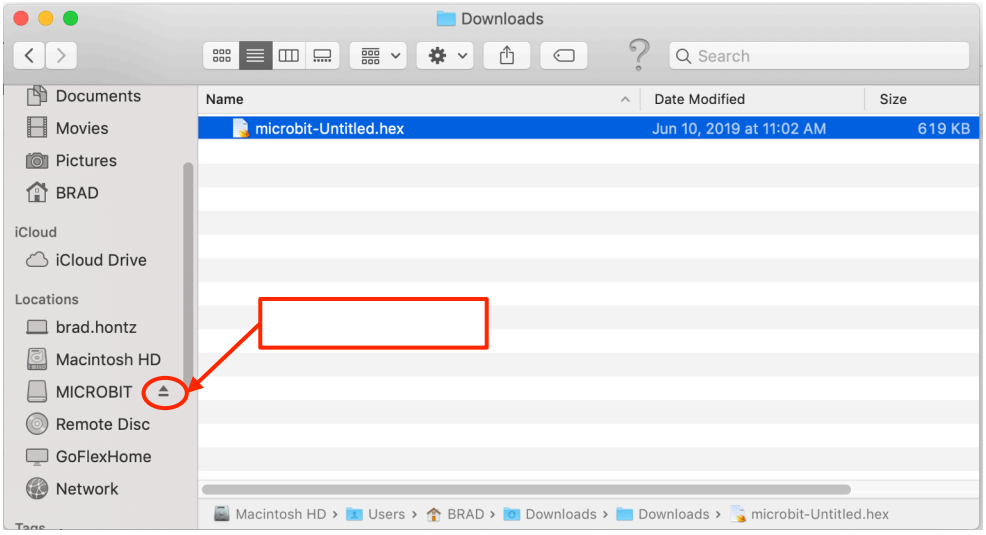


9) You should see “Hello!” begin to scroll on the micro:bit after the code is flashed and with the USB cable still connected to the laptop.
10) **AFTER THE MICRO:BIT’s YELLOW LIGHT STOPS FLASHING**, safely eject the micro:bit from your laptop, just as you would eject a USB flash drive. Windows and Mac pictures are provided below.

Windows computers:



Mac computers:



11) Be sure that the power switch (upper left corner of the Gigglebot) is switched to the OFF position. Disconnect the micro:bit from the laptop and plug it into the Gigglebot's connector such that the micro:bit's "LED grid" (where "Hello!" was shown) is oriented towards the front of the robot.
12) Switch the Gigglebot's power switch ON, and "Hello!" should scroll on the Gigglebot's micro:bit.

Congratulations! You have mastered the "code / flash / test" cycle. Please repeat the steps above until you feel comfortable with this process.

Within each of the subsequent Lessons within this document, we will:

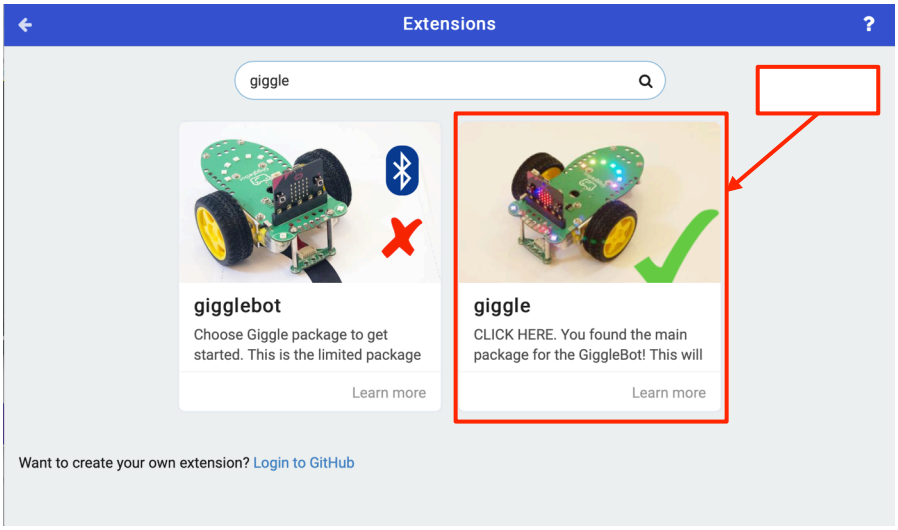
- Write code on our laptop using the Microsoft Makecode website software.
- Download the code from Microsoft Makecode to our laptop's download folder (i.e. the "HEX" file).
- Switch the Gigglebot OFF and then remove the micro:bit.
- Connect the Gigglebot's micro:bit to your laptop via the short USB cable.
- Copy (i.e. "flash") the HEX file from our laptop's Download folder onto the micro:bit.
- Eject the micro:bit, disconnect it from our laptop, and then plug the micro:bit back onto the Gigglebot.
- Turn the Gigglebot's switch ON, which will execute our code!

We will refer to this collective group of steps as "flash and run!".

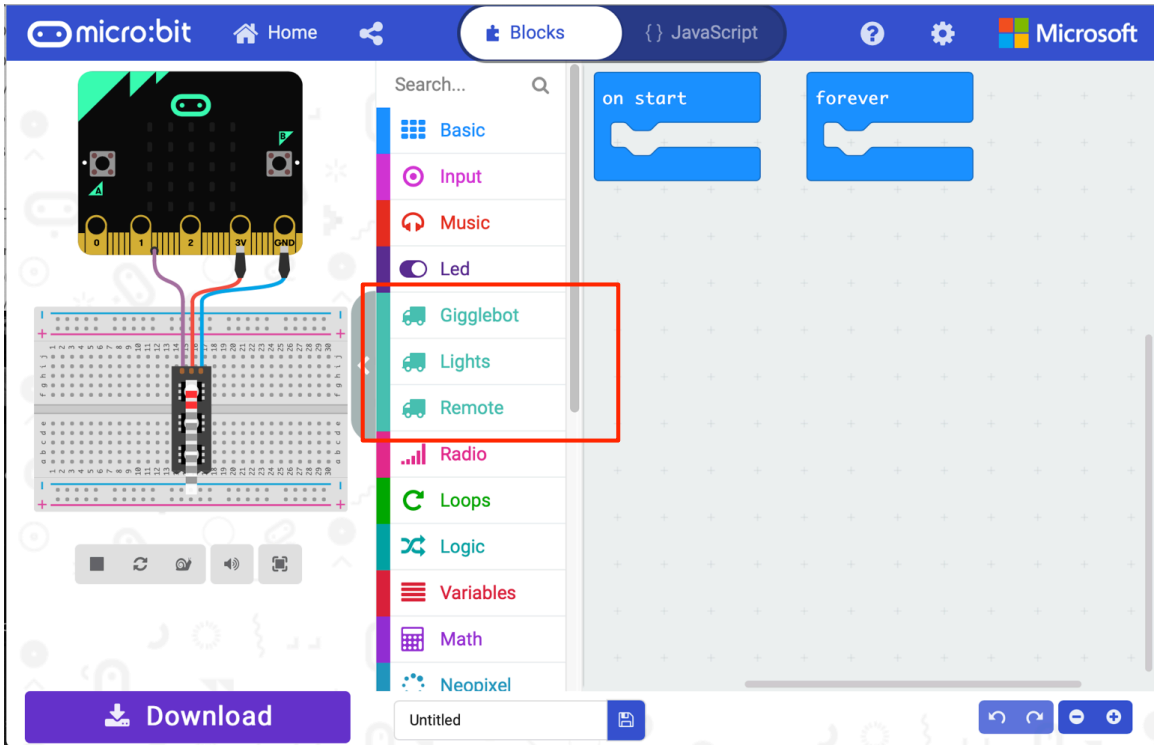
Lesson #1: MAKE OUR ROBOT MOVE IN A SQUARE!

Navigate your laptop browser to https://makecode.microbit.org. Click on the NEW PROJECT button beneath “My Projects”.

Click on the Advanced block folder and then Extensions from the Makecode menu. Enter the word “giggle” into the search bar and hit the Enter key. Click on the Giggle extension (with the green checkbox) as shown below.

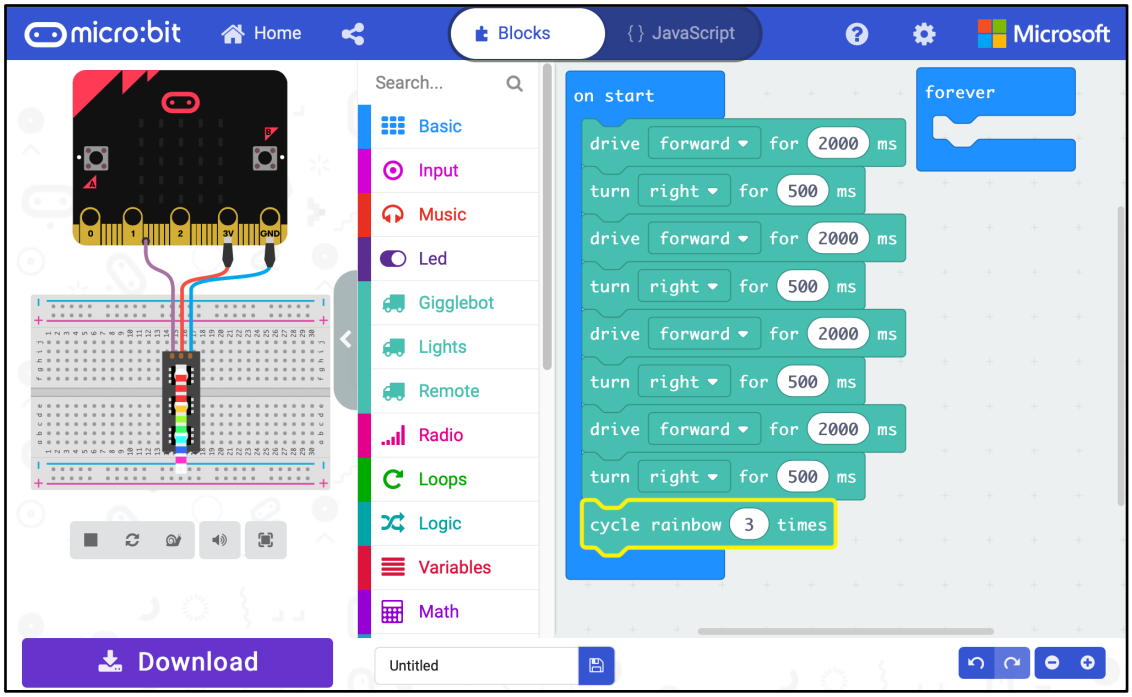


You will now see additional Gigglebot specific block folders as shown below.



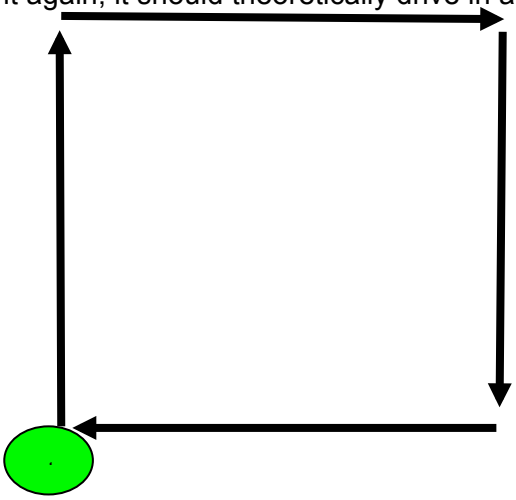
Click on the Gigglebot block folder and drag the “drive forward for 1000 ms” block inside of the blue “on start” block already on the screen. Change the 1000 ms value (1000 ms is equal to 1 second) to 2000 ms; simply click on the 1000 ms value and type in 2000.

Now drag a “turn right for 1000ms” block beneath the “drive forward” block you just modified. Change the “turn right” block 1000ms value to 500ms. Repeat this process until you have 4 pairs of “drive forward” and “turn right” blocks. Just for some extra pizazz, click on the Gigglebot Lights block folder, and add the block “cycle rainbow 3 times” as our last block. Your blocks should look like the image below.



What should we expect?

If our robot drives forward, turns right, drives forward, turns right, drives forward, turns right, drives forward and turns right again, it should theoretically drive in a square as shown below.



Once our robot completes the square, the Gigglebot’s “smile” lights should flash 3 times.

Click the Makecode’s Download button to save our program as a HEX file in our Downloads folder. ***TIP: Keep your Download’s folder sorted by file date/time so that the newest files***

appear at the top of the Downloads folder. That will make it easier to make sure you're actually downloading the LATEST COPY of your code onto the micro:bit.

Attach the micro:bit to your laptop with the USB cable and drag the downloaded HEX file to our micro:bit. Now plug the micro:bit onto the Gigglebot, find some open space on the floor, and turn the Gigglebot on using the on/off switch. Your robot should make right turns in a square like motion.

So why doesn't my Gigglebot make a precise square?

This is an excellent TEACHING MOMENT! Robots don't have a brain like we do; everything the robot does is controlled by our code which could be called an "algorithm". For example, our robot doesn't know if one of its two motors (one for each wheel) is a little stronger than the other; our robot just "turns on" each of the motors for the amount of time (e.g. milliseconds) that we specify in our code. If one motor is a little stronger, the robot will veer in that direction instead of driving in a straight line.

Additionally, our robot doesn't know how fresh the batteries are. The newer the batteries, the more the robot will move or turn for the number of milliseconds we specify. If we have new batteries, our robot may turn 90 degrees in 500 milliseconds. If we have very weak batteries, we may need 1200 milliseconds (for example) in order to turn 90 degrees.

Here are some things you can try to help make your robot move in a (more) precise square:

Drag the blocks you added in the steps above back onto the center "block folders" which will erase the blocks. Now drag a "drive forward" block from the Gigglebots folder into the On Start block. Set the 1000ms value to 5000ms (i.e. 5 seconds). Flash and run this code. Take notice if the robot veers to the right or left. Assuming it does, click on the Gigglebot block folder, and then on the "... more" folder immediately beneath it. Drag the block "correct towards right by 0" above the existing "drive forward" block.

Use the drop down menu on this block to set a right or left direction, depending upon how the robot veered (i.e. set to left if the robot was veering to the right). Replace the "by 0" value with a value of 5 by keying over top of the 0. Now download and flash this code and try again. Repeat this process, increasing the value in increments of 5, until the robot runs in a reasonably straight line. **Leave this revised "correct towards" block within your code, above any "drive forward" and "turn right" blocks you subsequently include in your code.**

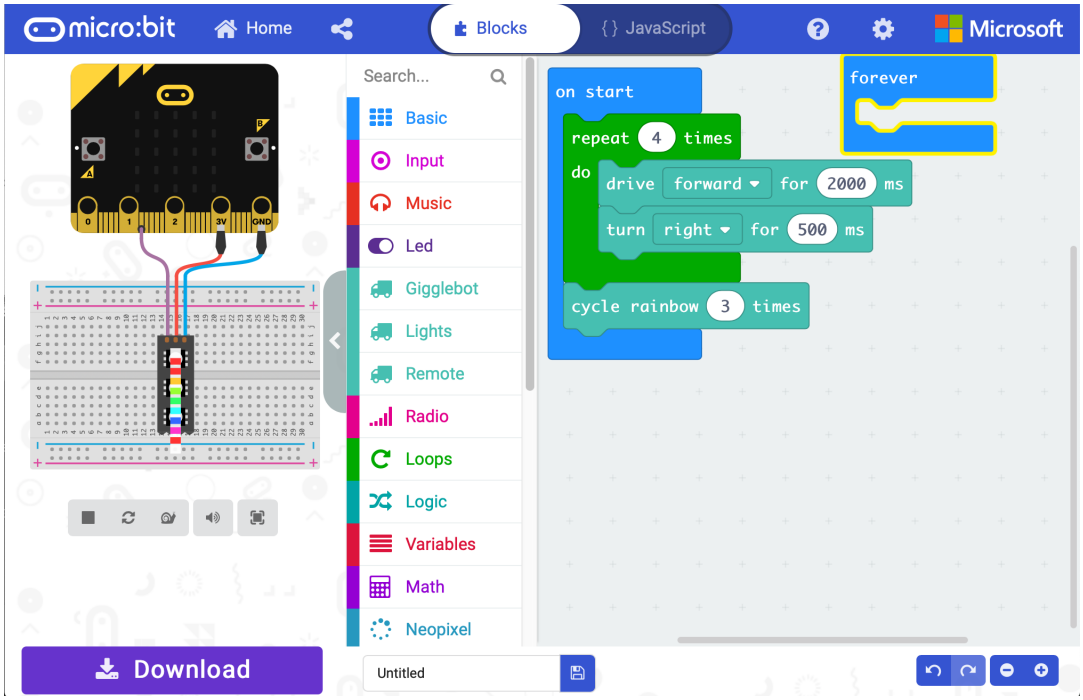
If your robot turns too much (i.e. more than 90 degrees) or too little (less than 90 degrees) to the right, decrease (or increase) the "turn right" values until the robot makes near 90 degree turns. With a little trial and error using these two methods, you will be able to make a fairly good square each and every time!

Let's learn about coding: LOOPS!

You may have noticed that our code contains four identical pairs of “drive forward” and “turn right” blocks. If we had a way to repeat this pairing of blocks four times we could reduce the overall number of blocks in our program, and thereby not only make it easier for us to write the code, but easier to later understand the code and to explain the code to others.

Robotic Engineers would use a LOOP in this situation for that very purpose; a loop eliminates writing redundant or repetitive code and thereby makes the code “cleaner” or more readable by humans programmers.

You may have noted the Loops block folder beneath the Gigglebot block folders. Click on the Loops folder and drag a “repeat 4 times” block into the blue on start block. Now drag a “drive forward” and “turn right” block (with the milliseconds you’d used prior) inside of the “repeat 4 times” loop block. Add the “cycle rainbow 3 times” block beneath (i.e. not inside, beneath) the loop block as shown below.



Flash and run the revised code and verify that it works the same way as our original version with four pairs of drive and turn blocks.

Fun Extra Credit!

- Can you make your robot make left turns instead of right?
- Can you make a near perfect square? Try putting clean paper beneath the Gigglebot and put the marker in the Gigglebot’s marker hole to check.
- Can you make a circle instead of a square?
- Can you make the robot spin in a circle when it completes the square?

Lesson #2: FOLLOW A LIGHT!

In **Lesson 1** we used code to tell our robot what to do. In this lesson we will begin using the Gigglebot's sensors, specifically the two light sensors built on the top of the robot (refer to the ***Parts of a Robot*** section of this document).

Specifically, we will code the robot to "follow" or continue to move in the direction of light source. We will need to use a bright flashlight or the flashlight from a phone for this purpose. It will help if you can dim / darken the room lights for this lesson (or at least when you are running the robot!).

As we noted within the ***Parts of a Robot*** section of this document, there are two light sensors located on the top left and right sides of the robot. We can use our code to determine what these light sensors sense and then have our robot react accordingly. In this case, we'll have our robot turn in the direction (right or left) in which the robot senses the greatest intensity of light. This is a great illustration of using the sensors on a robot to make sense of the robot's environment and react accordingly!

There really is only one key Microsoft Makecode block for following a light, which is unsurprisingly called "follow light". This block is found within the Gigglebot blocks folder. However, let's enhance our code a little in order to obtain the following behavior from our Gigglebot:

- Display a red LED smile on the Gigglebot when the robot is no longer following the light.
- Display a "heartbeat" on the micro:bit's LED grid (e.g. where we'd displayed "Hello!") when the robot is following the light.

We'll start with the existing blue "forever" block as this program can continue to run forever -- we can simply move the light away from the robot to stop the robot. Code blocks added to the "on start" blue block run only one time.

From the Gigglebot Lights block folder, drag "display a red smile" into the "forever" block. Next, click on the Basic block folder, and drag a "pause for 100ms" as the next block. Change the 100ms value to 1000ms (i.e. one second). Next, from the Gigglebot Lights folder, drag a second "display a red smile" beneath our "pause for" block, and use the drop down menu to change red to black. Selecting black essentially turns off the LED smile lights, meaning our code to this point turns on a red smile for 1 second and then turns the red smile off.

From within the Gigglebot block folder, drag the "follow light" block beneath our three existing blocks. Next, from with the Loops block folder, drag the "while True do..." block as the next block. Any subsequent blocks we drag inside of the "while True do..." block will execute until a False condition is encountered.

Rather than using True as the condition for executing the blocks within our loop, we will use the loop condition "while following light". From within the Gigglebot block folder, find the "following light" block that looks like:

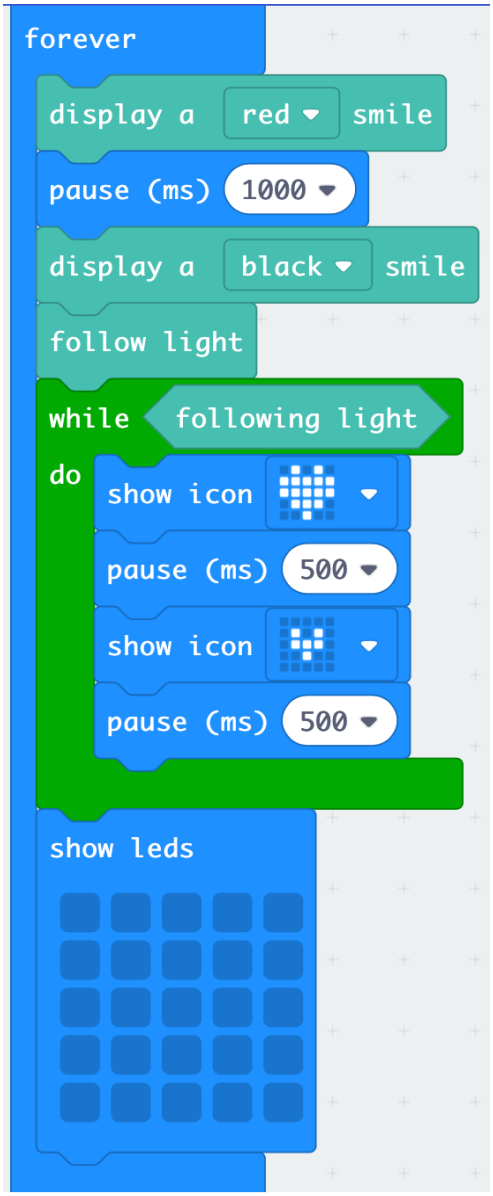
… and drag that over top (to replace) the True condition within the “while True do” block.

Now let’s add a heartbeat within our “while following light do” block so we can tell when our robot is following the light. From the Basic block folder, drag the “show icon <heart>” block inside of our “while following light do” block. Next, from this same Basic folder, drag a “pause for 100ms” block beneath the “show icon <heart>” block. Change the pause value from 100ms to 500ms (i.e. one-half of a second).

We’ll want to duplicate these two blocks immediate beneath. **TIP: If you “right click” on any block, you can select Duplicate from the right click menu, and duplicate that block, including any parameter setting changes (e.g. 500ms from 100ms) you’ve made.** Click on the “show icon <heart>” block’s drop down menu, and select the small heart from the grid of available icons.

Lastly, we need a way to turn off the micro:bit’s display (i.e. clear the hearts) after we leave the loop. Drag the “show leds” block from the Basic folder immediately beneath the “while following light do” block.

Your Makecode blocks should appear as follows:



Final “Follow a Light” Makecode blocks.

Click the Makecode’s Download button to save our program as a HEX file in our Downloads folder. ***TIP: Keep your Download’s folder sorted by file date/time so that the newest files appear at the top of the Downloads folder. That will make it easier to make sure you’re actually downloading the LATEST COPY of your code onto the micro:bit.***

Flash and Run!

Attach the micro:bit to your laptop with the USB cable and drag the downloaded HEX file to our micro:bit. Now plug the micro:bit onto the Gigglebot.

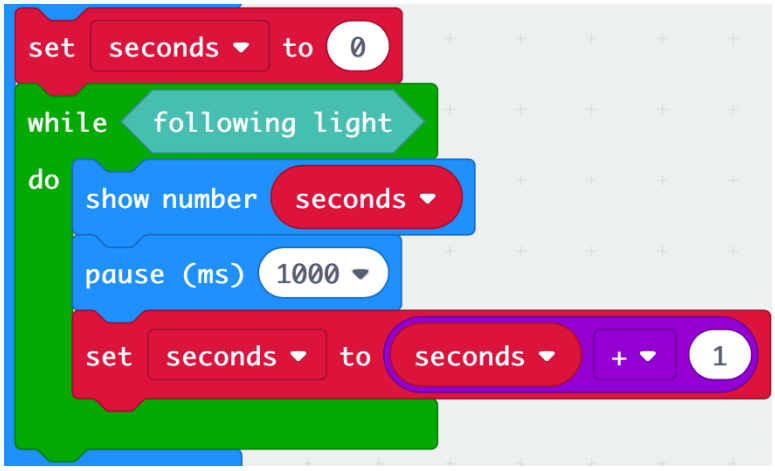
Prepare your flashlight / phone flashlight light source to shine within 6” (or less) on one of the front sides of the robot. Turn the Gigglebot on. Whenever the robot can no longer sense the light source, the red smile will appear. When the robot senses the light source, the robot will move forward, steering towards the light (move the light slowly to left and right), while flashing a “heartbeat” on the Micro:bit’s LED display.

Fun Extra Credit!

- Change the robot's LED smile from red (waiting to find the light) to green (found it!) when the light is found and while it is being followed.

- Use variables, math and show number blocks to display the number of seconds you can keep the robot following the light. See what troop member can keep the robot following the light the longest! (don't forget to remove the "show leds" block which will clear the seconds display)

  Here's a hint:



- If you have TWO robots, carefully use tape or twist ties to attach a small flashlight to the back of the first robot (over the metal ball wheel) and have one robot follow the other! Be sure the first robot makes gentle turns that the light sensing robot (follower) can follow. You might want to use the block "set both motors speed" set to slower; adding this block as the first block within your code.

Lesson #3: LINE FOLLOWING

In **Lesson 2** we used code to tell our robot what to do based upon the light sensors on top of the Gigglebot. In this lesson we use the Gigglebot's reflective light sensors, which are located on the bottom of the Gigglebot close to the table or mat surface (refer to the ***Parts of a Robot*** section of this document).

We will need a mat with a white background with thick (at least 1" wide) glossy black lines. Our objective is to code the robot to "follow" the black line, using the reflective light sensors to tell us if we're on the line. A mat should be provided within GSOC Program in a Box. If there isn't a mat, you can make a mat from a roll of shelving contact paper and black electrical tape or with glossy black spray paint (see appendix for further details). Weight the corners of the mat so it doesn't curl up in use.

We will start with the existing blue "forever" block as this program can continue to run forever, our Gigglebot will keep following the line until it accidentally runs off the line and can no longer find the black line, or until we turn off the Gigglebot using the on/off switch. If our Gigglebot accidentally runs off the line, you can tap the front of the Gigglebot back towards the line and it should reacquire the line and begin to follow it again.

Let's enhance our line following code a little in order to obtain the following behavior from our Gigglebot:

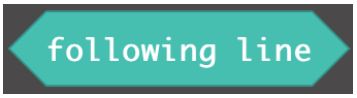
- Set the speed of the Gigglebot motors to SLOW. The reflective light sensors take some time to read the reflective light, so slowing the motors down helps.
- Display a red LED smile on the Gigglebot when the robot is no longer following the line. This makes it easy to know the Gigglebot is off course.
- Display a "heartbeat" on the micro:bit's LED grid (e.g. where we'd displayed "Hello!") when the robot is following the line.

From the Gigglebot block folder, drag "set both motors speed" into the "forever" block. Change the drop down menu on this block from slowest to slower. (You can experiment with this by changing between these two settings).

From the Gigglebot Lights block folder, drag "display a red smile" beneath the "set both motors speed" block. Next, click on the Basic block folder, and drag a "pause for 100ms" as the next block. Change the 100ms value to 1000ms (i.e. one second). Next, from the Gigglebot Lights folder, drag a second "display a red smile" beneath our "pause for" block, and use the drop down menu to change red to black. Selecting black essentially turns off the LED smile lights, meaning our code to this point sets our motors to "slower" and turns on a red smile for 1 second and then turns the red smile off.

From within the Gigglebot block folder, drag the "follow a thick black line" block beneath our three existing blocks. Next, from with the Loops block folder, drag the "while True do" block as the next block. Any subsequent blocks we drag inside of the "while True do..." block will execute until a False condition is encountered.

Rather than using True as the condition for executing the blocks within our loop, we will use the loop condition “while following line”. From within the Gigglebot block folder, find the “following line” block that looks like:



… and drag that over top (to replace) the True condition within the “while True do” block.

Now let’s add a heartbeat within our “while following line do” block so we can tell when our robot is following the line. From the Basic block folder, drag the “show icon <heart>” block inside of our “while following light do” block. Next, from this same Basic folder, drag a “pause for 100ms” block beneath the “show icon <heart>” block. Change the pause value from 100ms to 500ms (i.e. one-half of a second).

We’ll want to duplicate these two blocks immediate beneath. **TIP: If you “right click” on any block, you can select Duplicate from the right click menu, and duplicate that block, including any setting changes (e.g. 500ms from 100ms) you’ve made.** Click on the “show icon <heart>” block’s drop down menu, and select the small heart from the grid of available icons.

Lastly, we need a way to turn off the micro:bit’s display (i.e. clear the hearts) after we leave the loop. Drag the “show leds” block from the Basic folder immediately beneath the “while following light do” block.

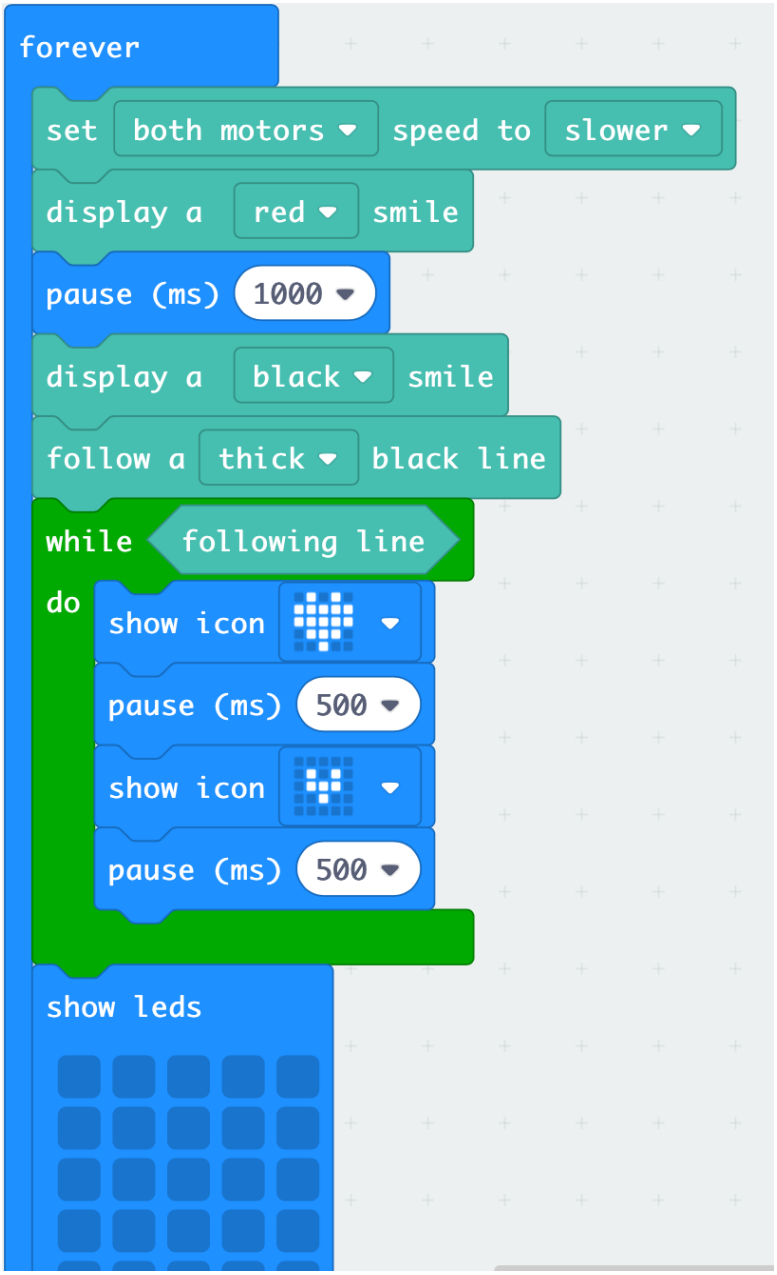
Click the Makecode’s Download button to save our program as a HEX file in our Downloads folder. ***TIP: Keep your Download’s folder sorted by file date/time so that the newest files appear at the top of the Downloads folder. That will make it easier to make sure you’re actually downloading the LATEST COPY of your code onto the micro:bit.***

Flash and Run!

Attach the micro:bit to your laptop with the USB cable and drag the downloaded HEX file to our micro:bit. Eject the micro:bit and disconnect it from your laptop, then plug it back into the Gigglebot. Place the front of the Gigglebot over top of the thick black line, being sure that you can see one or both of the reflective light sensors immediately over top of the black line. Turn the Gigglebot on and it should begin following the line.

If the Gigglebot “drops the line” the red smile will appear. You should be able to tap the front of the Gigglebot back towards the line, and it should reacquire the line and start to follow it once again.

Your Makecode blocks should appear as follows:



Final “Line Following” Makecode blocks.

Fun Extra Credit!

- Use the “A” button on the micro:bit to start and stop the line following program.
- Experiment coding the Gigglebot with micro:python as described in the appendix of this document. An example line following code module is included in the appendix.
- If your mat doesn’t have a white gap within the line, use some white tape to create one. Add blocks that will:
    - Count the number “laps” that the Gigglebot made around the mat course
    - Move the robot forward a little (pass the gap) so the robot will automatically reacquire the black line

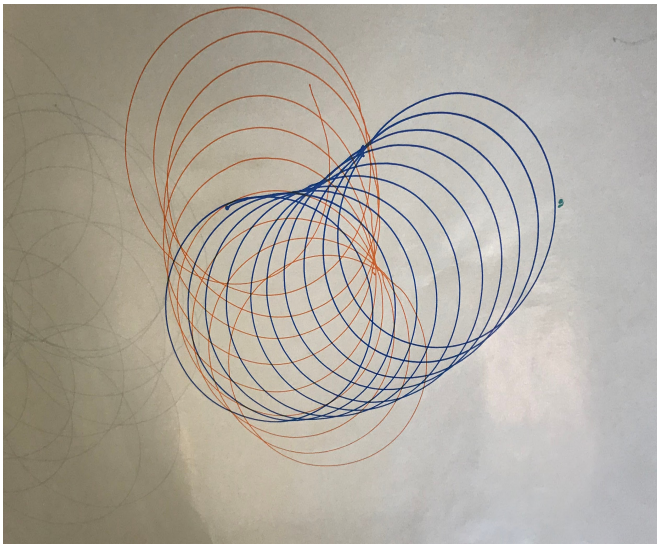
Lesson #4: ROBOT ART

If you had carefully followed the ***Parts of a Robot*** section of this document, you may have noted the "hole for a marker" reference on the Gigglebot's deck. There should be two felt tip markers within the program in a box; one of the markers can be positioned through the Gigglebot's chassis and onto paper below. ***Please do not draw on the line following mat provided within the program in a box; you should plan on supplying your own butcher block or roll paper for this lesson.***

Our objective with this exercise is to move the Gigglebot in fun and repetitive ways in order to create "robotic artwork". The markers should have some tape wrapped around them to tighten up their fit within the marker hole provided on the Gigglebot. The markers don't need to fit snuggly, the tape just takes up slack to prevent the markers from wobbling around while the Gigglebot is moving.

In the code example that follows, we are using a repeating pattern that will take up very little paper / table space. You could (carefully) use a letter size sheet of paper with this code example. Once you have completed the example provided, and you have challenged the girls to consider other "patterns", you should be prepared to utilize larger format roll paper like butcher block.

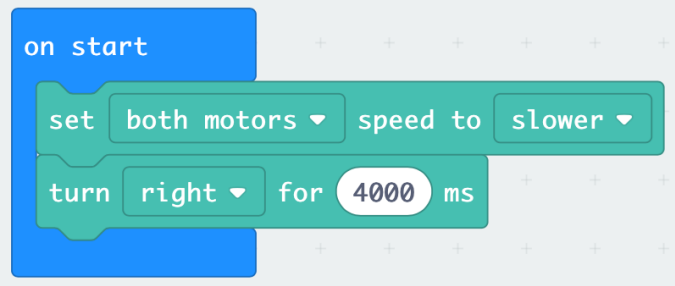
Here's an example of a drawing created using the code example provided below:



In this example we utilized a blue marker pen, then reran the same code example using an orange pen, starting roughly in the same position as our blue pen code run.

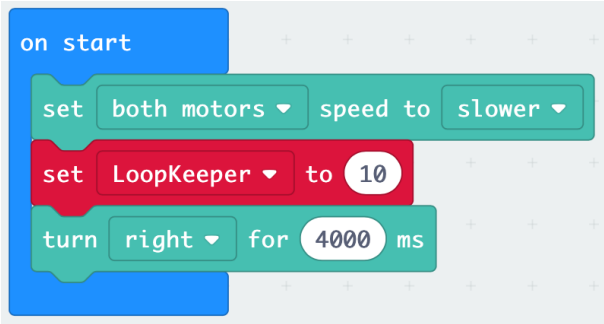
In this example, we will start with the "on start" existing blue block as opposed to the "forever" block, as we want our robot to draw ten iterations (circles) and then stop. To accomplish this, we will need to create a variable that will keep track of the number of iterations and thereby allow us to know when to stop our robot.

As a first step, we'll want to experiment to determine how long it takes for our robot to draw a complete circle. Drag the "set both motor speeds ..." Gigglebot block into the "on start" block, and set the speed parameter to slower. Next, drag the "turn right for …" block beneath the "set motor speeds …" block and set the time parameter to 4000ms as shown below.



Flash and run the code above and see how closely the robot comes to turning in one complete, full circle. Change the 4000 ms parameter as required (i.e. increase the value if the robot doesn't fully complete the circle, or decrease the value if the robot is going past a full circle). Record this value (i.e. milliseconds required to complete a full circle) as we will use it in our final code. 4000 ms worked for creating this document, but your robot could be different.

Next, we need to create a variable that keeps track of the number of loop iterations, or circles, we have drawn. Click on the Variables block folder, then click the "Make a Variable …" button. Name your variable; in this example we used the name ***LoopKeeper***. Once you have named your variable, two new red blocks will appear within the Variables block folder; "set variable to …" and "change LoopKeeper by ... " Drag the "set variable to …" block between the existing "set both motors …" and "turn right for …" blocks and change the "set variable to …" parameter from 0 to 10. This implies that we are planning to draw 10 circles. Our code should look like this at this point:



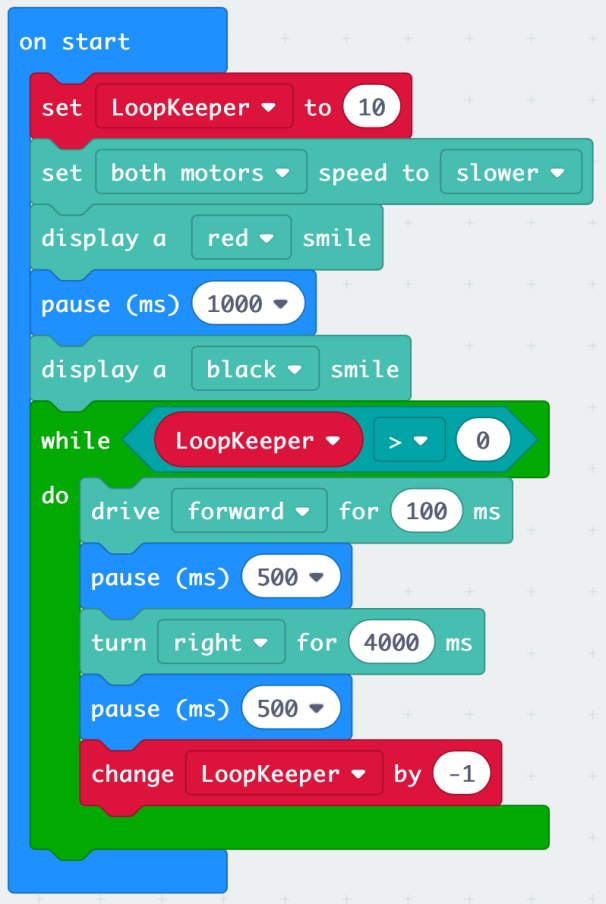
Similar to our previous examples, let's add a red smile so that we will know when our program is about to start. From the Gigglebot Lights block folder, drag "display a red smile" beneath the "set both motors speed" block. Next, click on the Basic block folder, and drag a "pause for 100ms" as the next block. Change the 100ms value to 1000ms (i.e. one second). Next, from the Gigglebot Lights folder, drag a second "display a red smile" beneath our "pause for" block, and use the drop down menu to change red to black.

Now we will add a loop to repeat our circle 10 times. From the Loops block folder, drag the "while true …" block beneath the "display a black smile …" block. We now need to change the "while true …" block's loop terminating condition from "true" to "while variable LoopKeeper is greater than a value of 0". From the Logic block folder, drag the first block beneath the

comparison section and into the “while true …” block, replacing the existing true loop terminating condition. Next, from the Variables block folder, drag the round “LoopKeeper” variable block into the left side comparison section block, replacing the 0 on the left. Use the drop down selector to change the equals sign on the compasion block to “greater than” (>). Blocks inside of our loop will execute as long as the value of variable LoopKeeper is greater than 0, and we have initialized the value of LoopKeeper to 10.

Now we need to add the blocks that will draw a circle and then move the robot a tiny bit before drawing the next circle. From the Gigglebot block folder, drag the “drive forward for …” block into our loop block and set the value to 100ms; this will allow the robot to move a tiny bit forward before drawing the next circle. Next, from the Basic block folder, drag the “pause for …” block beneath the “drive forward for 100ms” block and set the pause parameter to 500 ms (one half a second). Now add the “turn right for …” block we had experimented with initially beneath our “pause for 500ms …” block. Be sure to use the milliseconds value representative of a full robot circle, the value we discovered in our code experiment above. Add another “pause for 500ms …” block beneath our “turn right for …” block.

Lastly, we need a way to be sure that the LoopKeeper value changes or “decrements” with each loop. From the Variable block folder, drag the “change LoopKeeper by …” block into our loop (last block within the loop) and set its parameter to -1. This will subtract a value of one from variable LoopKeeper with each iteration of the loop. The final code should appear as shown below:



Click the Makecode’s Download button to save our program as a HEX file in our Downloads folder. ***TIP: Keep your Download’s folder sorted by file date/time so that the newest files***

appear at the top of the Downloads folder. That will make it easier to make sure you're actually downloading the LATEST COPY of your code onto the micro:bit.

Flash and Run!

Attach the micro:bit to your laptop with the USB cable and drag the downloaded HEX file to our micro:bit. Eject the micro:bit and disconnect it from your laptop, then plug it back into the Gigglebot.

Place the Gigglebot onto the center of your butcher block paper or paper roll, and place a marker pen into the Gigglebot, assuring that the marker is touching the paper. Turn the Gigglebot on and it should begin turning clockwise and drawing a circle, thereafter moving a tiny bit and repeating the process 9 more times before stopping. Turn the Gigglebot off, replace the marker with another color, and place the Gigglebot near the previous starting point. Turn the Gigglebot on to repeat the circle pattern using the new marker color.

Fun Extra Credit!

- Talk about the LoopKeeper variable and how that works. What if we had initially set the LoopKeeper value to 0 and the "change LoopKeeper by …" block parameter to 1 (instead of -1)? What difference would this change make to our code? (TRY IT!)
- Display the "loop iteration" count on the micro:bit's LED screen using the "show number …" block.
- Create an additional OUTSIDE loop for the purposes of moving the robot backwards before drawing the next set of 10 circles. Create an additional variable to control the number of iterations of the outside loop.
- Try drawing squares instead of circles (see lesson 1).
- Review the appendix and use micropython to make turns and as a result, potential new patterns. What other shapes can you make using Micropython?

Appendix

OTHER CODING RESOURCES

edublocks.org
Dexter Industries, maker of the Gigglebot, recommends the use of edublocks (http://edublocks.org). The interface is very similar to the Microsoft MakeCode's (i.e. block coding, you need to add an extension for the Gigglebot blocks, etc.). However, edublocks lets the user switch between block coding and micropython; you can write your code using blocks then switch to "python mode" to see what the equivalent micropython syntax looks like. Microsoft MakeCode allows you to switch between block coding and javascript in a similar manner.

Micropython
Micropython is a subset of the python language that can be used to program microcontrollers like the micro:bit. Here's a link to the micropython documentation specifically oriented towards micro:bit users: https://microbit-micropython.readthedocs.io/en/latest/.

You can use micropython to program the Gigglebot. I would recommend that you utilize the MU editor: https://codewith.mu for this purpose, as the MU editor was designed for teaching, and Dexter Industries supplies a micropython module along with directions for using the MU editor for coding a Gigglebot: https://www.gigglebot.io/pages/program-the-gigglebot-robot-micropython.

Micropython offers additional and finer control of the robot and robot sensor readings relative to the MakeCode and edublocks block coding alternatives. The final page of this Appendix includes source code for a line following program that was used for a Gigglebot line following table activity at the 2019 OC Fairgrounds Imaginology event.

Micro:bit (in general)
The micro:bit is a great standalone resource for introducing girls to maker skills and programming. After you have mastered the "flash and run" process described within this document, an entire world of maker and coding activities is open to you. Start here: http://microbit.org to learn more. Please note that the GSOC Robotics Program in a Box includes a micro:bit power supply (2 AA battery pack that plugs into the micro:bit) as well as a micro:bit "breadboard" attachment. As time permits, the STEM patrol will add micro:bit projects to the Robotics Program In a Box.

Extensions to the Gigglebot (NOT INCLUDED within the Program in a Box)
The Gigglebot includes two servo ports and two jacks to accept a distance sensor. Servos are small motors that would allow creation of a "dump truck" or moving arm to be built and attached to the Gigglebot, controlled with either micropython or Makecode blocks. See https://www.gigglebot.io/products/servo-motor-for-gigglebot-set-of-2 for additional information.

Dexter Industries sells an ultrasonic distance that can be plugged onto the front of the Gigglebot to allow the Gigglebot to determine how close it is to an object it is approaching. For example, with a distance sensor, we could write code that would allow a Gigglebot to know it is

approaching a wall and to then turn around and reverse its direction. See https://www.gigglebot.io/products/distance-sensor for additional information.

In both cases above, be sure to search YouTube for creative implementations of the Gigglebot with these extensions.

Making a line following mat
The Gigglebot's reflective light sensors should be used with thick, black glossy lines. I have found that paper-printed lines are difficult to follow, you need to use glossy lines made with either electrical tape or paint. The original mats provided within the "program in a box" kits were created using a roll of white contact paper (don't remove the backing!) and spray painted using a thin poster board mask cut with a razor knife and glossy black spray paint. It may be possible to use a vinyl printed mat, assuming that the black line is glossy.

Annual Coolest Projects competition
If your troop is excited about the Gigglebot, please be sure to share a link to Coolest Projects http://coolestprojects.org. The GSOC would love to sponsor girls or a team of girls to enter the annual Coolest Projects competition! In a nutshell, girls could create ANY PROJECT involving technology and showcase it at a Coolest Projects event. Coolest Projects is not a science fair, but rather a celebration of STEM education and having fun with STEM for ages 6 through high school.

In 2018 and 2019, the annual U.S. Coolest Projects event was held at the Orange County Discovery Cube. If you would like more information, please reach out to the GSOC STEM team or GSOC volunteer STEM patrol and check the Coolest Projects website for event scheduling.

Robotics Competitions and Societies

The GSOC has supported multiple **First Lego League FLL** (http://www.usfirst.org) robotics teams in the past. Therefore, girls could reach out to prior or existing GSOC FLL team members to fulfill the requirement of speaking with a robotics team member, or they could attend a GSOC team's practice or mock tournament events. The GSOC has hosted mock tournaments for the GSOC FLL teams for the past few years.

The **World Robotic Olympiad** https://www.wro-association.org has a local chapter located in Long Beach. Teams meet for practice competitions, and troops could contact a local team to visit with them and fulfill badge requirements.

The **Robotics Society of Southern California** https://www.rssc.org meets monthly in Long Beach. This society supports youth robotics education and often hosts youth open challenges or competitions at their monthly meetings. Reach out to the RSSC for more information and to visit in order to fulfill badge requirements.

Careers and Colleges

Most four year universities offer a robotics club or have robotics classes associated with their Engineering & Computer Sciences departments. GSOC maintains an association with Cal

State Fullerton's College of Engineering and Computer Science, which has an excellent robotics practice and club. Contact the GSOC STEM staff for and introduction and possible tour.

Leading university programs with both undergrad and graduate programs include:

Carnegie Mellon: https://www.ri.cmu.edu

MIT: http://roboteam.mit.edu

Georgia Tech: http://robotics.gatech.edu

Oregon State: http://robotics.oregonstate.edu

University of Michigan: http://robotics.umich.edu

Career Information:

NASA Robotics Career Corner:
http://www.nasa.gov/audience/foreducators/robotics/careercorner/index.html

Boston Dynamics Robotics:
https://www.bostondynamics.com/robots

Brain Corporation (San Diego):
https://www.braincorp.com

Micropython code example - line following program with lap counter.

```
"""
    2019 GSOC Imaginology Line Following Table Activity
    Flash to Gigglebot and have it follow a black thick
    line on a tabletop mat.
"""

from microbit import *
from gigglebot import *

button_a.was_pressed()
button_b.was_pressed()
display.show(Image.YES)

set_speed(40, 40)
threshold = 60
lapcounter = 0

while True:
    # this conditional supports viewing the reflective sensors
    # values on the micro:bit LED display
    if button_a.is_pressed() and button_b.is_pressed():
        right, left = read_sensor(LINE_SENSOR, BOTH)
        display.scroll(left)
        display.scroll(right)

    if button_a.is_pressed():
        while not button_b.is_pressed():
            right, left = read_sensor(LINE_SENSOR, BOTH)
            if left < threshold and right < threshold:
                display.show(Image.ARROW_S)
                drive(FORWARD)
            elif left > threshold and right > threshold:
                stop()
                lapcounter = lapcounter + 1
                drive(FORWARD, 500)
                display.scroll(lapcounter)
            elif left > threshold and right < threshold:
                display.show(Image.ARROW_E)
                turn(RIGHT)
            elif left < threshold and right > threshold:
                display.show(Image.ARROW_W)
                turn(LEFT)
        stop()
        display.show(Image.YES)
```