## Programming 101 Lesson Overview

<table>
<thead>
<tr>
<th>Title</th>
<th>Programming 101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade Level Target / Range</td>
<td>9 - 12</td>
</tr>
<tr>
<td>Subject Area(s)</td>
<td>engineering, computer science</td>
</tr>
<tr>
<td>Time Required</td>
<td>4-6 hours</td>
</tr>
<tr>
<td>Group Size</td>
<td>2-3 students per group</td>
</tr>
</tbody>
</table>

### Materials Needed

- **Part 1: Introduction to Programming**
  - Programming 101 (Google Slides)
  - The Cheese Factory Problem (student guide)
  - Cheese Factory Problem Programming Lines (print several for each group)

- **Part 2: MBlock Programming**
  - Pre-installed mBlock software or access to web version.
  - Makeblock mBot robot for each group
  - MeSound Sensor
  - User Guide to mBlock (for reference only)
  - User Guide specific to mBot (for reference only)
  - Instructions for Setting up the Robot
  - Beginner Programming Challenges (student guide)

### Assessments
- Assess student progress and understanding throughout the lesson as they try the cheese factory problem and beginner programming challenges.

### Expendable Cost Per Group
- Prices are Estimates as of March, 2022
- Note: The software is free.
- Makeblock mBot Robot - $80
- MakeBlock Me Sound Sensor with Potentiometer - $7
- Four AAA Batteries - $2

### Key Words
- robotics, block programming, mbot robot, visual programming language

### National Educational
- Next Generation Science Standards (NGSS)
Introduction

This lesson is intended to introduce students to programming the mbot robot. The mbot can be programmed using the mblock programming language. This is a visual programming language. In contrast to most programming languages, users write a program by manipulating elements graphically instead of typing text. While visual programming languages are often considered simpler, there are many examples of specialized visual programming languages used in engineering and science (“Visual Programming Language”, 2022). Therefore, depending on the context, visual programming languages can be compelling alternatives to text-based programming.

This unit was created collaboratively with faculty from the University of Cincinnati College of Arts and Sciences, College of Engineering, and School of Education. Combining biology with engineering activities provides students with a unique opportunity to understand the parallels between animal and robot behavior and sensory/sensor function and addresses broad Next Generation Science Standards (NGSS Lead States, 2013) and International Society for Technology in Education Standards (International Society for Technology in Education, 2022).

Investigating / Essential Questions

- How can instructions be written so that a computer understands them?
- How can a robot be controlled through programming?

Learning Objectives

1. Students will write algorithms.
2. Students will use computational thinking to design solutions.
3. Students will learn to use a visual programming language.
4. Students will be able to write basic programs that control a robot’s actions.
5. Students will develop, test, and refine prototypes.

Prerequisite Student Knowledge

Algebra I

Instructional Summary

In part one of this lesson, students learn about programming without using a computer. They will learn about the challenges faced in programming and how to write algorithms to solve simple problems. In the second part of the lesson, students will set up the mbot robot, understand some basic commands in the mblock software, and then learn to program through challenges that specify specific tasks for the robot to accomplish.
**Instructional Plan**

**What is Programming?**

**1.1 Introduction and Motivation**

Before starting with the mblock programming language, it is useful to define programming. Programming is writing step-by-step instructions for a machine, such as a computer. Or a robot, which is essentially a computer (plus some sensors and motors). The set of instructions you can pick from in writing the program is called a programming language. In this lesson, the programming language will be mblock. Mblock provides the instructions the robot understands.

Prior to delving into programming the robot, explore the challenges faced by programmers through the simple (made-up) example of a programming language. Engage students with the scenario pictured below (also on slide 2 of the Programming 101 Slides).

![Illustration of the hypothetical programming problem discussed above. You are an engineer hired to program this robot to reach the house's front door (indicated with an arrow) when the bell rings. The robot should be able to get to the front door from any location in the house.]

Discuss with students the information that follows.

In solving this, you are faced with two different problems: (1) understanding how to solve the problem in principle and (2) translating the algorithm into a set of programming language instructions.

**Problem 1: Understanding how to solve the problem in principle.** You will need to understand the problem well enough to formulate how you might solve it:

1. What are the main difficulties in solving the problem?
2. What are possible solutions?
3. How can you simplify the problem?
4. How will you break the larger problem into different steps?

The problem faced in this example is essentially a navigation problem (how to get from A to B?). Navigating is a difficult problem in robotics. Many people have proposed different solutions, but the problem has not yet been solved completely. Robot navigation is complicated by various factors. To illustrate this, we just discuss one of them. Navigating from A to B requires recognizing places A and B (and most likely, the places in between them). Let’s say the robot uses a camera as its main sensor. Visually identifying locations is not trivial for robots. The same site (say a room) might look very different depending on your exact position and orientation.
Also, lighting can change how a place looks. Another issue is that environments are not static: they might change over time. This is illustrated in the image below. Therefore some possible answers to the three questions above are the following:

1. One problem in navigating from room to room is that the robot needs to recognize individual rooms (preferably its location within a room). This is difficult for numerous reasons, including changes in lighting and objects being moved around.

2. People have come up with various image processing methods to improve place recognition. We will not discuss these further as these are rather technical.

3. As visual place recognition is difficult and requires some fancy image processing, other people have tried to simplify the problem. Often this involves placing some markers or beacons in the environment the robot can use to recognize places.

Show students the images that illustrate the challenges with visual recognition on the Programming 101 Google Slide (slides 3 & 4). Ask them what they notice about the pictures and how that might relate to a robot recognizing a place.

Based on the answers to questions (1-3), you might decide to simplify the problem by using markers. In particular, you decide to equip the house with a grid of markers on the floor. The robot will store a list of the makers with their locations in the house. In this case, your algorithm could read something like the following:

Randomly move around until you find a marker. Once you see a marker on the floor, look up its position in your list of markers. Use this to calculate the direction to the door to the next room. Do this until you arrive at the marker near the front door.

Note: This is a simple algorithm. In reality, your algorithm will need to deal with different contingencies. For example, the robot will need instructions on how to avoid obstacles. Or what to do if it cannot find a marker. And so forth.

This description is an (elementary) example of an algorithm (and a possible answer to question 4 above): a recipe for solving a problem.

Problem 2: Translating the algorithm into a set of programming language instructions.

Here, a second problem, apart from the problem analysis performed by answering questions 1-4, is translating the algorithm into a set of programming language instructions. This is the implementation problem: how do you string together a set of instructions such that they
implement your algorithm? In keeping with our example, the programming language for the robot might provide the following limited set of commands:

- Move forward xx meter
- Turn Left xx degrees
- Turn Right xx degrees
- Stop
- Get the distance to obstacles
- Read line sensor status
- If x, then do y

This is a trivial programming language, but it serves as an example. The second programming problem boils down to arranging these commands into a program that implements your algorithm.

When students start programming, both problems might be challenging. Students might struggle to come up with an algorithm to a challenge that, at least in principle, addresses the issue. Simultaneously, students are not yet familiar with the set of instructions provided with the programming language. Therefore they might struggle with the translation problem. Over time, both will become easier: students will notice that specific recipes pop up over and over again in different contexts. Also, students’ understanding of the programming language increases. The only way to get better at this is through practice. Some resources estimate it takes about ten years to evolve from being a novice to mastering programming. Luckily, this does not mean that it will take years to be sufficiently experienced to start enjoying programming. To give students a feeling about these two problems, have them play the sandwich factory game introduced in the following section.

1.2 Procedure
The Sandwich Factory Game

The goal of this game is to confront students with the two programming challenges outlined above. After completing the challenge, students should have experienced that programming consists of two problems. First, students need to understand how a cheese sandwich can be built, in principle. Second, students need to shoehorn their intuition into the commands provided by the programming language. This requires understanding the commands. This exercise aims to encourage students to perform a problem analysis before they start coding when they are programming robots.

Give students a copy of The Cheese Factory Problem. After students have read and understood the problem, show them slide 5 on the Programming 101 Google Slide. The image is an example of a valid program. However, the program does not make sandwiches. Discuss as a class why it does not.

Using these four commands, students should try to write a program that results in proper sandwiches being built on the truck's back. When playing this game with students, print the
commands above on paper slips. This allows students to write a program by organizing the slips of paper. Make sure to provide the students with multiple instances of each command.

Solution

This game has multiple solutions. Before presenting a possible program, let's revisit the problem analysis steps above and apply them to the current challenge.

<table>
<thead>
<tr>
<th>What are the main difficulties in solving the problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The main issue is the conveyor belt at location 1 delivers items in random order. Therefore, you can not assume that the next article will be useful to you: it might provide several bread slices while you need some cheese. Therefore, you will need a method of selecting those items you need and discarding what you do not need. A second, perhaps minor problem, is that you can not pick something up from location 1 if you're already holding something.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What are possible solutions?</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can check whether a picked-up item is useful by considering whether it is on top at location 2. If location 2 shows cheese, you need some bread. If it shows bread, you need some cheese.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How can you simplify the problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>This problem can not be readily simplified. However, you could encourage students to think about changes to the problem that might simplify things. For example, what if conveyor belts 1 and 2 were connected? Perhaps the robot could simply wait at location 1 for whatever item it needs while superfluous items move out to other machines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How will you break the larger problem into different steps?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The problem can be sliced in more than one way. For example:</td>
</tr>
<tr>
<td>• We can start by picking up an item from conveyor belt 1.</td>
</tr>
<tr>
<td>• Go to location 3.</td>
</tr>
<tr>
<td>• If the item you see at location 3 is not the same as you're holding: drop it.</td>
</tr>
<tr>
<td>• If the item you see at location 3 is the same as you're holding: move to location 2 and drop it</td>
</tr>
<tr>
<td>• Go back to location 1</td>
</tr>
</tbody>
</table>

The algorithm above can be implemented in different ways. The only way to know whether a student's program works is to step through it and see the result (this is also what programmers do: run the program and see what happens).
One possible solution is the following:

- Goto location 1
- Pick up
- Goto location 3
- If holding cheese and below is bread:
  - Drop
- If holding bread and below is cheese:
  - Drop
- Goto location 2
- Drop
- Go to step 1

This solution picks up whatever is presented at location 1. Next, it moves to location 3. The arm will only drop its load if whatever it is holding is not what is currently showing at location 3. Remember that whenever a complete sandwich is produced, the truck carries it away. Therefore, if we see bread at location 3, we can assume it is the bottom (start) of a new sandwich. Finally, the robot moves to location 2. There it drops whatever it is holding. It is possible that the robot is not holding anything. However, executing the drop command ensures that the robot's arm is empty when moving back to location 1.

If students think this example is not realistic, you can reassure them. Robots often come with a specific set of instructions, which are, in essence, a specialized programming language for this particular robot. For example, the robot arms distributed by ST Robotics use a custom programming language called roboforth. The brave could consult this manual for the robotforth language. The message here is that specialized programming languages for specialized robots are common.

Part 2: Mblock
2.1 Introduction and Motivation
Students will be excited to get their hands on a robot and to try out the software.

2.1.1 Mblock Software Set Up

Mblock can be downloaded free from the web. There is also an online version available. Here, we use version 5, which is available for Windows and Mac. After installing and starting the application, mblock might ask you to log in or create an account. For this tutorial, an account is not necessary. Slide 6 on the Programming 101 Google Slide has a screenshot of the main interface to show students (also shown below). Overview the main interface with students.
1. This area is used for constructing a program. You will drag and drop blocks onto this field to form a program.

2. This area of the window lists the blocks that can be used in creating a program. The blocks are organized into categories such as [Sensing] and [Control]. To use a block, you drag and drop it to the area (1). Notice the [extension] button at the bottom of this area (2). Additional blocks can be added by installing extensions.

3. This part of the screen will show the values of variables in a program. This is useful information when trying to figure out why a program does not work as expected.

4. Here the robots for which blocks are available are listed. When first starting the program, the mbot is not listed. This means we have to load the code for the mbot. We will do this in a later step.

2.1.2 Connecting the Robot

This section and the following assume that students have built the robot according to the instructions. Use slide 7 or 8 on the Programming 101 Google Slide to give students instructions on how to connect the robot to the computer.

The older version of mbot comes with a dongle that allows connecting the robot to your computer. The newer version uses Bluetooth. Use the instructions according to your robot’s connection type to connect to the computer.

Connect with a dongle as follows: Plug the dongle into the computer.

1. Switch on the robot.
2. Click the (+) sign in the window's device area (area 4, as indicated in the screenshot above).
3. A screen will pop up, allowing you to select the mbot. Select the mbot (see image and arrow below) and click ok.
4. Click the blue connect button. This will bring up a window. At the top of the window, select [2.4Ghz]. Next click [Connect]. The robot is now connected.
*Be careful to keep each dongle with its designated robot: each dongle connects to a specific robot.

Connect with Bluetooth as follows:

1. Turn on mBot.
2. From the Connect menu select “Bluetooth” > “Discover”.
3. mBlock will scan for available Bluetooth devices and populate a list.
4. mBlock will connect to mBot.

Adding the mbot to the list of devices.

Connecting to the robot using the dongle.
Step 1: click connect.
Step 2: select 2.4GHz.
Step 3: click connect.
2.1.3 Programming: Where to Start?

Starting to construct a program can be daunting. Therefore, it is important to give your students some pointers that should make it easier to get started. You can also outline the structure for most of the programs you will be writing.

The information for this section is on the Programming 101 Google Slide (slides 9 - 25). There is also a user guide available within the software for reference.

2.2 Procedure

It's now time for students to try to write some simple programs on their own. This section lists a few programming challenges students can complete using the robot. The first three do not use motors. Therefore, the robots will be stationary while solving these three challenges. This simplifies testing the programs. Trying to figure out what is wrong with a program while a robot is bumping around the room is hard. The programming challenges are written for students on this document.

Challenge 1: Blinking LEDs

Challenge: Construct a program that switches on the onboard LEDs for a second, turns them off for a second.

In this challenge, students will construct a program to make the onboard LEDs of the robot blink. Blinking LEDs is a prevalent first example in tutorials about programming hardware. Here, we adhere to this tradition. An example program is below.

The program below repeatedly sets the value of the onboard LEDs to red. Then it waits for a second before switching the LEDs off. Next, the program pauses for a second before this cycle repeats. Notice how the program incorporates the template introduced above. This program's backbone consists of the [if flag is clicked] and [forever] blocks.

To run this program, you need to connect your robot to the computer, as explained above. Next, you click the green flag in the left upper corner of the screen (below the panda).
Challenge 2: Using the Onboard Light Sensor

Challenge: Construct a program that briefly blinks the onboard LEDs when the onboard light sensor registers a value smaller than 500.

The robot has an onboard light intensity sensor. The sensor gives a value from 0 to 1000, depending on the light's intensity falling on the sensor. In this example, we will use this sensor to program the robot to do the following:

1) Read the onboard light sensor and store the result in a variable.
2) If the light value is smaller than 500, blink the lights briefly.

The program below is one solution to this challenge. The program starts by switching the LEDs to ensure they are off at the program's start. Next, the program repeatedly reads the value of the onboard light sensor. If the sensor (stored in the variable [light]) is smaller than 500, the program switches on and off the onboard LEDs.
When running this program, you should see the light variable's value shown in the panda window. For example, in the screenshot below, the current value of the light variable is 980. If you cover the top of the robot, the light value should decrease. Whenever the light value is below 500, the onboard LEDs will flash.

**Challenge 3: Sound Detection**

*Challenge:* Construct a program that blinks the LEDs if the sound is louder than a set value. The duration of the LEDs' blink should depend on the loudness of the sound.

Using this new block, we will program the robot switch on the onboard LEDs if the pickup sound's loudness is larger than a set value. The LEDs will be switched on for longer if the sound was louder.

There are various ways in which to implement this. The example below sets a variable min = 200, if the loudness is lower than this value, the LEDs are not switched. If the loudness is higher than this threshold, the LEDs are switch on for a duration given as follows:

$$\text{Duration} = \frac{\text{loudness} - \text{min}}{50}$$

With min set to 200

This program should respond to noise: louder noises result in a longer blink of the LEDs.
Challenge 4: Programming the Motors

The following programming challenge will use the motors. We will program the robot to maintain a fixed distance from an object using the sonar sensor. When starting the program, the robot will read out the distance to the nearest object (detected by the sonar). Next, the robot will move forward if the distance increases and backward if the distance decreases.

Challenge: Program the robot to keep a fixed distance from an object in front of the sonar.

To the left is an example of this challenge in action. The robot tries to maintain a fixed distance to the box. Moving the box back and forth causes the robot to move back and forth.

For this exercise, we will use the blocks from the [action] category. These blocks allow you to have the robot move forward or backward and turn. These blocks assume that you have connected the motors to the correct motor ports (M1 and M2). If you notice the robot moving in the opposite direction you ask it to move, you have probably swapped the motor connectors.

As always, there are different solutions. The program below is one possible solution.
This program works as follows. At the start of the program, we read the nearest object's distance and store it in a variable [preferred distance] (see step 1). Next, we repeat several steps forever. We read the current distance and calculate the difference with the preferred distance. If the absolute value of the difference is smaller than 2 cm (step 3), we stop the robot (step 6). In this case, we do not move the robot. If abs(difference) > 2, we have to move the robot. If the difference < 0, we are too close to the object, and we switch on the motors to move backward. In case the difference > 0, we move forward.

Moving the box in the picture back and forth results in the robot moving back and forth. It tries to maintain a fixed distance to the box.

In this program, we tolerate an error of 2 cm. You can experiment with lower values. However, you will notice that the program will become unstable at some point: the robot will never stop and keeps oscillating back and forth. This is because the distance read by the sonar is noisy. Also, the program has some delay between moving and getting updated distance values. Even if the sonar were noise-free, this delay would cause some oscillation.
Extensions
Have students do their own programming for the lessons included in the Biology Meets Engineering curriculum instead of giving them access to pre-written programs.

Assessments
Assess student progress and understanding throughout the lesson as they try the cheese factory problem and beginner programming challenges.

Supporting Activity Information
The mBlock user guide and tutorials (available within the software) are helpful in learning the visual programming language. There is also a user guide specific to programming the mBot robot.
**National Educational Standards**

<table>
<thead>
<tr>
<th>NGSS</th>
<th>HS-ETS1-2. Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Science and Engineering Practices:</strong></td>
<td></td>
</tr>
<tr>
<td>● Asking Questions and Defining Problems</td>
<td></td>
</tr>
<tr>
<td>● Developing and Using Models</td>
<td></td>
</tr>
<tr>
<td>● Using Mathematics and Computational Thinking</td>
<td></td>
</tr>
<tr>
<td>● Constructing Explanations and Designing Solutions Obtaining, Evaluating, and Communicating Information</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISTE Standards</th>
<th>1.1 Empowered Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td>c. Students use technology to seek feedback that informs and improves their practice and to demonstrate their learning in a variety of ways.</td>
<td></td>
</tr>
<tr>
<td>d. Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.4 Innovative Designer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b. Students select and use digital tools to plan and manage a design process that considers design constraints and calculated risks</td>
<td></td>
</tr>
<tr>
<td>c. Students develop, test and refine prototypes as part of a cyclical design process.</td>
<td></td>
</tr>
<tr>
<td>d. Students exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.5 Computational Thinker</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Students formulate problem definitions suited for technology-assisted methods such as data analysis, abstract models and algorithmic thinking in exploring and finding solutions.</td>
<td></td>
</tr>
<tr>
<td>c. Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.</td>
<td></td>
</tr>
<tr>
<td>d. Students understand how automation works and use algorithmic thinking to develop a sequence of steps to create and test automated solutions.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.7 Global Collaborator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c. Students contribute constructively to project teams, assuming various roles and responsibilities to work effectively toward a common goal.</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Standards</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>Ohio</td>
<td>Standards</td>
</tr>
</tbody>
</table>
| Indiana | Science Standards | Science and Engineering Process Standards (SEPS)  
SEPS.1 Posing questions (for science) and defining problems (for engineering)  
SEPS.2 Developing and using models and tools  
SEPS.5 Using mathematics and computational thinking  
SEPS.6 Constructing explanations (for science) and designing solutions (for engineering)  
SEPS.8 Obtaining, evaluating, and communicating information |
| West Virginia | Science Standards | S.HS.ETS.2 Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering. |
References

https://www.in.gov/doe/students/indiana-academic-standards/science-and-computer-science-2016-2010/

https://www.iste.org/standards/iste-standards-for-students


West Virginia Department of Education. (2015). *Policy 2520.3C: Next Generation Content Standards and Objectives for Science in West Virginia Schools.*

Authors (in alphabetical order)
Anna DeJarnette is an Associate Professor at the University of Cincinnati, School of Education, Cincinnati, OH 45221.

Anna E. Hutchinson is a postdoctoral fellow and program coordinator at the University of Cincinnati, Department of Biological Sciences, Cincinnati, OH, 45221.

John E. Layne is an Associate Professor at the University of Cincinnati, Department of Biological Sciences, Cincinnati, OH, 45221.

Erin Milligan is an Education Consultant at Hamilton County Educational Service Center, Instructional Services, Cincinnati, OH, 45231.

Stephanie M. Rollmann is a Professor at the University of Cincinnati, Department of Biological Sciences, Cincinnati, OH, 45221.

Dieter Vanderelst is an Assistant Professor at the University of Cincinnati, Departments of Biological Sciences, Electrical Engineering, Mechanical Engineering, and Psychology, Cincinnati, OH, 45221.
Contributor, Supporting Program, Acknowledgements, and Classroom Testing
We thank the teacher and student participants in the University of Cincinnati (UC) Biology meets Engineering summer program for their feedback. We thank Gloria Ononye and Valecia Kelly for discussions of instructional frameworks. This material is based upon work supported by the National Science Foundation ITEST grant number 1759150 to SMR, JEL, AD, and DV.