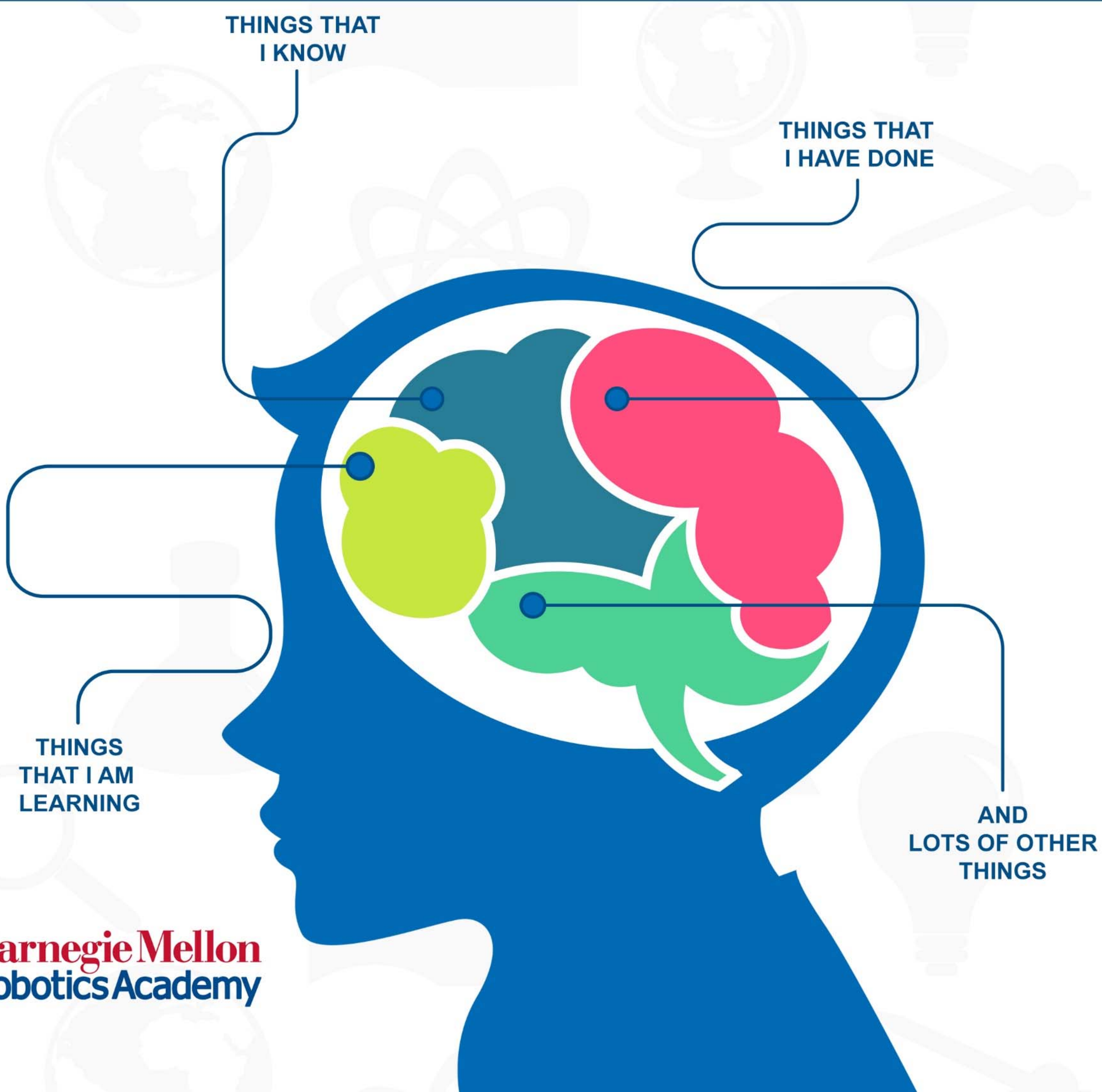
HELPING STUDENTS BUILD CONCEPTUAL MODELS - the Lost Manual

CHANGING CULTURE IN ROBOTICS CLASSROOMS - NSF DRK-12 1418199



Carnegie Mellon Robotics Academy

Helping Students Build Conceptual Models – the Lost Manual

Carnegie Mellon Robotics Academy
Pittsburgh, Pa.
December, 2016

Authors

Robin Shoop, Jesse Flot **Carnegie Mellon Robotics Academy**
Jason McKenna **Robomatter Incorporated**

Acknowledgments

Our work would not be nearly as good without the support, direction, and ideas that are spawned by Chris Schunn and Mary Kay Stein's team from the University of Pittsburgh's Learning Research and Development Center – Chris has guided our collaborative work from the University of Pittsburgh side since 2006 when CMU and PITT began their working together. Ross Higashi and Eben Witherspoon are current doctorial students at PITT and have been have been instrumental in the development of the lessons and teacher materials as well as working closely with our teacher partners.

Thank you teachers from the North Allegheny School District who have been instrumental in helping our team to test the new curriculum modules in their classrooms: Jenn Smoller, Josh Krawchyk, Heath Lauster, Matt Kohler, Nick Vorrasi, Greg Waslo, and Steve Zurbach. They have been instrumental in providing feedback on the lessons, identifying classroom issues, and helping the team to collect data.

Vu Nguyen is the lead programmer for the Computer Science STEM Network (CS2N). His work on the development of CS2N enables the team to collect and aggregate large amounts of student data quickly and efficiently.

Krishna Pandravada for his art direction.

Robomatter Incorporated, VEX Robotics, and the National Science Foundation – who provide funding for this work.

This material is based on work supported by the National Science Foundation under DRK-12 research, Award Number 1418199, *Changing Culture in Robotics Classrooms. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or any collaborator or partner named herein.*

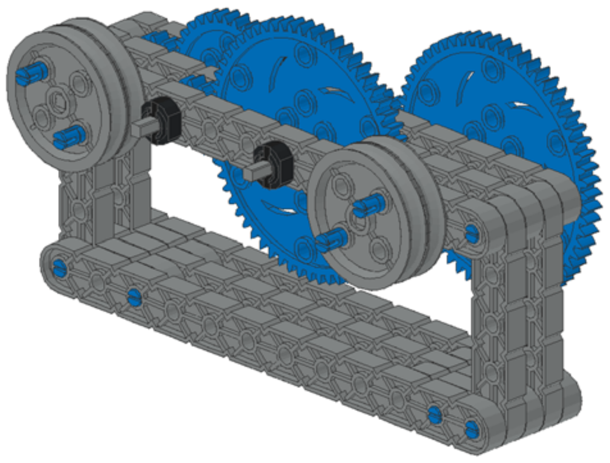
Suggested Citation

Flot, J., McKenna, J., Shoop,R. (2016). *Helping Students Build Conceptual Models – the Lost Manual.* Carnegie Mellon Robotics Academy, Pittsburgh, PA. Retrieved from: http://www.cs2n.org/teachers/research

Helping Students Build Conceptual Models – the Lost Manual

Have you ever taught students how to program a robot and then when you challenged them to program something just a little different they get stuck? The Lost Manual is part one of a series of articles that describe how you can help students develop a conceptual model of what computing is that they can build on as they learn programming. This approach moves teachers away from teaching students to memorize code snippets and reserved words, *to placing the responsibility for learning on the student as they build a conceptual framework that enables them to understand how computers make decisions*. After reading these articles, we hope that you test the lessons and give us feedback. Our goal is to build better tools for teachers. Please send your comments to rshoop@CMU.edu .

Constructivism – Building on an Approach That You Already Use!


Picture 1 – Student build compound gear system

Constructivism is a theory of learning which claims that people construct knowledge through their experiences and interactions with the world rather than by merely receiving and storing knowledge transmitted by the teacher[i]. Robotics teachers use constructivist lesson design regularly. For example, when robotics teachers teach mechanical advantage, a physics principle, they often have students build and conduct experiments using compound gears (picture 1), or they use the example of a 10-speed bicycle's gear train in order to make connections between what the student already understands and the new concept—mechanical advantage. When students build and play with the mechanical system they begin to create a mental model of what the term mechanical advantage means. Then, when the teacher introduces the formula to calculate mechanical advantage, the student discovers the proportional relationships that exist between the associated gears and the concept of mechanical advantage. This constructivist approach begins by helping students develop a mental model of an abstract concept, is followed up with contextual examples, and then is formalized through the introduction of the academic theory behind the concept. The mechanical advantage lesson example provides students with the type of scaffolding that lead to deeper understanding and with anchors that provide them with the ability to recreate meaning later when they need to designing a robot to lift or pull a heavy load.

Constructivists believe that the learner must be actively engaged in the learning process and places the responsibility for learning on the learner[ii] *and that learning is an active process in which learners construct meaning by linking new ideas with their existing knowledge*[iii]*.*

This project began with the following question - *Can we develop a set of teaching tools that not only teach students how to program robots, but also shows measurable student gains in their ability to think computationally*[iv]*?* The methods that we've developed uses Model Eliciting Activities[v] that enable students to build a mental model of what programming and computing looks like before they are tasked to actually program their robot. And finally, we place the activities into the Computational Thinking Practice[vi] framework (Table 1) enabling teachers to foreground computer science understandings, practices, and terminology in a robotics context.

Table 1. Computational Thinking Practices Framework

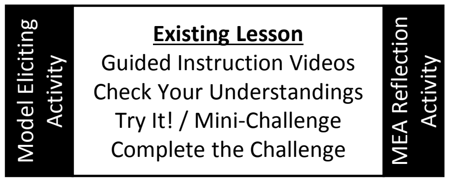
1. Analyzing the effects of developments in computing
2. Analyzing their computational work and the work of others
3. Designing and apply abstractions and models
4. Designing and implement creative solutions and artifacts
5. Communicating thought processes and results
6. Collaborating with peers on computational activities

Teaching Programming Concepts Rather Than Robot Specific Programming Solutions

Research shows that "memory and organization are not only correlated, but organization is a necessary condition for memory."[vii] One of the issues that needs to be addressed for novice programmers is that they have no effective model of a computer;[viii] that is how a computer stores and processes information. An *effective model* is a cognitive structure that connects prior knowledge and experience to new knowledge. *Our goal is to create lessons that enable students to build mental models where they can process, analyze, store, and build new understandings*. Lessons need to be scaffolded, and designed so that new programmers learn that programming is built on a common conceptual foundation that involves how computers make decisions. These types of lessons will enable students to gain a generalizable understanding of computing as opposed to a hardware/software specific understanding of how to program a robot.

Bookends

We are testing this approach using our existing curriculum (which is posted online). Each change is designed to immediately engage students in self-directed problem solving activities where they build conceptual models and identify their own learning targets. Each updated lesson begins with a Model Eliciting Activity, is followed by the existing curriculum module, and then has a reflection activity, see Picture 2. All problems are solved collaboratively in teams and require students to document and communicate what they have learned; this lesson design aligns with features found in both Computational Thinking Practices and Model Eliciting Activities. Concurrently, students are asked to develop a list of things that they don't know; these topics provide the student with a set of new self-defined learning targets. The reflection portion of the activity provides additional opportunities for learning by enabling students to make sure that they had all of their student defined learning targets answered as well as new topics that emerge as they complete the lesson. The goal of each lesson is to place on onus of learning on the student and to enable them to create conceptual models that allow them to rectify, synthesize, and categorize new information as it emerges similar to how students learned in the compound gear ratio lesson example.



Picture 2 – Bookend Format

These lessons are designed to complement the Introduction to Programming VEX IQ curriculum found for free at the Robotics Academy website. The lessons are being tested with over 500 middle school students. The test uses VEX IQ robots and ROBOTC Graphical programming language.

The First Lesson – The Lost Manual

When you tell novice programmer that they are going to "program a robot" what do you think that they envision? Have they seen the programming interface (API)? Do they know how to navigate the API? Do they know about syntax? What is a robot? I imagine that some kids are excited, but have no idea what to expect. This first lesson, the Lost Manual, is designed to provide novice programmers with a mental model of what it means to "program a robot". This model sets the stage for the rest of the lesson. The next couple of pages include a set of teacher notes and handouts that enable you to try out and test "the Lost Manual" lesson. The handouts are designed to be used with VEX IQ and ROBOTC Graphical, but could be easily be modified for LEGO.

The next article will highlight how to implement the same lesson using ROBOTC text based programming, this lesson is implementable with either VEX IQ or VEX EDR robots.

If you try this lesson and have comments on what worked and what didn't please let us know.

---

[i] Ben-Ari, Mordechai. "Constructivism in computer science education." *Acm sigcse bulletin*. Vol. 30. No. 1. ACM, 1998.
[ii] https://en.wikipedia.org/wiki/Constructivism_(philosophy_of_education)
[iii] Naylor, Stuart, and Brenda Keogh. "Constructivism in classroom: Theory into practice." *Journal of Science Teacher Education* 10.2 (1999): 93-106.
[iv] Grover, Shuchi, and Roy Pea. "Computational Thinking in K–12 A Review of the State of the Field." *Educational Researcher* 42.1 (2013): 38-43.
[v] Lesh, R., Hoover, M., Hole, B., Kelly, A., & Post, T. (2000). Principles for developing thought-revealing activities for students and teachers. In A. E. Kelly & R. Lesh (Eds.), *Handbook of Research Design in Mathematics and Science Education* (pp. 591-646). Mahwah, NJ: Lawrence Erlbaum Associates.
[vi] Bienkowski, M., et al. "Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science: A First Look." *SRI International* (2015).
[vii] Mandler, George. "Organization and memory." *Psychology of learning and motivation* 1 (1967): 327-372.
[viii] Ben-Ari, Mordechai. "Constructivism in computer science education." *Journal of Computers in Mathematics and Science Teaching* 20.1 (2001): 45-74.

SRI Computational Thinking Practice Design Patterns

Table 1. Computational Thinking Practices Framework

1. Analyze the effects of developments in computing
2. Analyze their computational work and the work of others
3. Design and apply abstractions and models
4. Design and implement creative solutions and artifacts
5. Communicate thought processes and results
6. Collaborate with peers on computational activities

Our goal is to encourage robotics teachers to foreground and integrate Computational Thinking Practices (CTP) into their daily classroom practices. This page is designed as a CTP primer for teachers not familiar with CTP. All teachers should read the full paper which can be found at: http://pact.sri.com/resources.html Open up the *Assessment Design Patterns for Computational Thinking Practices* paper. The Assessment Design Patterns for Computational Thinking Practices shown below provide a very small glimpse into the actual design patterns. These partial design patterns were taken directly from the SRI website which is referenced above.

1. Analyze the effects of developments in computing – This design pattern askes student to recognize aspects of computers and computing. They will show an understanding of how computing has enabled innovations in various disciplines and in society as a whole and at the same time has given rise to ethical (e.g., privacy) and social justice (e.g., equal access) issues. They will also demonstrate a broad understanding of "intelligent" machines and the idea of networked systems.
2. Analyze their computational work and the work of others – This design pattern supports the development of tasks in which students demonstrate that they can evaluate computational work (resulting in artifacts such as a program, program outputs, a website, or problem solution) and compare multiple computational artifacts. Students are able to recognize how different techniques can be used to solve problems or achieve computational goals in different ways.
3. Design and apply abstractions and models – This design pattern supports the development of tasks in which students use ideas and representations that capture general to specific aspects, or patterns, of an entity or a process and the relationships/structures among entities or processes, including level of detail. This may include designing general solutions to problems or generalizing a specific solution to encompass a broader class of problems (functional abstraction).
4. Design and implement creative solutions and artifacts – This design pattern supports the development of tasks in which students translate novel ideas and problem solutions into computational solutions and artifacts. This design pattern encompasses steps of both problem solving and creative processes, including understanding, decomposing, exploring (e.g., by creating different representations of the problem with storyboards, flowcharts, and pseudocode), creating products that show one or more designed solutions and/or artifacts, and testing and improving the solution and or artifact.
5. Communicate thought processes and results - Communicating about computational artifacts supports many phases of computational thinking. This design pattern supports the development of tasks in which students show that they can communicate the process and results of their work in a way that is appropriate for the particular audience. Students can articulate major themes and ideas related to computing in writing and orally supported by graphs, visualizations, and computational analysis.
6. Collaborate with peers on computational activities - Collaborative problem-solving or collaborative design competency is the capacity of an individual to effectively engage in a process whereby two or more agents attempt to solve a problem or design an artifact by sharing the understanding and effort required to come to a solution/design and pooling their knowledge, skills and efforts to reach that solution/design.

The Lost Manual – Teacher Notes

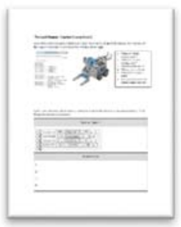
Introduction – The goal of this mini lesson is to enable students to develop a mental model of what it means to program a VEX IQ robot using ROBOTC's Graphical language. At the end of the mini lesson students should be familiar with: the graphical code, elements within the code, the configuration of the robot, and the ROBOTC Graphical user interface.

The Problem – The Robotics Club has sample code, but doesn't have the ROBOTC Graphical User Manual. Your job is to look at sample code and interpret what it does. Circle and note things that you don't understand. Your team is responsible begin to develop a ROBOTC "Getting Started" manual based on your interpretation and testing of the code.

Steps to the project

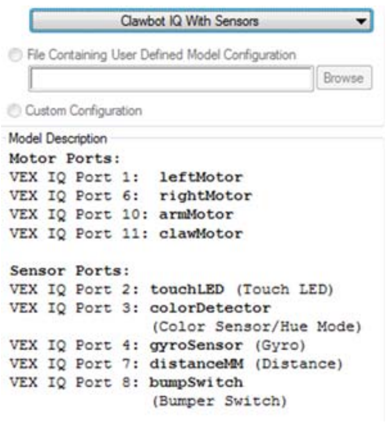
1. Print the handouts that go with this lesson.
2. Present "The Problem" to your students. *The robotics club has…* - five minutes.
3. Pass out the handouts. One is a picture of a Clawbot robot and code, the others are examples of ROBOTC Graphical code. Assign students to work in pairs and write down what they think that the code does, have them circle anything that they don't understand. – One period.
4. Class/Group discussion – Have students review the handouts one handout at a time. This activity can be done as a class or in teams. Students should review the code one line at a time to make sure that they understand all parts of the code.
5. Assign students to – Upload the code to the robot and test it to see if the robot did what they thought that it would do.
   Note: Open the MEA folder in the VEX IQ Sample Programs to find the code.
6. Assign students to - Update their documentation if the robot did something that you didn't expect that it would do.
7. Work as a team to develop a digital version of a "ROBOTC Graphical "Getting Started" programming guide.
8. Begin the Basic Movement lesson found in Introduction to Programming VEX IQ
9. Complete the Basic Movement lesson and review the unit. *Note: students do not need to complete the full "Orchard Challenge". Once your students get a general feel for accurately moving straight and turning have the students move on to Sensors.*

Handouts Pictured directly below are the lesson handouts.



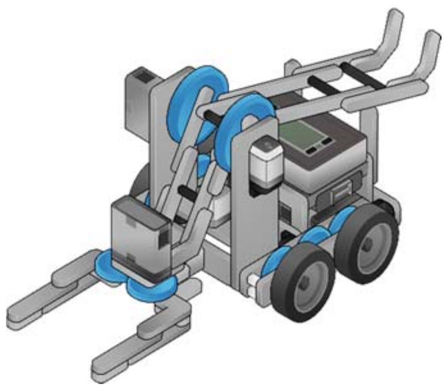
The Lost Manual: Clawbot Source Code 1

Look at the table and picture below and describe what you think that it means. On the back of the paper answer the questions in the text box at the right.



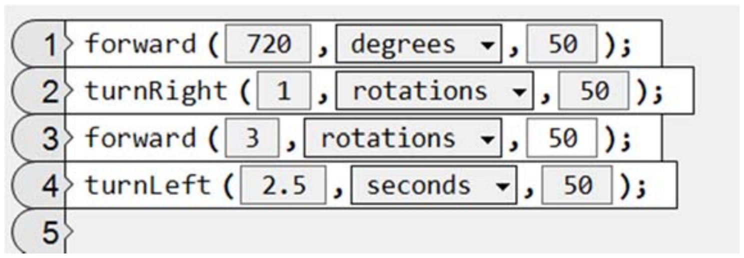
```
Motor Ports:
VEX IQ Port 1:  leftMotor
VEX IQ Port 6:  rightMotor
VEX IQ Port 10: armMotor
VEX IQ Port 11: clawMotor

Sensor Ports:
VEX IQ Port 2: touchLED (Touch LED)
VEX IQ Port 3: colorDetector
               (Color Sensor/Hue Mode)
VEX IQ Port 4: gyroSensor (Gyro)
VEX IQ Port 7: distanceMM (Distance)
VEX IQ Port 8: bumpSwitch
               (Bumper Switch)
```



1. What is a "Model Configuration"?
2. What is a "Custom Configuration?"
3. What are Motor Ports?
4. What are Sensor Ports?
5. What do the numbers mean?
6. Can you identify what the robot's sensors look like?

Look at the code below. Write what you think that each line of code does in the pseudocode box. Circle things that you don't understand.

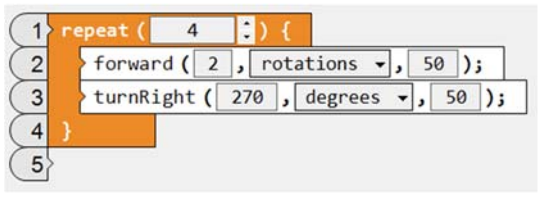
Source Code 1

Pseudocode
1 2 3 4

```
1 forward ( 720 , degrees , 50 );
2 turnRight ( 1 , rotations , 50 );
3 forward ( 3 , rotations , 50 );
4 turnLeft ( 2.5 , seconds , 50 );
5
```

The Lost Manual - Clawbot Source Code 2

Look at the code below. Write what you think that each line of code does in the pseudocode box. Circle things that you don't understand. Answer the questions below.

Source Code 2



```
1  repeat ( 4 ) {
2      forward ( 2 , rotations , 50 );
3      turnRight ( 270 , degrees , 50 );
4  }
5
```

Pseudocode

1

2

3

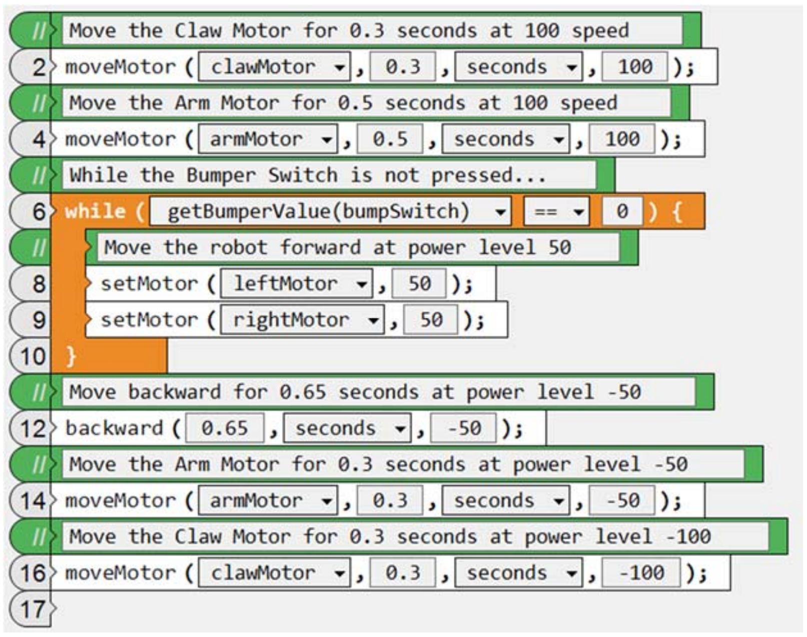
4

1. What is the difference between rotations and degrees?

2. What will this code make the robot do?

The Lost Manual - Clawbot Source Code 3

Look at the code below. Write what you think that each line of code does below the code. Circle things that you don't understand.



```
//  Move the Claw Motor for 0.3 seconds at 100 speed
2   moveMotor ( clawMotor , 0.3 , seconds , 100 );
//  Move the Arm Motor for 0.5 seconds at 100 speed
4   moveMotor ( armMotor , 0.5 , seconds , 100 );
//  While the Bumper Switch is not pressed...
6   while ( getBumperValue(bumpSwitch) == 0 ) {
//      Move the robot forward at power level 50
8       setMotor ( leftMotor , 50 );
9       setMotor ( rightMotor , 50 );
10  }
//  Move backward for 0.65 seconds at power level -50
12  backward ( 0.65 , seconds , -50 );
//  Move the Arm Motor for 0.3 seconds at power level -50
14  moveMotor ( armMotor , 0.3 , seconds , -50 );
//  Move the Claw Motor for 0.3 seconds at power level -100
16  moveMotor ( clawMotor , 0.3 , seconds , -100 );
17
```

1. Why are the blocks different colors?
2. Why did some of the numbers at the left disappear and were replaced by "//"?