Table of Contents

Welcome to Computer Science Fundamentals

Welcome to Computer Science Fundamentals, the Code.org curriculum designed for students in kindergarten through fifth grade (K-5), which includes students 5-11 years old! This guide has been created to help you navigate the lessons in Courses A-F. It begins with an introduction to the CS Fundamentals curriculum, provides a look into our core values and methods, and includes a detailed overview of each course offering. You will also find customized implementation solutions for many different classroom situations.

Who is CS Fundamentals for?

CS Fundamentals was built with elementary school educators and students in mind. Courses A-F have been specifically tailored to students in K-5, and no prior experience is assumed. These courses work best in a classroom setting, led by an engaged teacher. For scenarios where students will be learning independently or asynchronously, we recommend our CS Fundamentals Express courses. The lessons in CS Fundamentals are presented with the understanding that many teachers will not have any previous computer science experience, and educators are therefore encouraged to learn along with their students.

Which course is right for my students?

CS Fundamentals Courses A-F

CS Fundamentals is made up of 6 courses — one course for each grade, K-5. This grade alignment allows for the most robust content along the entire elementary pipeline, while also allowing for students and teachers to enter the pathway at any point.

Course A *Kindergarten*	**Course B** *1st Grade*	**Course C** *2nd Grade*	**Course D** *3rd Grade*	**Course E** *4th Grade*	**Course F** *5th Grade*
12 lessons, ~12 hours to complete	*12 lessons, ~12 hours to complete*	*18 lessons, ~18 hours to complete*	*19 lessons, ~19 hours to complete*	*19 lessons, ~19-21 hours to complete*	*19 lessons, ~19-21 hours to complete*
Concepts					
• *Digital Citizenship* • *Sequencing* • *Loops* • *Events*	• *Digital Citizenship* • *Sequencing* • *Loops* • *Impacts of Computing* • *Events*	• *Digital Citizenship* • *Sequencing* • *Binary* • *Loops* • *Events* • *Data*	• *Sequencing* • *Events* • *Loops* • *Conditionals* • *Binary* • *Digital Citizenship*	• *Sprites* • *Digital Citizenship* • *Nested Loops* • *Functions* • *Impacts of Computing*	• *Digital Citizenship* • *Variables* • *Data* • *For Loops* • *Internet* • *Sprites*

All courses make suitable entry points for students. Courses E and F feature “ramp up” lessons which are intended to introduce or review important concepts from previous courses at an accelerated pace.

CS Fundamentals Express Courses

In addition to courses A-F, CS Fundamentals also offers two “express courses”, which are designed for situations where the teacher is a less active role in engaging students. An express course might be used, for example, if a student is learning CS on her own.

Express courses do not have unplugged lessons (lessons that do not use a computer), and instead focus on covering the content from their A-F counterparts in a way that can be delivered without a teacher. The table below maps courses A-F and the two express courses:

Course A *Kindergarten*	**Course B** *1st Grade*	**Course C** *2nd Grade*	**Course D** *3rd Grade*	**Course E** *4th Grade*	**Course F** *5th Grade*
Pre-reader Express Course Built with lessons from Courses A - B		**Express Course** Built with lessons from Courses C - F			

Standards Mapping

CS Fundamentals was written using both the K–12 Computer Science Framework (k12cs.org) and the 2017 Computer Science Teachers Association (CSTA) standards as guidance. Courses have opportunities to connect to Common Core and NGSS standards. Details can be found at curriculum.code.org/csf/standards.

Assessments

At Code.org, we believe that you know your students best, which is why we do not attempt to automatically determine what “grade” students should receive for any given lesson. Instead, we try to build tools that allow you to easily see student progress and to identify evidence of learning. The ability to see where a student is succeeding and where they need help is fundamental to providing the opportunity to tailor their learning experience. For that reason, our teacher dashboard is continually evolving to better highlight the work done by your class sections. Keep an eye on Code.org Support (https://support.code.org/) for more information on changes and improvements.

Please note, we have provided assessment worksheets with most unplugged lessons and “assessment” puzzles for many online lessons.

Code.org Values and Philosophy

Curriculum Values

While Code.org offers a wide range of curricular materials across a wide range of ages, the following values permeate and drive the creation of every lesson we write.

Computer Science is Foundational for Every Student

We believe that computing is so fundamental to understanding and participating in society that it is valuable for every student to learn as part of a modern education. We see computer science as a liberal art, a subject that provides students with a critical lens for interpreting the world around them. Computer science prepares all students to be active and informed contributors to our increasingly technological society whether they pursue careers in technology or not. Computer science can be life-changing, not just skill training.

Teachers in Classrooms

We believe students learn best with the help of an empowered teacher. We design our materials for a classroom setting and provide teachers robust supports that enable them to understand and perform their critical role in supporting student learning. Because teachers know their students best, we empower them to make choices within the curriculum, even as we recommend and support a variety of pedagogical approaches. Knowing that many of our teachers are new to computer science themselves, our resources and strategies specifically target their needs.

Student Engagement and Learning

We believe that students learn best when they are intrinsically motivated. We prioritize learning experiences that are active, relevant to students' lives, and provide students authentic choice. We encourage students to be curious, solve personally relevant problems and to express themselves through creation. Learning is an inherently social activity, so we interweave lessons with discussions, presentations, peer feedback, and shared reflections. As students proceed through our pathway, we increasingly shift responsibility to students to formulate their own questions, develop their own solutions, and critique their own work.

Equity

We believe that acknowledging and shining a light on the historical inequities within the field of computer science is critical to reaching our goal of bringing computer science to all students. We provide tools and strategies to help teachers understand and address well-known equity gaps within the field. We recognize that some students and classrooms need more support than others, and those with the greatest needs should be prioritized. All students can succeed in computer science when given the right support and opportunities, regardless of prior knowledge or privilege. We actively seek to eliminate and discredit stereotypes that plague computer science and alienate the very students we aim to reach.

Curriculum as a Service

We believe that curriculum is a service, not just a product. Along with producing high quality materials, we seek to build and nourish communities of teachers by providing support and channels for communication and feedback. Our products and materials are not static entities, but a living and breathing body of work that is responsive to feedback and changing conditions. To ensure ubiquitous access to our curriculum and tools, they are web-based and cross-platform, and will forever be free to use and openly licensed under a Creative Commons license.

Pedagogical Approach To Our Values

When we design learning experiences, we draw from a variety of teaching and learning strategies all with the goal of constructing an equitable and engaging learning environment.

Role of the Teacher

We design curriculum with the idea that the instructor will act as the lead learner. As the lead learner, the role of the teacher shifts from being the source of knowledge to being a leader in seeking knowledge. The lead learner's mantra is: "I may not know the answer, but I know that together we can figure it out." A very practical result of this is that we rarely ask a teacher to lecture or offer the first explanation of a CS concept. We want the class activity to do the work of exposing the concept to students, allowing the teacher to shape meaning from what the students have experienced. We also expect teachers to act as the curator of materials. Finally, we include an abundance of materials and teaching strategies in our curricula - sometimes too many to use at once - with the expectation that teachers have the professional expertise to determine how to best conduct an engaging and relevant class for their own students.

Discovery and Inquiry

We take great care to design learning experiences in which students have an active and equal stake in the proceedings. Students are given opportunities to explore concepts and build their own understandings through a variety of physical activities and online lessons. These activities form a set of common lived experiences that connect students (and the teacher) to the course content and to each other. The goal is to develop a common foundation upon which all students in the class can construct their understanding of computer science concepts, regardless of prior experience in the discipline.

Materials and Tools

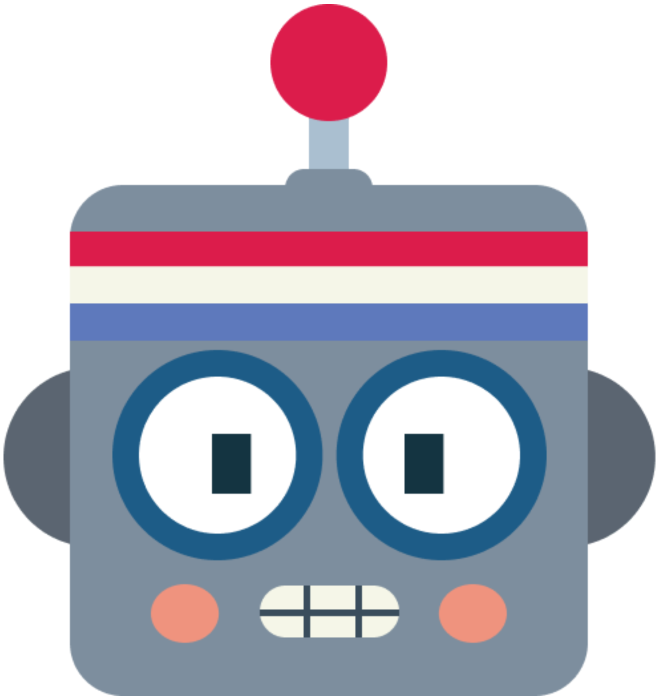
Our materials and tools are specifically created for learners and learning experiences. They focus on foundational concepts that allow them to stand the test of time, and they are designed to support exploration and discovery by those without computer science knowledge. This allows students to develop an understanding of these concepts through "play" and experimentation. From our coding environments to our non-coding tools and videos, our resources have been engineered to support the lessons in our curriculum, and thus our philosophy about student engagement and learning. In that vein, our videos can be a great tool for sensemaking about CS concepts and provide a resource for students to return to when they want to refresh their knowledge. They are packed with information and "star" a diverse cast of presenters and CS role models.

Creation and Personal Expression

Many of the projects, assignments, and activities in our curriculum ask students to be creative, to express themselves, and then to share their creations with others. While certain lessons focus on learning and practicing new skills, our goal is always to enable students to transfer these skills to creations of their own. Everyone seeks to make their mark on society, including our students, and we want to give them the tools they need to do so. When computer science provides an outlet for personal expression and creativity, students are intrinsically motivated to deepen the understandings that will allow them to express their views and carve out their place in the world.

The Classroom Community

Whether learners are simply conferring with a partner during a warm up discussion, or engaging in a long-term group project, our belief is that a classroom where students are communicating, solving problems, and creating things is a classroom that not only leads to active and better learning for students, but also leads to a more inclusive culture in which all students share ideas and listen to ideas of others. For example, classroom discussions usually follow a Think-Pair-Share pattern; we ask students to write computer code in pairs.



CS Fundamentals Curriculum Overview

At the highest level, each CS Fundamentals course is organized into **concept chunks**.

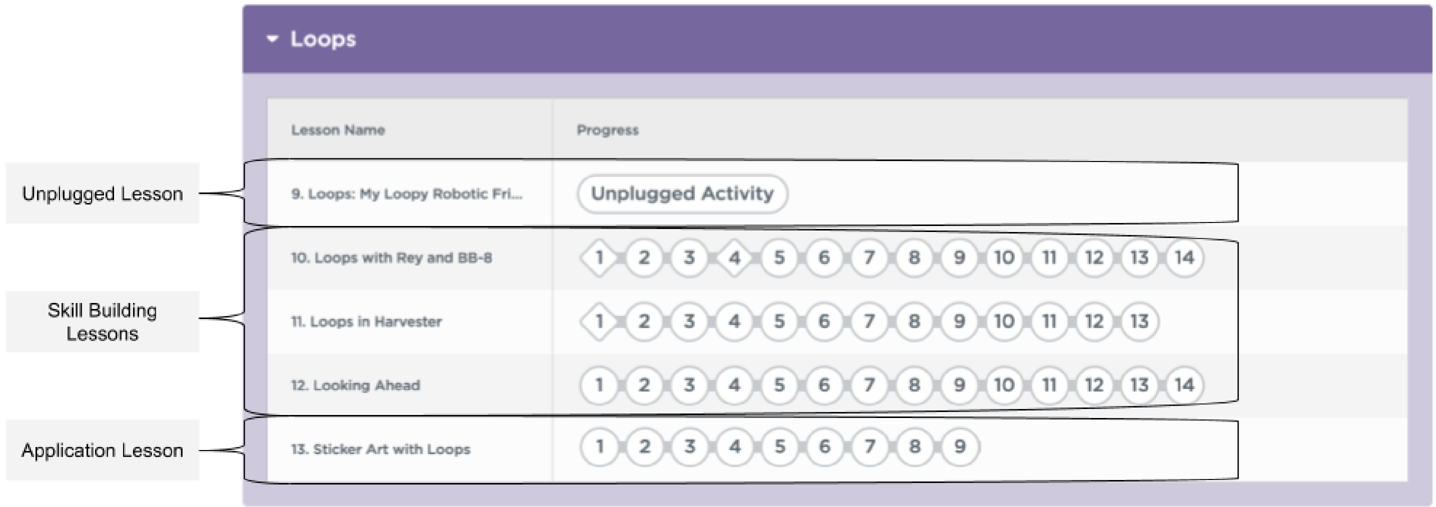
> Concept chunks are the big ideas that provide structure to a course. Concept chunks make it easy for teachers to see at a glance what is covered in a course.

Each CS Fundamentals course has between 4 and 6 concept chunks, and each chunk can have one or several lessons. In most cases, concept chunks begin with a computer-free lesson called an *Unplugged Lesson*, which is meant to introduce a concept before students engage with programming lessons. Programming lessons require students to use computers and consist of *Skill Building*, *Application*, and *End of Course Project* lesson types.

Each course is composed of the following concept chunks:

Course A	**Course B**	**Course C**	**Course D**	**Course E**	**Course F**
Kindergarten	*1st Grade*	*2nd Grade*	*3rd Grade*	*4th Grade*	*5th Grade*
• *Digital Citizenship* • *Sequencing* • *Loops* • *Events*	• *Digital Citizenship* • *Sequencing* • *Loops* • *Impacts of Computing* • *Events*	• *Digital Citizenship* • *Sequencing* • *Binary* • *Loops* • *Events* • *Data*	• *Sequencing* • *Events* • *Loops* • *Conditionals* • *Binary* • *Digital Citizenship*	• *Sprites* • *Digital Citizenship* • *Nested Loops* • *Functions* • *Impacts of Computing*	• *Digital Citizenship* • *Variables* • *Data* • *For Loops* • *Sprites*

To illustrate, below are the five lessons contained in the "Loops" concept chunk in Course C, as they appear on Code Studio.



Loops

Lesson Name	Progress
9. Loops: My Loopy Robotic Fri...	Unplugged Activity
10. Loops with Rey and BB-8	1 2 3 4 5 6 7 8 9 10 11 12 13 14
11. Loops in Harvester	1 2 3 4 5 6 7 8 9 10 11 12 13
12. Looking Ahead	1 2 3 4 5 6 7 8 9 10 11 12 13 14
13. Sticker Art with Loops	1 2 3 4 5 6 7 8 9

You will learn more about how the curriculum is organized in the 'Course Resources' section below.

The following pages provide an overview of each of the 6 courses in the CS Fundamentals curriculum. For each course, there is one page giving an overview, describing the core concepts, attitudinal goals, and teaching tips. Then, there is one page that follows, outlining each lesson of the course.

Course A

Overview

Course A offers a computer science curriculum for beginning readers in early elementary grades. Students will learn to program using commands like loops and events. The lessons featured in this course also teach students to collaborate with others meaningfully, investigate different problem-solving techniques, persist in the face of difficult tasks, and learn about internet safety. By the end of this course, students create their very own custom game in the Play Lab programming environment on Code.org.

Core concepts:

- Digital Citizenship
- Sequencing
- Loops
- Events

Attitudinal goals:

- Programming is fun.
- It's okay not to get it right the first time.
- I can solve problems if I keep trying.

Key teaching tips:

- Use the stories as a read-aloud and discuss the scenarios as a class.
- Use pair programming and encourage students to help each other.
- Work through sample problems with students as a class.
- Connect unplugged lessons to the online lessons using "bridging activities".
- Celebrate persistence as well as successes.
- Remind students that they can go back and fix mistakes.
- Honor the humor in the lessons and add more where possible.

Course A: Lesson Outlines

Online lessons are in regular text and unplugged lessons are **bolded**.

Concept Chunk	#	Lesson Name	Description
Digital Citizenship	**1**	**Safety in My Online Neighborhood**	Created by Common Sense Education, students practice staying safe while exploring online.
Sequencing	2	Learn to Drag and Drop	The main goal of this lesson is to build students' experience with computers. By covering the most basic computer functions such as clicking, dragging, and dropping, we are creating a more equal playing field in the class for future puzzles.
	3	**Happy Maps**	This activity will help students gain experience reading and writing in shorthand code.
	4	Sequencing with Scrat	This lesson begins with a brief discussion on computer lab manners, then students progress into using a computer to complete online puzzles.
	5	Programming with Scrat	In this set of online puzzles, students will build on the understanding of algorithms, debugging, and general computer literacy.
	6	Programming with Rey and BB-8	In this lesson, students will use their newfound programming skills in more complicated ways to navigate a tricky course with BB-8.
Loops	**7**	**Happy Loops**	Students learn about an easier way to solve problems using loops.
	8	Loops with Scrat	This lesson builds on the idea of using loops in a program.
	9	Loops with Laurel	Continuing practice with loops, students will help Laurel the Adventurer collect treasure.
	10	Ocean Scene with Loops	Here, students use loops to create patterns. At the end of this lesson, students will be given the opportunity to create their own images using loops.
Events	**11**	**The Big Event Jr.**	This lesson demonstrates that events are a great way to add variety to a sequential algorithm.
	12	On the Move with Events	Students will have the opportunity to learn how to use events in Play Lab and apply their coding skills to create an animated game.

Course B

Overview
Course B was developed with first graders in mind. Tailored to a novice reading level, this course also assumes limited knowledge of shapes and numbers.

While the concepts in Course B parallel those in Course A, students will be exposed to more sophisticated unplugged lessons and a greater variety of puzzles. Students will learn the basics of programming, collaboration techniques, investigation and critical thinking skills, persistence in the face of difficulty, and internet safety. At the end of this course students will create their very own custom game in the Play Lab programming environment on Code.org.

Core concepts:

- Digital Citizenship
- Sequencing
- Loops
- Impacts of Computing
- Events

Attitudinal goals:

- Programming is fun.
- It's okay not to get it right the first time.
- I can solve problems if I keep trying.

Key teaching tips:

- Use pair programming and encourage students to help each other.
- Work through sample problems with students as a class.
- Connect unplugged lessons to the online lessons using "bridging activities".
- Celebrate persistence as well as successes.
- Remind students that they can go back and fix mistakes.
- Honor the humor in the lessons and add more wherever possible.

Course B: Lesson Outlines

Online lessons are in regular text and unplugged lessons are **bolded**.

Concept Chunk	#	Lesson Name	Description
Digital Citizenship	**1**	**Digital Trails**	Created by Common Sense Education, students will learn that the information they put online leaves a digital footprint or "trail."
Sequencing	**2**	**Move It, Move It**	This lesson mentally prepares students for the coding exercises that they will encounter over the length of this course.
	3	Sequencing with Angry Birds	This lesson begins with a brief discussion on computer lab manners, then will progress into using a computer to complete online puzzles.
	4	Programming with Angry Birds	In this set of online puzzles, students will build on the understanding of algorithms, debugging, and general computer literacy.
	5	Programming with Harvester	Students will apply the programming concepts that they have learned to the Harvester environment.
Loops	**6**	**Getting Loopy**	Students will dance their way to a better understanding of how to use repeat loops.
	7	Loops with Harvester	Building on the concept of repeating instructions, this lesson will have students using loops to more efficiently get to the veggies.
	8	Loops with Laurel	Students use loops to collect treasure more efficiently.
	9	Drawing Gardens with Loops	Here, students use loops to create patterns. At the end of this stage, students will be given the opportunity to create their own images using loops.
Impacts of Computing	**10**	**The Right App**	Students exercise empathy and creativity to sketch their own smartphone app that addresses the needs of an imaginary user.
Events	**11**	**The Big Event Jr.**	This lesson shows that events are a great way to add variety to a pre-written algorithm.
	12	A Royal Battle with Events	In this online activity, students will have the opportunity to learn how to use events in Play Lab and apply all of the coding skills that they've learned to create an animated game.

Course C

Overview
Course C was developed for students in and around the second grade. Lessons in this course may assume a limited understanding of shapes and elementary math concepts.

Students will create programs with sequencing, loops, and events. They will translate their initials into binary, investigate problem-solving techniques, and develop strategies for building positive communities both online and off. By the end of the course, students will create interactive games that they can share. Each concept in Course C is taught from the beginning, graduating toward experiences that allow for growth and creativity to provide all students a rich and novel programming experience.

Core concepts:

- Digital Citizenship
- Sequencing
- Binary
- Loops
- Events
- Data

Attitudinal goals:

- I can read code and predict the outcome.
- Programming can make repetitive tasks easy.

Key teaching tips:

- Talk with students before you begin about how they may experience frustration.
- Use pair programming and encourage students to help each other.
- Provide lesson examples to set students off on the right foot.
- Connect unplugged lessons to the online lessons using “bridging activities”.
- Celebrate persistence as well as successes.
- Remind students that they can go back and fix mistakes.

Course C: Lesson Outlines

Online lessons are in regular text and unplugged lessons are **bolded**.

Concept Chunk	#	Lesson Name	Description
Digital Citizenship	1	**Putting a STOP to Online Meanness**	Created by Common Sense Education, students learn about meanness and what to do if they encounter it online.
	2	**Password Power-Up**	Created by Common Sense Education, students learn about how strong passwords can help protect their privacy.
Sequencing	3	**My Robotic Friends Jr.**	This lesson teaches students about the connection between algorithms and programming, as well as the valuable skill of debugging.
	4	Programming with Angry Birds	Students will develop sequential algorithms to move a bird from one side of the maze to reach a pig at the other side.
	5	Debugging in Maze	Students will step through existing code to identify errors, including incorrect loops, missing blocks, extra blocks, and blocks that are out of order.
	6	Collecting Treasure with Laurel	Students continue to develop their understanding of algorithms and debugging by creating sequential algorithms to pick up treasure with Laurel the Adventurer.
	7	Creating Art with Code	This Artist lesson will allow students to create images of increasing complexity using new blocks like `move forward by 100 pixels` and `turn right by 90 degrees.
Binary	8	**Binary Bracelets**	This lesson helps demonstrate how it is possible to take something from real life and translate it into a series of ons and offs.
Loops	9	**My Loopy Robotic Friends Jr.**	Using the language from the first 'My Robotic Friends Jr.' activity, students find that they can build big structures faster using loops.
	10	Loops with Rey and BB-8	Students will use loops to traverse mazes more efficiently than before.
	11	Harvesting Crops with Loops	Students will loop new actions to help the harvester collect multiple veggies growing in large bunches.
	12	Looking Ahead with Minecraft	Students will get the chance to practice ideas that they have learned up to this point, as well as getting a sneak peek at conditionals.
	13	Sticker Art with Loops	This lesson builds on the understanding of loops from previous lessons and gives students a chance to be truly creative.
Events	14	**The Big Event**	Students will learn that events are a great way to make their program interactive.
	15	Build a Flappy Game	In this special stage, students get to build their own Flappy Bird game by using event handlers to detect mouse clicks and object collisions.
	16	Chase Game with Events	It's time to get creative and make a game in Play Lab.
Data	17	**Picturing Data**	Students create visualizations of data to help them reason and predict about what they observe.
Project	18	End of Course Project	Students plan and build a game using Play Lab, totally from scratch.

Course D

Overview
Course D was created for students who read at roughly a third grade level. Angles and mathematical concepts are introduced with helpful videos and hints.

The course begins with a review of the concepts found in Courses A, B, and C. This review can be used to introduce or refresh basic ideas, such as loops and events. Afterward, students will develop their understanding of algorithms, nested loops, while loops, conditionals, and events, as well as learn about digital citizenship. This course is crafted to build a strong foundation of basic concepts before opening up to a wide range of new and exciting topics.

Core concepts:

- Sequencing
- Events
- Loops
- Conditionals
- Binary
- Digital Citizenship

Attitudinal goals:

- Struggle is good and a sign that I’m growing.
- I can read programs and predict their outcomes.
- Programs can be written to make simple choices.

Key teaching tips:

- Talk with students before you begin about how they may experience frustration.
- Use pair programming and encourage students to help each other.
- Provide lesson examples to set students off on the right foot.
- Remind students of the importance of persistence.
- Begin to teach students the importance of solving their own issues.
- Encourage students to use a journal during and after activities.
- Give students the opportunity to share successes.

Course D: Lesson Outlines

Online lessons are in regular text and unplugged lessons are in **bolded** text.

Concept Chunk	#	Lesson Name	Description
Sequencing	**1**	**Graph Paper Programming**	In this lesson, students will program their friends to draw pictures.
	2	Introduction to Online Puzzles	This lesson will give students practice in the skills they will need for this course.
	3	**Relay Programming**	This lesson builds on the previous lessons by introducing teamwork.
	4	Debugging with Laurel	In this lesson, students will learn about the secrets of debugging.
Events	5	Events in Bounce	Students get to make their own video game in this lesson.
	6	Build a Star Wars Game	Students build their own Star Wars game in this lesson.
	7	Dance Party	Time to celebrate! In this lesson, students will program their own interactive dance party.
Loops	8	Loops in Ice Age	As a quick update (or introduction) to using loops, this lesson will have students using the repeat block to get Scrat to the acorn more efficiently.
	9	Drawing Shapes with Loops	In this lesson, loops make it easy for students to make even cooler images with Artist.
	10	Nested Loops in Maze	This lesson will teach students what happens when they place a loop inside another loop.
Conditionals	11	**Conditionals with Cards**	It's time to play a game in which students earn points only under certain conditions.
	12	If/Else with Bee	Now that students understand conditionals, it's time to program Bee to use them when collecting honey and nectar.
	13	While Loops in Farmer	This lesson will teach students about a new kind of loop: while loops.
	14	Until Loops in Maze	Students learn to use until loops in this lesson.
	15	Harvesting with Conditionals	This lesson will help students practice deciding when to use each conditional.
Binary	**16**	**Binary Images**	Students learn how computers store pictures using simple ideas like on and off.
	17	Binary Images with Artist	In this lesson, students will learn how to make images using binary.
Digital Citizenship	**18**	**Be A Super Digital Citizen**	Created by Common Sense Education, students learn how they can be upstanders when they see cyberbullying.
Project	19	End of Course Project	This capstone lesson takes students through the process of designing, developing, and showcasing their own projects!

Course E

Overview
Course E in CS Fundamentals was tailored to the needs of students in the fourth grade.

At this point, students should be growing in their confidence with using basic programming concepts and are ready to start using them to solve more novel problems. Throughout this course, students will learn to identify when to apply and combine the many concepts they've learned in previous courses. Students will begin with some light review, followed by a deep dive into the idea of functions. For many, the lessons in Course E will provide the first puzzles where difficult concepts are mixed together, making it one of the most challenging courses in the series.

Because of the complexity of Course E, it is important to be consistent with expectations from the very beginning. With fourth graders, it is advised that students are encouraged to work together to find solutions rather than relying on help from the teacher or another experienced supervisor. Students should be empowered to try multiple techniques and should be given praise for persistence and for helping others.

Ultimately, Course E will set the foundation for Course F in the fifth grade. This means that it is as critical for students to understand the ideas behind each puzzle as it is for them to successfully solve it. For this reason, you might want to show students how to use peer interaction or journaling to help with difficult puzzles. Mainly, they should be able to ask and answer four questions:

- What does the puzzle want me to do?
- What did I try to make that happen?
- Where did it go wrong?
- What might be the next thing I could try?

Core concepts:

- Sprites
- Digital Citizenship
- Nested Loops
- Functions
- Impacts of Computing

Attitudinal goals:

- There are often many ways to solve a problem.
- Reflecting on past problems helps me solve new ones.
- Programming is creative.

Key teaching tips:

- Talk with students before you begin about how they may experience frustration.
- Use pair programming and encourage students to help each other.
- Require students to make a first attempt at problem solving before asking for help.
- Remind students of the importance of persistence.
- Encourage students to use a journal during and after activities.
- Promote an environment of cross-team collaboration for group activities and projects.

Course E: Lesson Outlines

Online lessons are in regular text and unplugged lessons are **bolded**.

Concept Chunk	#	Lesson Name	Description
Ramp Up	1	Sequencing in Maze	Students will practice sequencing and debugging before adding new skills.
	2	Drawing with Loops	This lesson gets students thinking about why loops are better than longhand.
	3	Conditionals in Minecraft: Voyage Aquatic	Students will get the chance to practice ideas that they have learned up to this point, as well as getting a sneak peek at conditionals.
	4	Conditionals with the Farmer	This lesson introduces students to \`while\` loops and \`if / else\` statements.
Sprites	**5**	**Simon Says**	In this lesson, students will play a game intended to get them thinking about the way commands need to be given to produce the right result. This will help them more easily carry over to the Sprite Lab programming environment in the upcoming lessons.
	6	Swimming Fish with Sprite Lab	In this lesson, students will learn about the two concepts at the heart of Sprite Lab: sprites and behaviors.
	7	Alien Dance Party with Sprite Lab	Using Sprite Lab, students create their own alien dance party with interactions between characters and user input.
Digital Citizenship	**8**	**Private and Personal Information**	Created by Common Sense Education, this lesson is about the difference between information that is safe to share online and information that is not.
	9	About me with Sprite Lab	By creating an interactive poster in Sprite Lab, students will apply their understanding of sharing personal and private information on the web.
	10	**Digital Sharing**	Students will learn the proper way to use content that is not their own.
Nested Loops	11	Nested Loops in Maze	In this online activity, students will have the opportunity to push their understanding of loops to a whole new level.
	12	Fancy Shapes using Nested Loops	Students will create intricate designs using Artist. By continuing to practice nested loops with new goals, students will see more uses of loops in general.
	13	Nested Loops with Frozen	This lesson will take students through a series of exercises to help them create their own portfolio-ready images.
Functions	**14**	**Songwriting**	This lesson will help students intuitively understand why combining chunks of code into functions can be such a helpful practice.
	15	Functions in Minecraft	Students will begin to understand how functions can be helpful in this fun and interactive Minecraft adventure.
	16	Functions with Harvester	Students will use functions to harvest crops in Harvester. This lesson will push students to use functions in the new ways by combining them with \`while\` loops and \`if / else\` statements.
	17	Functions with Artist	Students will be introduced to using functions with the Artist to create and modify magnificent images.
Impacts of Computing	18	**Designing for Accessibility**	In this lesson, students will learn about accessibility and the value of empathy through brainstorming and designing accessible solutions for hypothetical apps.
Project	19	End of Course Project	Students will be given their own space to create their project with either Artist or Sprite Lab.

Course F

Overview
The final course in CS Fundamentals is tailored to the needs of students in the fifth grade.

In this course, students will investigate problem-solving techniques and discuss societal impacts of computing and the internet. By the end of the course, students will have created interactive stories and games that they can share with their friends and family.

In Course F, students begin to understand how the concepts that they have learned impact the world around them and how they can be applied to solve interesting and personally-relevant problems. By this point, students should be cognitively mature enough to think about plans that they want to bring to life and have the skills to start down that path.

Starting with the first few lessons, students are given greater autonomy and creative freedom in programming, which also necessitates an increased emphasis on debugging and problem solving. Students in the fifth grade should be expected to take the first steps in solving all of their own coding problems as they arise. When solving problems, they should be encouraged to work with peers to overcome obstacles rather than relying on the teacher to do so.

Remember, *solving* a puzzle is not as important as *understanding* a puzzle, so when students are stuck, encourage them to look at several angles until a solution begins to appear.

Core concepts:

- Digital Citizenship
- Variables
- Data
- For Loops
- Sprites

Attitudinal goals:

- I can use computer science to solve real and meaningful problems.
- Programming is creative.

Key teaching tips:

- Talk with students before you begin about how they may experience frustration.
- Use pair programming and encourage students to help each other.
- Require students to make a first attempt at problem solving before asking for help.
- Remind students of the importance of persistence.
- Encourage students to use a journal during and after activities.
- Promote an environment of cross-team collaboration for group activities and projects.

Course F: Lesson Outlines

Online lessons are in regular text and unplugged lessons are **bolded**.

Concept Chunk	#	Lesson Name	Description
Ramp Up	1	Functions in Minecraft	Students will begin to understand how functions can be helpful in this fun and interactive Minecraft adventure.
	2	Swimming Fish with Sprite Lab	This lesson is designed to introduce students to the Sprite Lab programming environment and allow them to apply concepts they learned in other environments to this tool.
	3	Alien Dance Party with Sprite Lab	Using Sprite Lab students create their own alien dance party with interactions between characters and user input.
	4	Drawing with Loops	This Artist stage will allow students to create images of increasing complexity using new blocks and the concept of loops.
	5	Nested Loops in Maze	In this online activity, students will have the opportunity to push their understanding of loops to a whole new level.
Digital Citizenship	**6**	**The Power of Words**	Created by Common Sense Education, students learn what they should do when someone uses mean or hurtful language on the internet.
Variables	**7**	**Envelope Variables**	This lesson explains what variables are and how to use them.
	8	Variables with Artist	Students explore the creation of repetitive designs using variables in Artist.
	9	Changing Variables with Bee	Students will get further practice with variables with the bee.
	10	Changing Variables with Artist	This artist level takes variables to new heights.
Data	11	Simulating Experiments	By running a simple simulation in Sprite Lab, students will experience how computing can be used to collect data that identify trends or patterns.
	12	AI for Oceans	This tutorial is designed to quickly introduce students to machine learning, a type of artificial intelligence. Students will explore how training data is used to enable a machine learning model to classify new data.
	13	**The Internet**	In this lesson, students will pretend to flow through the internet, all the while learning about connections, URLs, IP addresses, and the DNS.
For Loops	**14**	**For Loop Fun**	Students play a game with dice to learn a powerful new programming concept: for loops.
	15	For Loops with Bee	This lesson focuses on `for` loops as students look for patterns in puzzles with the bee.
	16	For Loops with Artist	Students continue to practice `for` loops, this time with Artist.
Sprites	17	Behaviors in Sprite Lab	Here, students will use Sprite Lab to create their own customized behaviors.
	18	Virtual Pet with Sprite Lab	In this lesson, students will create an interactive Virtual Pet that looks and behaves how they wish.
Project	19	End of Course Project	Students will be given their own space to create their project with either Artist or Sprite Lab.

Teaching and Learning Strategies

The following teaching and learning strategies for CS Fundamentals are used repeatedly in many different lessons and units. They represent our ideal approach to delivering these lessons in a classroom and are at the core of the ways the curriculum is designed as we believe these are critical to positive classroom culture and ultimately student learning.

Lead Learner

What is it?
The curriculum has been written with the idea that the instructor will act as the lead learner. As the lead learner, your role shifts from being the source of knowledge to being a leader in seeking knowledge. The lead learner's mantra is: "I may not know the answer, but I know that together we can figure it out." The philosophy of the lead learner is that you don't have to be an expert on everything; you can start teaching CS Fundamentals knowing what you already know and learn alongside your students. To be successful with this style of teaching and learning, the most important things are modeling and teaching how to learn.

How does it connect to the curriculum?
One of the Code.org curriculum values is developing teachers who are new to computer science. In order to support those teachers, the curriculum is set up to create an engaging and relevant class that helps students uncover and develop the knowledge they need. This makes it possible for a teacher to lead the course without knowing all of the answers at first, as long as they embrace the lead learner role. In addition, it is not possible to have complete command over every rapidly-changing facet of computer science, no matter how much experience you have. Rather than feeling daunted, the lead learner welcomes this fact.

We believe that the lead learner technique represents good teaching practice in general. Acting as the lead learner is an act of empathy toward your students and the challenges they face in learning material for the first time. One important job of the teacher in the CS Fundamentals classroom is to model excitement about investigating how things work by asking motivating questions about why things work the way they do and or why they are the way they are. With your guidance, students will learn how to hypothesize; ask questions of peers; test, evaluate, and refine solutions collaboratively.

How do I use it?

- Allow students to dive into an activity without front loading the content first.
- Encourage students to rely on each other for support.
- Don't give the answer right away, even if you know it.
- Feel open to making mistakes in front of students so that they see it is part of the learning process.
- Ask students questions that direct their attention toward the issue to investigate without giving away what they need to change.
- Model the steps you would go through as a learner of a new subject. Explain the different questions you ask yourself along the way and the ways you go about finding answers.

Pair Programming

What is it?

Pair programming is a technique in which two programmers work together at one computer. The “driver” writes code while the “navigator” directs the design and setup of the code. The two programmers switch roles often. Pair programming has been shown to:

- improve computer science enrollment, retention, and students' performance.
- increase students' confidence.
- develop students' critical thinking skills.
- introduce students to a "real world" working environment.

How does it connect to the curriculum?

In CS Fundamentals there are many lessons on the computer (plugged lessons) during which students develop programming skills. Pair programming can help to foster a sense of camaraderie and collaboration and can promote diversity in the classroom by reducing the "confidence gap" between female and male students, while increasing the programming confidence of all students.

How do I use it?

To get students pair programming:

1. Form pairs.
2. Give each pair one computer to work on.
3. Decide upon initial roles.
4. Have students start working.
5. Ensure that students switch roles at regular intervals (every 3 to 5 minutes).
6. Ensure that navigators remain active participants.

It can be hard to introduce pair programming after students have worked individually for a while, so we recommend that teachers start with pair programming in the first few lessons. Just like any other classroom technique, you may not want to use this all the time as different types of learners will respond differently to working in this context. Once you have established pair programming as a practice early on, it will be easier to come back to later.

Resources

Code.org also has a feature to help both students get “credit” on their accounts for the work they do together. Check out the blog on Pair Programming: https://goo.gl/MorPnx.

Videos:

- For Teachers: youtu.be/sxToW3ixrwo
- For Students: youtu.be/vgkahOzFH2Q

The National Center for Women & Information Technology (NCWIT) has a great resource about the benefits of pair programming. Check it out at: www.ncwit.org/resources/pair-programming-box-power-collaborative-learning

Authentic Choice

What is it?

Authentic choice is the practice of allowing students to decide on the focus of their creation when they are working on a project. This can be scoped in different ways with different projects, but the central point is to allow students to work on something they are personally invested in.

How does it connect to the curriculum?

In the curriculum, we give students many opportunities to work on projects that we hope will feel personally relevant. Whether it be a small freeplay level at the end of a lesson or a course project designed by students in upper elementary, every student should get ample opportunity to develop creations of their own.

In addition, we encourage teachers to help students utilize their new skills in creative ways at the end of each lesson using the Lesson Extras option. There, students will find challenge puzzles and open-canvas projects to use for deeper learning and self-expression.

How do I use it?

- Give students time to get creative and find something they are passionate about in the project that they are working on.
- Encourage students to find personally relevant contexts for the work they do.
- Try to keep the projects as open to students' interests as possible while still keeping them focused on the learning at hand.

Journaling

What is it?

In CS Fundamentals, students are encouraged to keep a journal nearby to write down thoughts and answer questions.

How does it connect to the curriculum?

Courses A-F of Computer Science Fundamentals were written with the importance of journaling in mind. Journaling for reflection is a popular tool in education, but we take that one step further. Like a chemist would catalog strategies and solutions, so do we ask our budding computer scientists to take notes on their trials and achievements. Journals are useful as scratch paper for building, debugging, and strategizing, and they offer a fantastic resource for referencing previous answers when struggling with more complex problems.

How do I use it?

- Encourage students to keep their journals beside them at all times when coding.
- Remind students that they can write solutions out longhand, then circle patterns to find prime opportunities for loops and functions.
- Have students copy down answers to puzzles that they might need in future levels.
- Ask students to draw emoticons at the top of the pages to help them identify how they're feeling about concepts.
- End each lesson with a thought or question that students can answer in writing as a way of reflecting on their growth for the day.

Student Practices

Lessons in CS Fundamentals help students work in a wide array of contexts, but these experiences are tied together by a core set of practices and skills they develop throughout each course. These student practices provide coherence and serve as helpful reminders of the high-level skills and dispositions we want students to develop.

Problem Solving

- Use a structured problem solving-process to help solve new problems.
- View challenges as solvable problems.
- Break down larger problems into smaller components.

Persistence

- Value and expect mistakes as a natural and productive part of problem solving.
- Continue working towards solutions in spite of setbacks.
- Iterate and continue to improve partial solutions.
- Keep track of elements that worked and elements that did not to avoid repeating mistakes.

Creativity

- Incorporate your own interests or ideas into your work.
- Experiment with new ideas and consider multiple possible approaches.
- Extend or build upon the ideas and projects of others.

Collaboration

- Work with others to develop solutions that incorporate all contributors.
- Mediate disagreements and help teammates agree on a common solution.
- Actively contribute to the success of group projects.

Communication

- Structure your work so that it can be easily understood by others.
- Consider the perspective and background of your audience when presenting your work.
- Provide and accept constructive feedback in order to improve your work.

Debugging

Everyone gets bugs! Therefore, debugging is an essential skill all students should develop that builds on the student practices discussed above. Without strong debugging skills, students can become frustrated. Help keep students moving by implementing both active and reflective debugging strategies in your classroom.

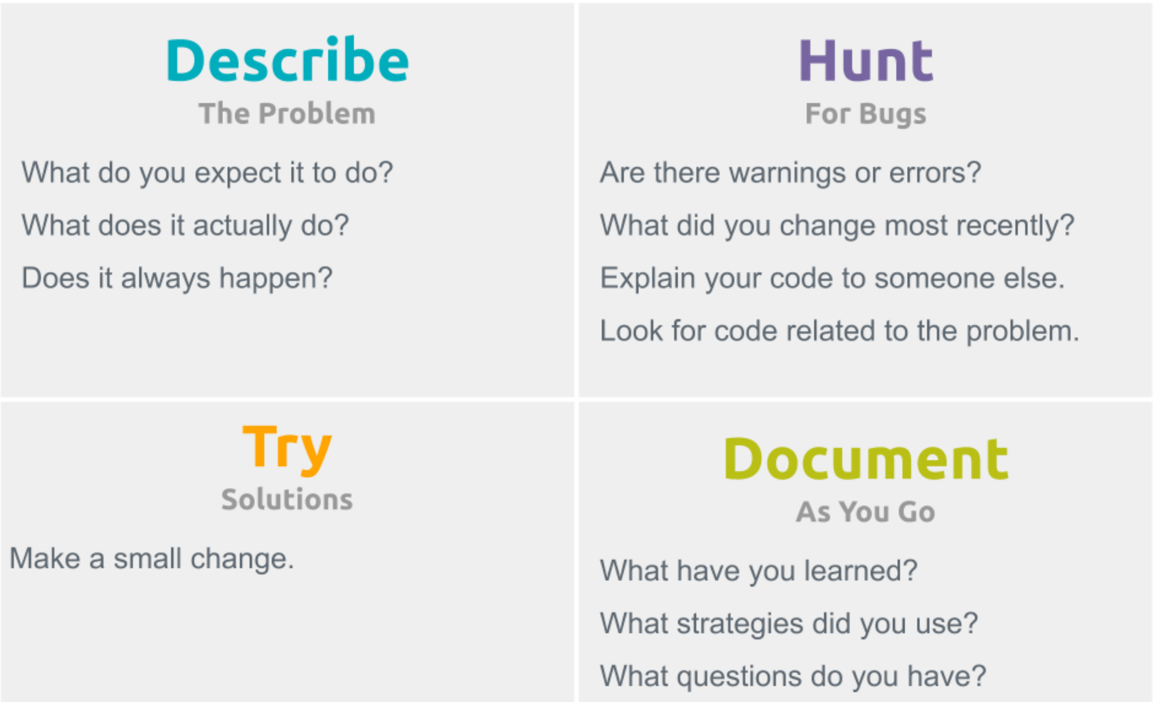
Active Debugging

New programmers are prone to writing long chunks of code without pausing to read or test their work along the way, which can snowball into a program that is very hard to debug. To prevent this, we suggest implementing active debugging as part of a general coding philosophy.

Active debugging describes the practice of debugging *while* coding. This generally means students taking time during coding exercises to read, process, and test small pieces of code that they have just written. “Small pieces of code” is relative, of course, particularly if students are tasked with coming up with a solution that requires just one or two blocks. However, the practice itself can be tremendously helpful as students write longer programs, which gradually occurs in all CS Fundamentals courses.

A practical student-facing guide to active debugging exists in our **Debugging Guide**, which can be found on the Course Overview Page of each course, and is shown below. This guide lists active debugging strategies for students. We suggest making this document available to students as a classroom poster or individual handout, then referring to it as students progress through your course.

Guide to Debugging



Describe
The Problem

What do you expect it to do?

What does it actually do?

Does it always happen?

Hunt
For Bugs

Are there warnings or errors?

What did you change most recently?

Explain your code to someone else.

Look for code related to the problem.

Try
Solutions

Make a small change.

Document
As You Go

What have you learned?

What strategies did you use?

What questions do you have?

Reflective Debugging

While students are working, you will notice the same issues appearing again and again. Some students are able to quickly move through these issues, while others are left behind. Even with active debugging practices in place, these experiences can be disheartening for any teacher. However, they can, in fact, inspire a productive reflective debugging session at some point during the class.

Active debugging is how individual students can find and fix their own bugs. Reflective debugging, on the other hand, is a practice in which students participate in debugging as an entire class. ‘Bug Talks’ and keeping a ‘bug tracker’ are two examples of strategies that can be used to implement reflective debugging in your classroom and are described below:

Bug Talks

Bug Talks are very similar to “Number Talks” from the world of mathematics education. Teachers can use Bug Talks before, during, or after class to help students clear up misconceptions that manifest as common bugs. A typical Bug Talk should last 5-10 minutes and might proceed as follows:

1. A problem is displayed for students to solve on their own, quietly. The problem should be small enough that they can solve it mentally or with scratch paper. Students who were able to solve the problem very quickly should try coming up with more than one solution. An example problem might be a buggy solution to a puzzle on the website, perhaps similar to the one students are currently struggling with.
2. After having time to work alone, students can share their ideas with a neighbor.
3. After sharing with a neighbor, students raise hands to share answers with the class.
4. The teacher neutrally records as many answers as possible on the board. That is, the teacher does not say which are correct or incorrect. Duplicate answers are given +1, +2, and so forth.
5. Teacher calls on students to explain the reasoning behind their answers. Students can respond to each other’s arguments with sentence starters like, “I agree/disagree, because…”.
6. Finally, students discuss and vote on which answer they collectively believe is the best solution.

To conclude a Bug Talk, teachers should reiterate why the chosen strategy works best, including any unmentioned points the class might not have brought up.

Bug Tracker

A Bug Tracker is a poster that can help your students name and understand common problems they encounter while coding. Throughout a CS Fundamentals course, students collectively drive the development of their class’ Bug Tracker through teacher-led discussions about coding and problem solving.

Why Create a Bug Tracker?	Small misconceptions can lead to big problems when learning to code. You can help students clear these misconceptions by facilitating class discussions about specific instances of bugs they catch. During these discussions, your Bug Tracker can be used as a class-sourced running record that generalizes those instances into more widely applicable and accurate coding concepts. In addition to identifying and describing common bugs, students should share debugging strategies on your Bug Tracker as well. This collective exchange of knowledge can help empower students to persist together through even the toughest challenges.
When Should I Introduce a Bug Tracker?	To ensure your class develops the foundation of documenting bugs and debugging as early as possible, the introduction of the Bug Tracker should coincide with the first programming lesson, then be revisited throughout the course as needed.
How Do I Use the Bug Tracker?	You may want to discuss bugs students have found at the beginning or end of class, or even potentially during class if it seems many students are encountering the same types of problems. Encourage students to refer back to the Bug Tracker while they work. You can challenge your students to look for the bugs they spotted as they work. Have them tell a neighbor when they see one. Can they come up with strategies to get past it? Ask students to hold on to new bugs if they happen to find them. They can share their bugs during the next Bug Talk.
What “Bugs” Might Appear on a Bug Tracker?	This is actually a great question and one that we hope your students will grapple with as they learn. At least at the beginning of your course, you might hear students suggest, “The computer won’t turn on,” or “The Internet is slow” be added to the class’ Bug Tracker. This is a perfect time to discuss, and further instill, the meaning of “coding” and “debugging,” although relating these CS-specific concepts to other difficulties students experience (technical and non-technical) is certainly worthy of discussion.

Course Resources

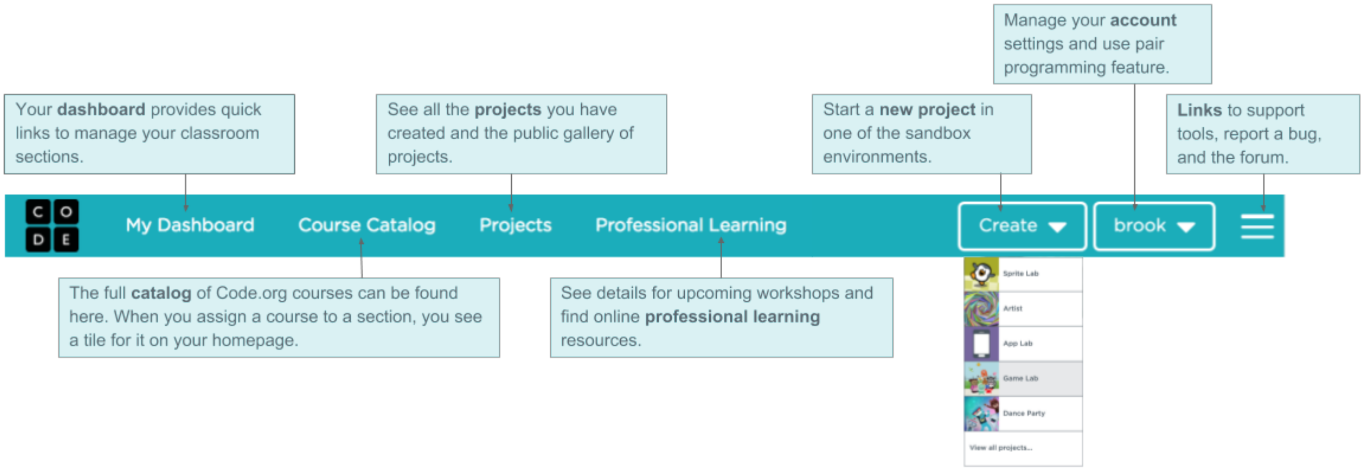
The CS Fundamentals curriculum is made up of student-facing and teacher-facing components. Teachers will access curriculum materials in two different places on the Code.org website: our Code Studio platform and in the teacher-facing curriculum. The table below outlines what you can do in each of these places:

Code Studio	Teacher-facing Curriculum
● Access all online student-facing lesson materials ○ Review completed student work, including program code and assessment questions ● Create and manage sections of students, including assigning courses and lessons to students	● Access teacher-facing lesson plans that provide detailed context for how to deliver lessons ● Navigate links to all printable materials needed for the course ● Explore course resources, such as standards mapping, vocabulary lists, code documentation, PDFs of lessons, etc.

The following pages contain an overview of the layout and organization of these important course resources.

Code.org Website

Log in to Code.org with your teacher account. The website header will help you navigate the site:



Manage your **account** settings and use pair programming feature.

Your **dashboard** provides quick links to manage your classroom sections.

See all the **projects** you have created and the public gallery of projects.

Start a **new project** in one of the sandbox environments.

Links to support tools, report a bug, and the forum.

The full **catalog** of Code.org courses can be found here. When you assign a course to a section, you see a tile for it on your homepage.

See details for upcoming workshops and find online **professional learning** resources.

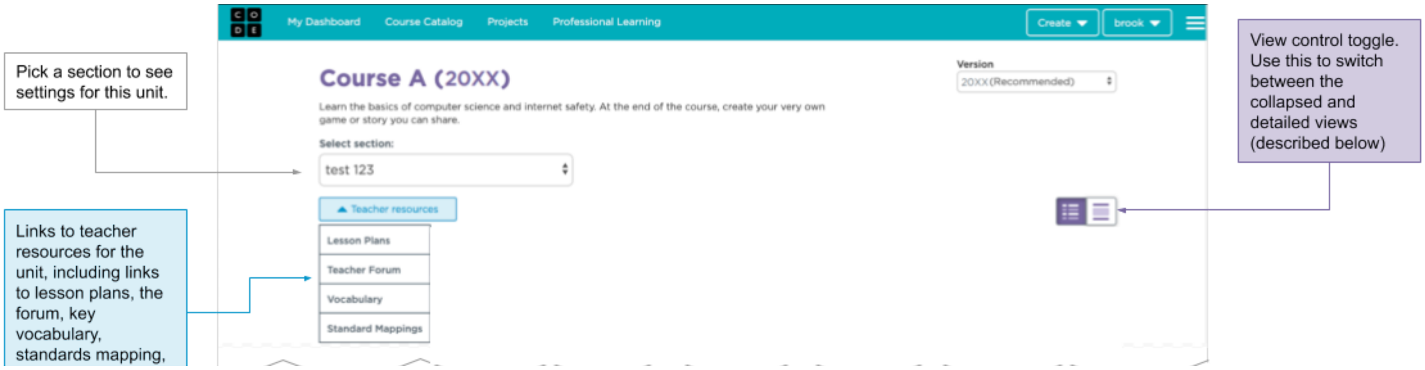
The Code.org home page is the starting point for everything in the curriculum. To get started with your students, you will need to create a section. For details on how to create a section, visit the **getting started** support articles at support.code.org.

Once you've assigned your CS Fundamentals students to a section, a tile that can be used to access the course overview page will appear on the homepage. This is your starting point for lesson planning and all the resources you need to teach the course.

Course Overview Structure and Iconography

Code Studio — Course Overview

The course overview page on Code Studio (e.g., studio.code.org/s/coursea) is a hub for managing your course and includes the following:



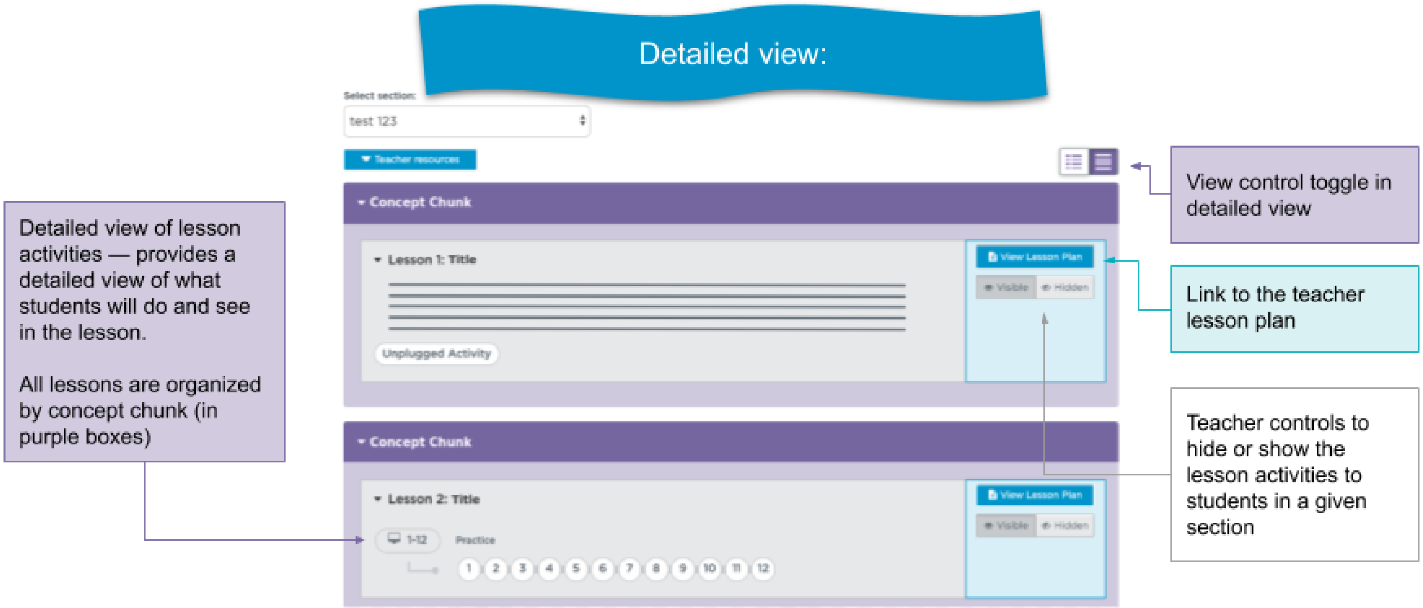
Learn the basics of computer science and internet safety. At the end of the course, create your very own game or story you can share.

Pick a section to see settings for this unit.

Links to teacher resources for the unit, including links to lesson plans, the forum, key vocabulary, standards mapping,

View control toggle. Use this to switch between the collapsed and detailed views (described below)

Using the toggle on the top right of the unit overview page in Code Studio yields one of two options: a detailed view of the unit or a collapsed view:



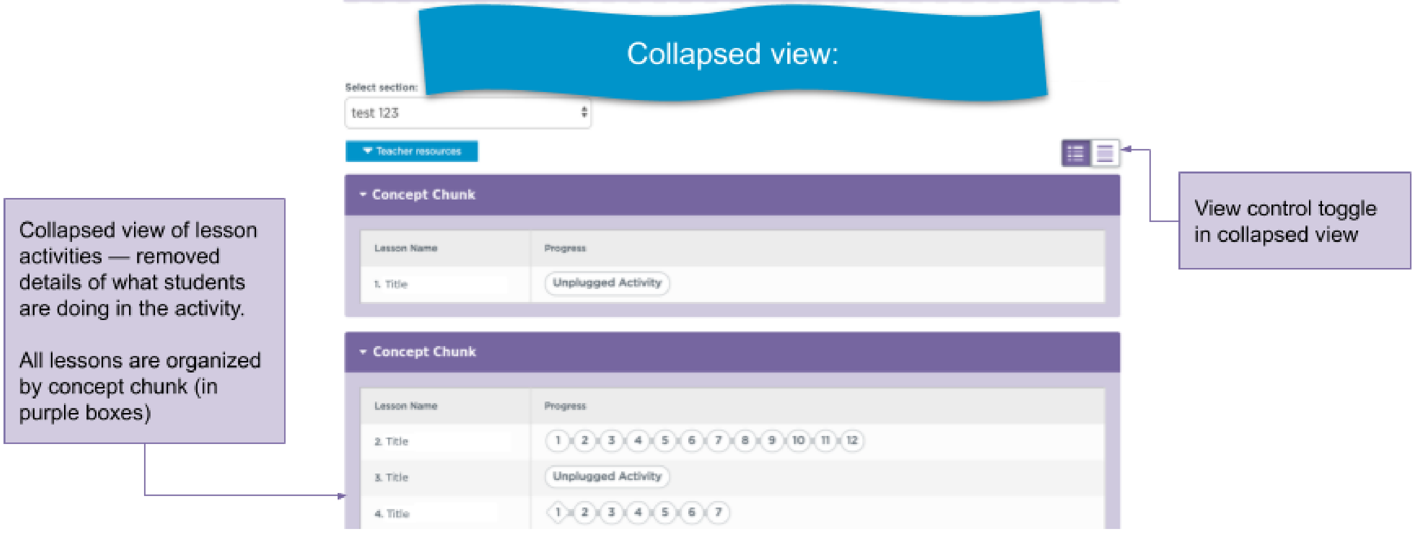
View control toggle in detailed view

Link to the teacher lesson plan

Teacher controls to hide or show the lesson activities to students in a given section

Detailed view of lesson activities — provides a detailed view of what students will do and see in the lesson.

All lessons are organized by concept chunk (in purple boxes)



View control toggle in collapsed view

Collapsed view of lesson activities — removed details of what students are doing in the activity.

All lessons are organized by concept chunk (in purple boxes)

Code Studio - Iconography on Course Overview page

When looking at the detailed view of the course overview page (e.g., studio.code.org/s/coursea) described above, you will notice a number of different icons that represent different types of levels within a given lesson. Those icons are listed below, along with a brief description of what they represent and how you might use them as a teacher.

✂ Unplugged

These levels contain videos along with any other resources necessary for students to access while completing unplugged lessons.

Because the nature of an unplugged lesson means that you might not want your students on a computer during the lesson, you should feel free to project these resources at the front of the room and invite any students who missed the unplugged lesson to use the materials posted here as a tool to learn more about what they missed.

Text

These levels contain instructions, text, or images to help you run a class activity. Lesson instructions will indicate how these levels should be incorporated into the activity. A lesson overview provides a short activity description and links to documents used throughout the lesson.

Consider going over these as a whole-class activity. These also provide good stopping points to check in with the students and make sure everyone is together before moving on to the next set of tasks.

Video

Video levels contain a video to be used in the curriculum and are typically hosted in multiple formats, including a downloadable file, to be compatible with a variety of technology needs across classrooms.

Videos can be watched as a whole class to allow for group discussion afterward. Remind students that they can use these videos as reference if they need some extra help during programming.

☰ Question

These levels represent some sort of check for understanding, usually in the form of multiple-choice or free-response questions. You will find these levels in individual lessons, indicated with an assessment icon. In that case, these are intended to be used as *formative* assessment items. Students can always see them and change their responses at any time.

Question levels are also in the post-project test found at the end of each unit. In those cases, the items are meant to be *summative* assessment items.

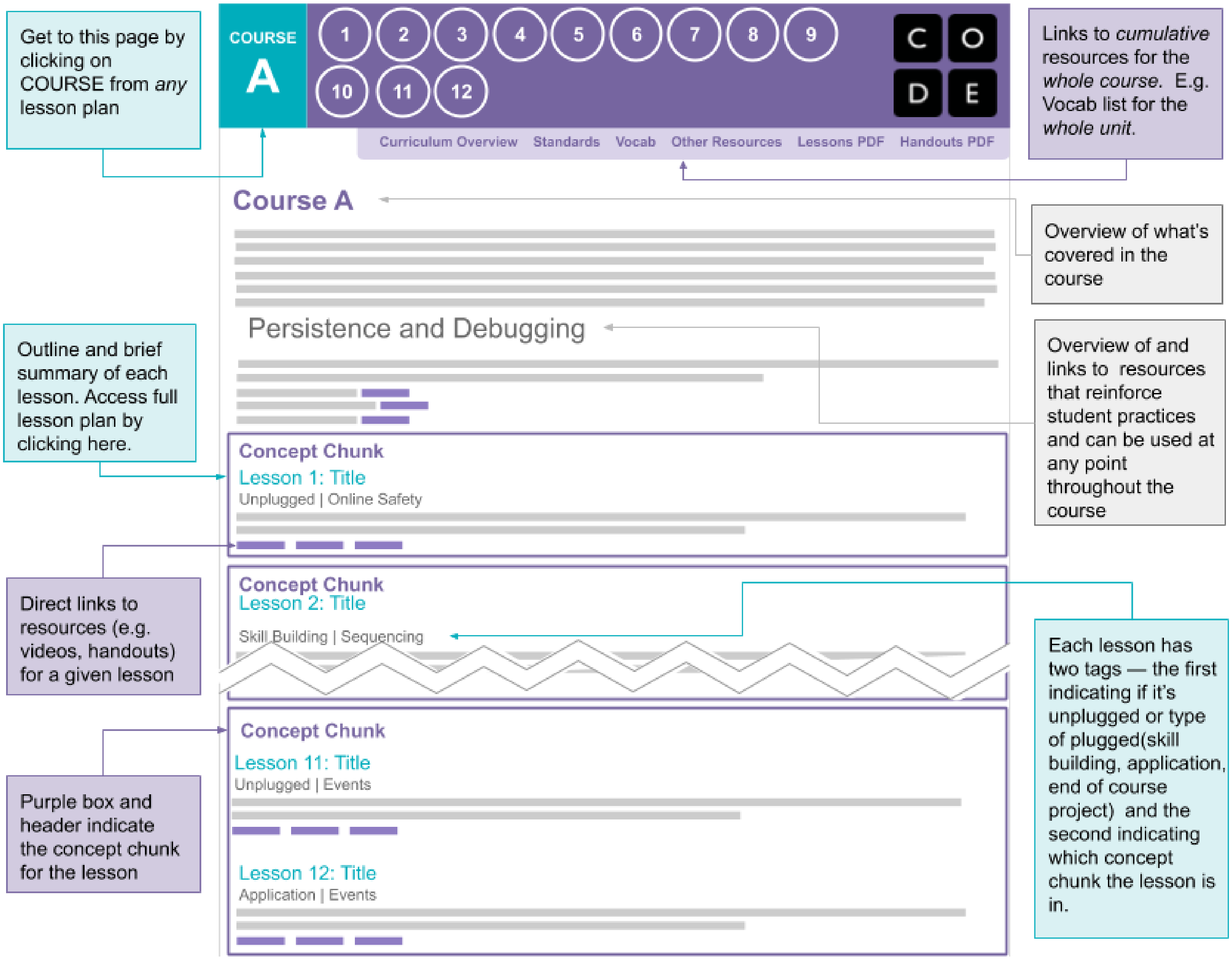
Online

These levels use a Code.org tool or programming environment. An instructions panel explains any new content introduced in the level, provides a checklist of tasks to complete, and may include starter code. Teachers can review their students' code from the Teacher Panel.

Enable students to develop skills by completing targeted tasks individually or in pairs. Support them by directing them to available resources and helping them to develop general coding strategies

Teacher-facing curriculum — Course Overview

The teacher-facing course overview page (e.g., curriculum.code.org/csf/coursea) is a hub for seeing all lesson plans for a given course and includes the following:



Get to this page by clicking on COURSE from *any* lesson plan

Links to *cumulative* resources for the *whole course*. E.g. Vocab list for the *whole unit*.

Overview of what's covered in the course

Outline and brief summary of each lesson. Access full lesson plan by clicking here.

Overview of and links to resources that reinforce student practices and can be used at any point throughout the course

Direct links to resources (e.g. videos, handouts) for a given lesson

Each lesson has two tags — the first indicating if it's unplugged or type of plugged(skill building, application, end of course project) and the second indicating which concept chunk the lesson is in.

Purple box and header indicate the concept chunk for the lesson

Note that from this view, lessons are organized by concept chunk, and there are direct links to any materials needed in the lesson.

Lesson Structure and Iconography

Lessons in CS Fundamentals are written for a wide variety of classrooms. A typical lesson is broken down with time estimates, but the actual time will vary based on the age of your students, their background with the material, and their interests. Teachers looking to complete our courses in less than the suggested time have also found success by looking at clusters of related lessons together. See **Concept Chunks** above.

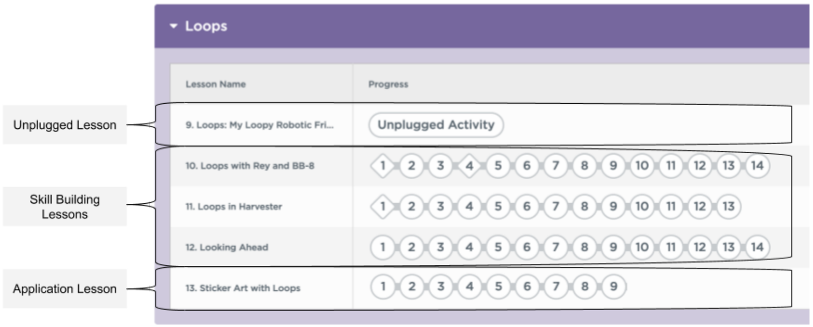
Types of Lessons in CS Fundamentals

The CS Fundamentals curriculum has four main types of lessons, which build in complexity and allow students to apply their learning in different contexts. Those four lessons types are:

1. **Unplugged.** These lessons are done away from the computer and are often used to introduce a new concept in a hands-on, tangible way.
2. **Skill Building**. These lessons are done on the computer and give students structured practice with a new tool or programming concept.
3. **Application**. These lessons are done on the computer and give students space to apply learned concepts in creative ways.
4. **End-of-Course Project.** These longer lessons are done both on and off the computer and provide an open-ended space for students to bring together everything they've learned in the course into a single, creative project.

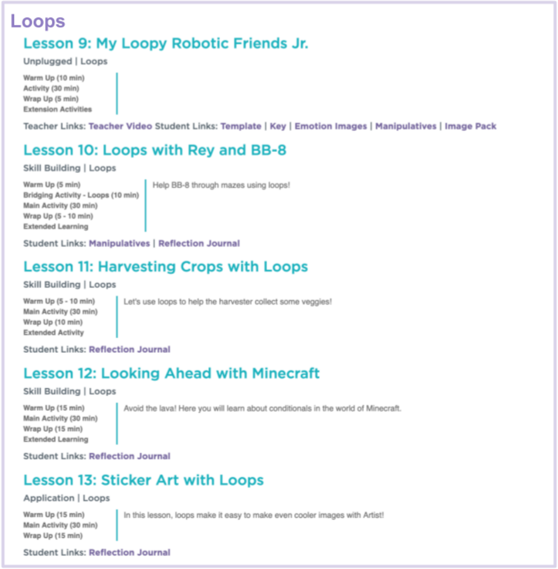
These lesson types contribute to the flow of concept chunks, which were described on page 6. See below for an example of a concept chunk from Course C as seen on Code Studio (left) and in the teacher-facing curriculum (right):

Concept chunk in Code Studio:



Lesson Name	Progress
9. Loops: My Loopy Robotic Fri...	Unplugged Activity
10. Loops with Rey and BB-8	1 2 3 4 5 6 7 8 9 10 11 12 13 14
11. Loops in Harvester	1 2 3 4 5 6 7 8 9 10 11 12 13
12. Looking Ahead	1 2 3 4 5 6 7 8 9 10 11 12 13 14
13. Sticker Art with Loops	1 2 3 4 5 6 7 8 9

Concept chunk in teacher-facing Curriculum



Loops

Lesson 9: My Loopy Robotic Friends Jr.

Unplugged | Loops

Warm Up (10 min)
Activity (30 min)
Wrap Up (5 min)
Extension Activities

Teacher Links: Teacher Video Student Links: Template | Key | Emotion Images | Manipulatives | Image Pack

Lesson 10: Loops with Rey and BB-8

Skill Building | Loops

Warm Up (5 min)
Bridging Activity - Loops (10 min)
Main Activity (30 min)
Wrap Up (5 - 10 min)
Extended Learning

Help BB-8 through mazes using loops!

Student Links: Manipulatives | Reflection Journal

Lesson 11: Harvesting Crops with Loops

Skill Building | Loops

Warm Up (5 - 10 min)
Main Activity (30 min)
Wrap Up (10 min)
Extended Activity

Let's use loops to help the harvester collect some veggies!

Student Links: Reflection Journal

Lesson 12: Looking Ahead with Minecraft

Skill Building | Loops

Warm Up (15 min)
Main Activity (30 min)
Wrap Up (15 min)
Extended Learning

Avoid the lava! Here you will learn about conditionals in the world of Minecraft.

Student Links: Reflection Journal

Lesson 13: Sticker Art with Loops

Application | Loops

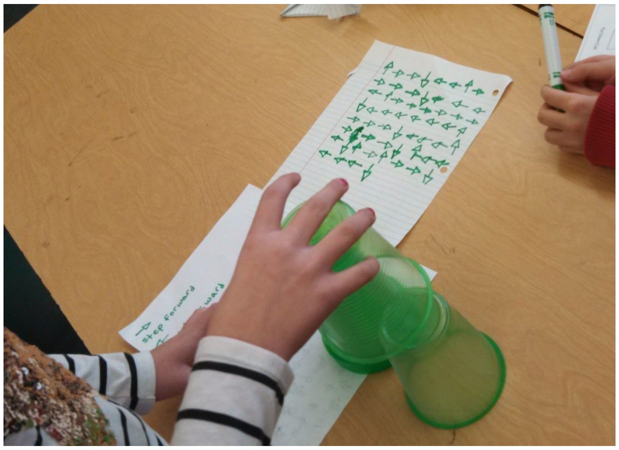
Warm Up (15 min)
Main Activity (30 min)
Wrap Up (15 min)

In this lesson, loops make it easy to make even cooler images with Artist!

Student Links: Reflection Journal

The following pages provide more details on each of these lesson types and how they manifest in concept chunks.

Unplugged Lessons

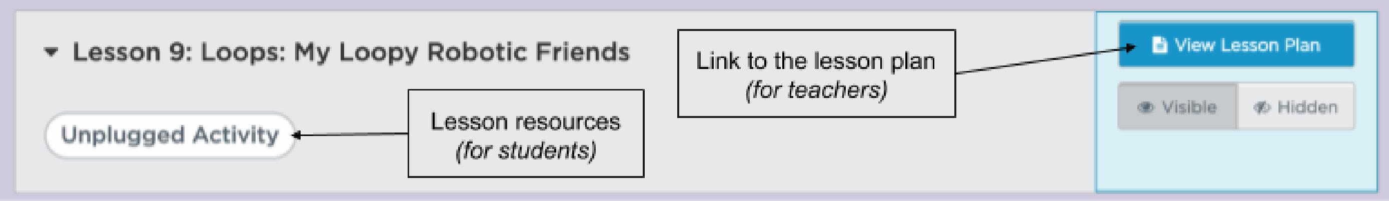


We refer to lessons in which students are not working on a computer as "unplugged." Students will often work with pencil and paper or physical manipulatives. These are intentionally placed kinesthetic opportunities that help students digest complicated concepts in ways that relate to their own lives. Often an unplugged lesson sets the stage for a subsequent skill-building lesson investigating the same concept on the computer. Both types of lessons are vital pieces of the curriculum.

Unplugged lessons are particularly useful for building and maintaining a collaborative classroom environment, and they are useful touchstone experiences you can refer to when introducing more abstract concepts. While these lessons sometimes involve more advanced preparation, they provide a shared and concrete context that can be referenced during other lessons. For a list of all unplugged lessons covered in the CS Fundamentals curriculum (plus a few extras!) visit: code.org/curriculum/unplugged

Tips for Effectively Teaching Unplugged Lessons:

- ❏ Don't skip these activities! They're often an essential introduction to new concepts.
- ❏ Teach lessons in the order they are written. The sequence is designed to scaffold student understanding.
- ❏ Help students identify the computer science concepts underlying the activities.
- ❏ Refer back to unplugged lessons to reinforce concepts in subsequent lessons.

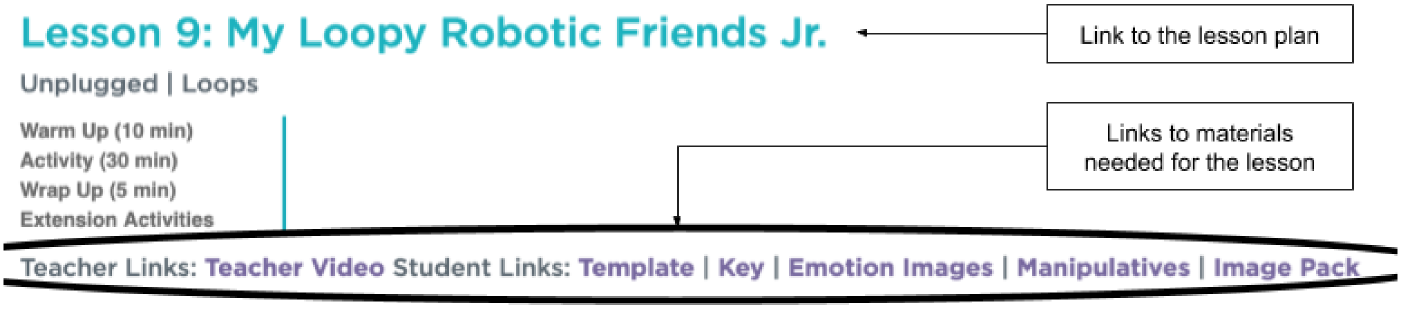


▾ Lesson 9: Loops: My Loopy Robotic Friends

Unplugged Activity
Lesson resources (for students)
Link to the lesson plan (for teachers)
View Lesson Plan
Visible
Hidden

Code Studio: Unplugged lesson example from Course C, "Loops" concept chunk

The unplugged activity bubble often contains a video which can be shared with students prior to the main activity. Be sure to review the full lesson plan.



Lesson 9: My Loopy Robotic Friends Jr.

Unplugged | Loops

Warm Up (10 min)
Activity (30 min)
Wrap Up (5 min)
Extension Activities

Teacher Links: Teacher Video Student Links: Template | Key | Emotion Images | Manipulatives | Image Pack

Teacher-Facing Curriculum: Unplugged lesson example from Course C, "Loops" concept chunk

Skill-Building Lessons

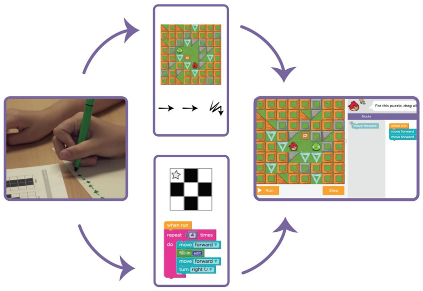
The majority of “plugged” or computer-based lessons in CS Fundamentals are skill building and are designed to help students get hands-on practice with tools and concepts. On Code Studio, these lessons typically consist of **practice** puzzles, **videos**, **prediction** puzzles, and **free play** activities. You can read more about these Code Studio activity types in the next section.

Skill building lesson plans typically have many of the same features as their unplugged counterparts. Lessons will begin and end with discussions or activities that help motivate and synthesize learning. Key moments for you to check in with your students are noted in lesson plans. Students will use a computer, but the ways students interact with each other and your role as the teacher are still important considerations.

These puzzle progressions generally start with a sequence of practice puzzles that gradually increase until reaching a challenge puzzle. There are often additional puzzles after the challenge that intentionally get easier to help build efficacy and confidence at the end of the lesson.

Bridging Activities

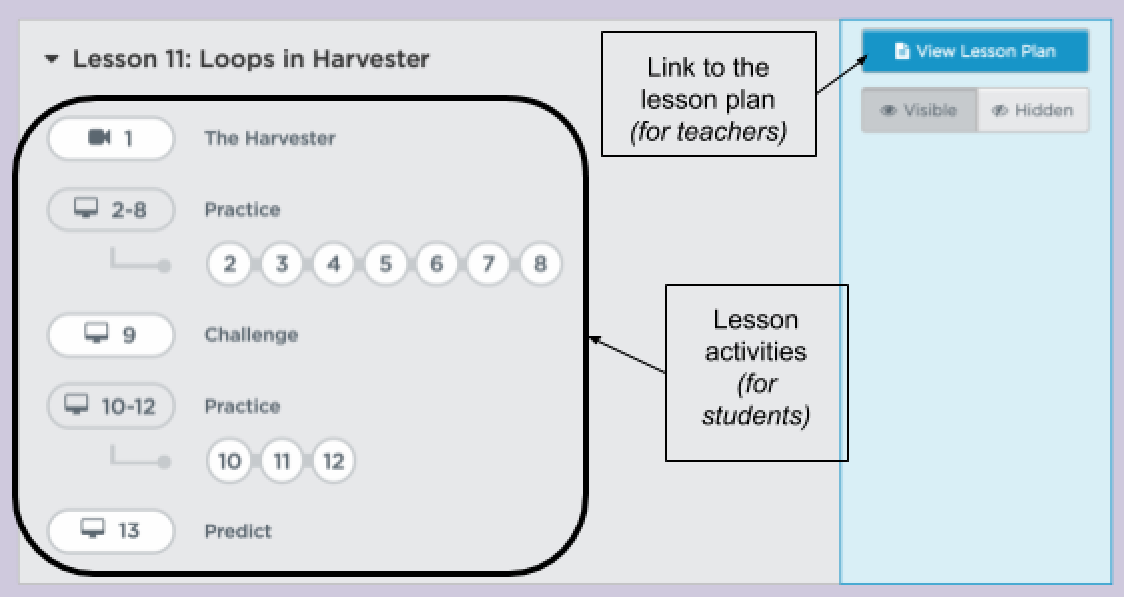
Bridging activities connect our unplugged lessons to our skill building lessons in a real and concrete way. They often exist as a method of turning an abstract concept or idea learned through unplugged play into an actionable tool for the upcoming puzzles.



In a given concept chunk that starts with an unplugged lesson, you will find a bridging activity as the “warm up” in the first skill building lesson that follows. As you become more comfortable with the curriculum, feel free to come up with your own online/offline blends to keep the curriculum relevant to your classroom.

Tips for Effectively Teaching Skill-Building Lessons:

- ❏ “Plugged” doesn’t mean the computer is the teacher! In many ways, you will need to take a more active role in checking student progress since it’s hard to know what’s happening when students are working on screens.
- ❏ Use warm ups, wrap ups, and suggested check-ins to ensure students are synthesizing concepts.
- ❏ Encourage students to work with one another to maintain the collaborative classroom culture established during unplugged lessons.
- ❏ Highlight connections from the preceding unplugged lessons.



Lesson 11: Loops in Harvester

1 The Harvester

2-8 Practice

2 3 4 5 6 7 8

9 Challenge

10-12 Practice

10 11 12

13 Predict

View Lesson Plan

Visible Hidden

Link to the lesson plan (for teachers)

Lesson activities (for students)

Skill-Building lesson example from Course C, “Loops” concept chunk

Application Lessons

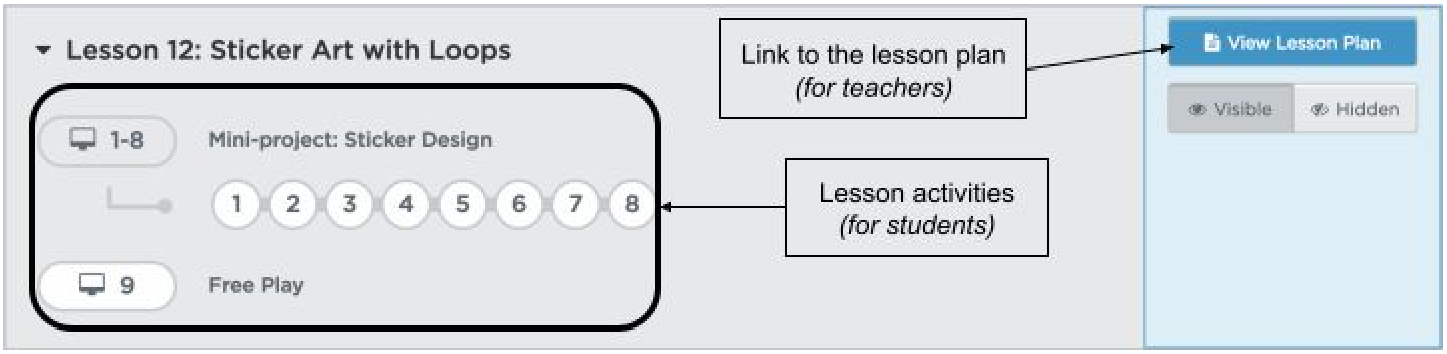
While similar in appearance to our other online puzzles, application tasks are special in that they were designed to allow students to apply what they have learned in a creative way.

These lessons typically walk students through the creation of a mini-project that will be unique for them based on their own creative decisions. There are no right or wrong answers here! Unlike skill building lessons where student work is typically validated for correctness automatically, there is no validation in these lessons. This is because the mini-project work that students do in application lessons is much more open-ended. To determine if students are successfully applying learned concepts, reviewing their code is essential.

Tips for Effectively Teaching Application Lessons:

- ❏ Consider completing the lesson's mini-project yourself ahead of time. This can build empathy for the student experience and help you to better understand the capabilities and limitations of the tool.
- ❏ Be sure to talk with the class to establish context and expectations for this type of lesson.
- ❏ Students may need to determine for themselves when they are ready for the next step in a mini-project. Stress the importance of reading instructions in these lessons.

Application Lesson Appearance on Code Studio



Application lesson example from Course C, "Loops" concept chunk

End-of-Course Projects

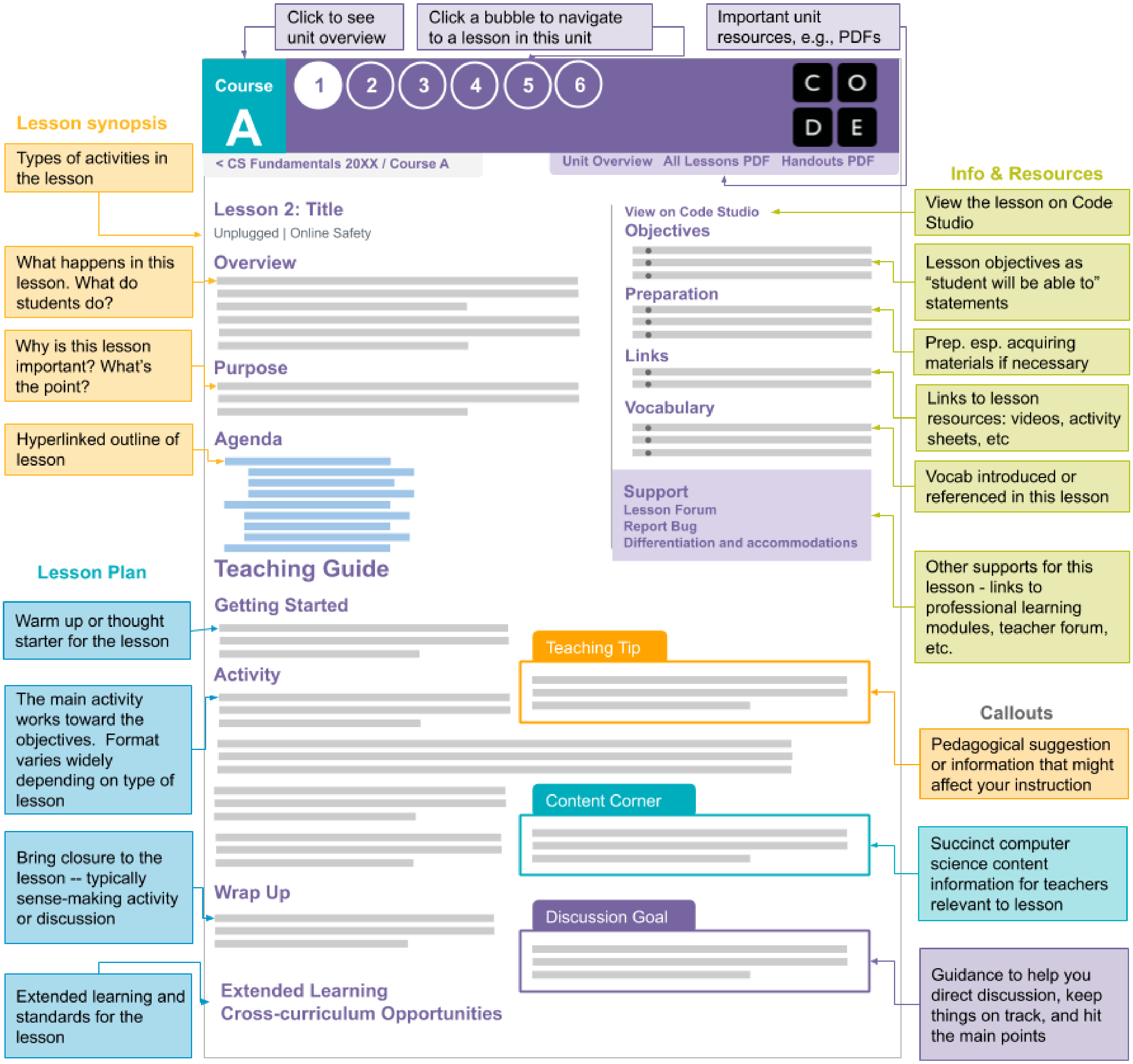
Each course offers the opportunity for students to take what they've learned at the end of a lesson and put it together into a unique project that represents their own creativity.

In Courses A-B, this takes the format of exercises that have multiple solutions. Course C takes students through a progression to build a more complex program in which the students drive many of the decisions.

In Courses D-F, project development takes the stage. Here, students are encouraged to plan, build, revise, and present projects of their own. Following a project from inception to delivery offers an inside look at the software development cycle. These guided projects offer scaffolded rubrics for the benefit of both student and teacher.

Teacher-Facing Curriculum - Lesson Plans

Every lesson plan has a common structure designed to make it easy to find what you need. As you plan for a lesson, we recommend starting with the overview, then reviewing the core activity to get a deeper sense of what will happen in the lesson and how long it might take.



Click to see unit overview

Click a bubble to navigate to a lesson in this unit

Important unit resources, e.g., PDFs

Lesson synopsis

Types of activities in the lesson

What happens in this lesson. What do students do?

Why is this lesson important? What's the point?

Hyperlinked outline of lesson

Lesson Plan

Warm up or thought starter for the lesson

The main activity works toward the objectives. Format varies widely depending on type of lesson

Bring closure to the lesson -- typically sense-making activity or discussion

Extended learning and standards for the lesson

Info & Resources

View the lesson on Code Studio

Lesson objectives as "student will be able to" statements

Prep. esp. acquiring materials if necessary

Links to lesson resources: videos, activity sheets, etc

Vocab introduced or referenced in this lesson

Other supports for this lesson - links to professional learning modules, teacher forum, etc.

Callouts

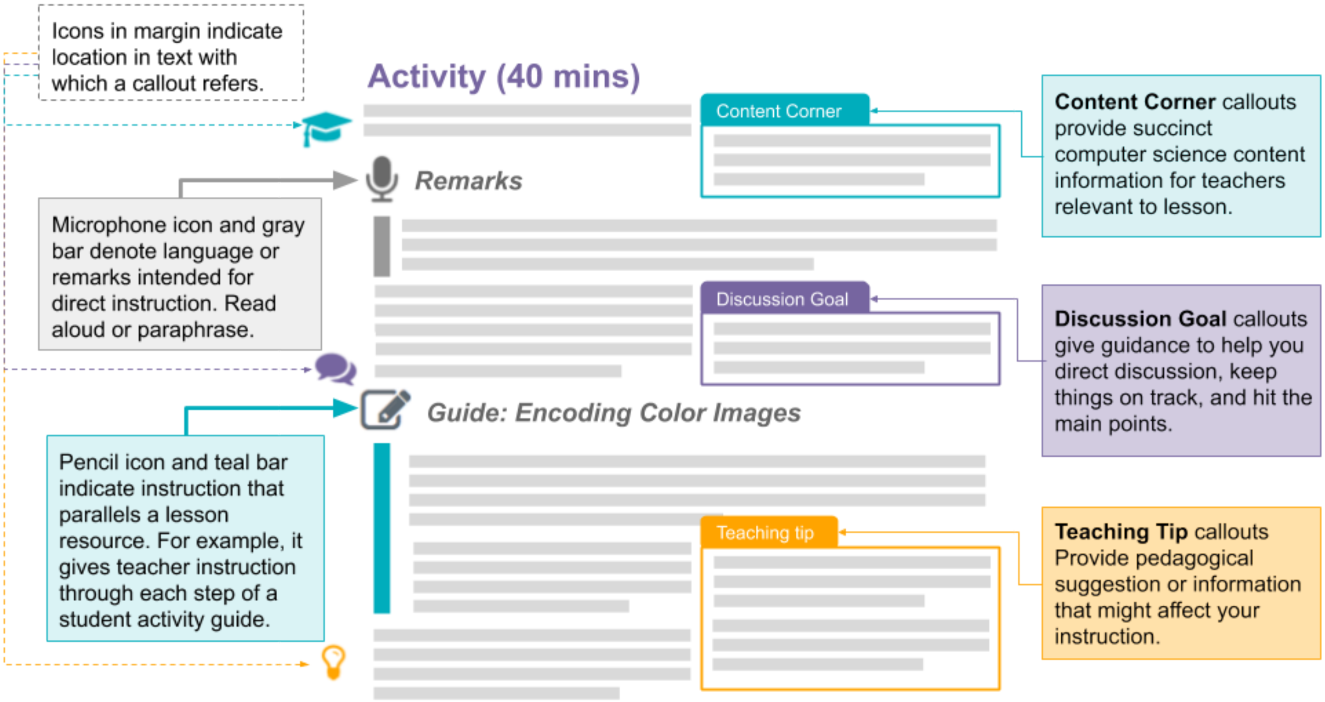
Pedagogical suggestion or information that might affect your instruction

Succinct computer science content information for teachers relevant to lesson

Guidance to help you direct discussion, keep things on track, and hit the main points

Iconography

Within lesson plans you'll notice a number of icons and other kinds of callouts. These are intended to give context about what "mode" you should be operating in for each part of the lesson. Sometimes you speak directly to the students, and other times you need to understand the goal of a discussion or give guidance during an activity.



Icons in margin indicate location in text with which a callout refers.

Microphone icon and gray bar denote language or remarks intended for direct instruction. Read aloud or paraphrase.

Pencil icon and teal bar indicate instruction that parallels a lesson resource. For example, it gives teacher instruction through each step of a student activity guide.

Content Corner callouts provide succinct computer science content information for teachers relevant to lesson.

Discussion Goal callouts give guidance to help you direct discussion, keep things on track, and hit the main points.

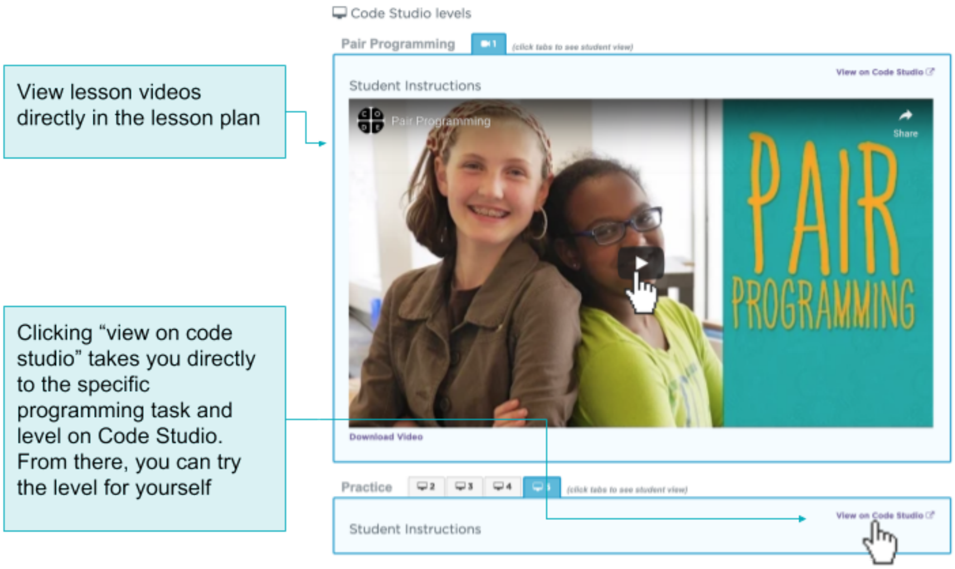
Teaching Tip callouts Provide pedagogical suggestion or information that might affect your instruction.

Interactive Code Studio View

Lesson plans give you an interactive view into all of the text content and instructions students see on the platform.

With this view, you can quickly browse through what students see for each level in the lesson without having to step through each level on Code Studio.

This should greatly speed up your preparations for class or serve as fast way to remind yourself what's in each lesson.



View lesson videos directly in the lesson plan

Clicking "view on code studio" takes you directly to the specific programming task and level on Code Studio. From there, you can try the level for yourself

Code Studio - Lesson Iconography

Once students navigate to lesson levels on Code Studio, a new set of iconography is used to communicate about some types of levels. Those icons are listed below, along with a brief description of what they represent and how you might use them as a teacher.

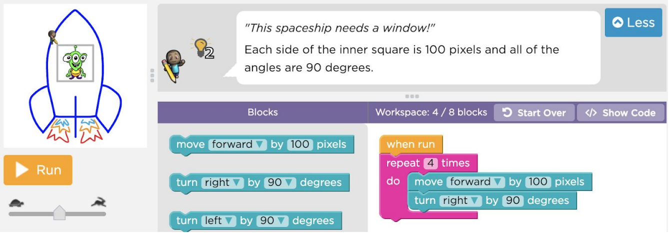
You can alway see how levels are classified in a lesson on the course overview page (e.g., studio.code.org/s/coursea) by selecting the expanded view (see the icon to the left)

Practice

10-12 Practice

10 11 12

Practice puzzles are the most common type of online activity found in CS Fundamentals lessons. Students are given explicit instructions about a task to perform and provided with a toolbox of programming commands and hints. These puzzles are automatically validated which means that students receive instant feedback about the accuracy of their code. Students can typically just be expected to work through these progressions, either with a partner or at their own pace.



"This spaceship needs a window!"

Each side of the inner square is 100 pixels and all of the angles are 90 degrees.

Blocks | Workspace: 4 / 8 blocks | Start Over | Show Code

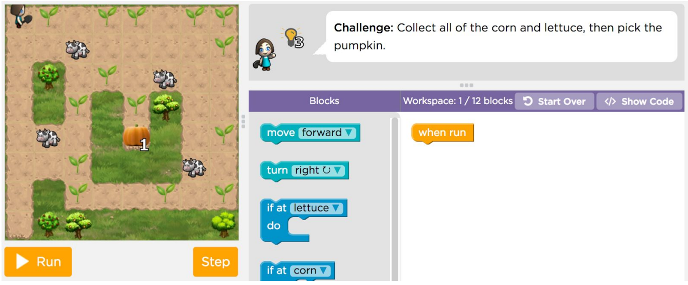
```
move forward by 100 pixels
turn right by 90 degrees
turn left by 90 degrees
```

```
when run
repeat 4 times
do  move forward by 100 pixels
    turn right by 90 degrees
```

Challenge

9 Challenge

Most of our skill building lessons contain one “challenge” puzzle near the end of the lesson series. Challenge puzzles are intended to inspire students to try new things with the concepts they are learning. It will test their persistence, highlight misconceptions, and hopefully lead them to the “ah-ha” moment that educators love. If the lesson you are teaching contains a challenge puzzle, consider letting students know in advance. Encourage them to persevere and understand that it may take additional time and effort to complete these tasks. In case it is needed, all challenge puzzles have a “Skip” button which allows students to move on. Be sure to set expectations with your class around how to engage with this type of activity.



Challenge: Collect all of the corn and lettuce, then pick the pumpkin.

Blocks | Workspace: 1 / 12 blocks | Start Over | Show Code

```
move forward
turn right ↻
if at lettuce
do
if at corn
```

```
when run
```

Video

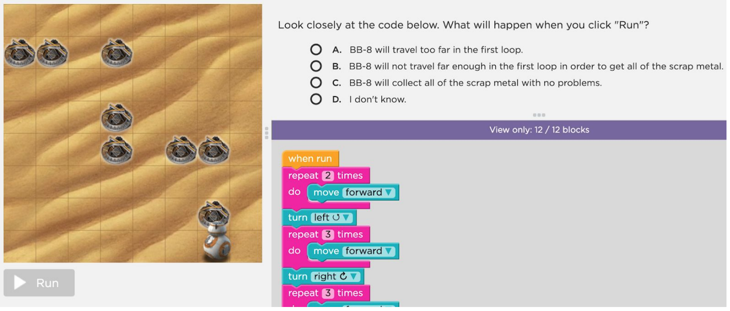
1 The Harvester

All CS Fundamentals courses feature great videos designed to introduce tools or to help explain new concepts. Featuring a diverse set of speakers who also share their own connections to the growing field of computer science, these videos are meant to be viewed by students. Depending on the lesson and your classroom organization, you may wish for students to watch them on their own or show them to the entire class all at once.

Prediction

13 Predict

Instead of writing or changing code, as they do in practice and challenge puzzles, in prediction puzzles students are asked to read a provided program and answer a question about it. The question must be answered before the student may press the “Run” button. While it is possible to use these questions as a form of assessment, keep in mind that they are often placed at the start of a lesson as a way to get students questioning things before they are even formally shown how something new works.



Look closely at the code below. What will happen when you click "Run"?

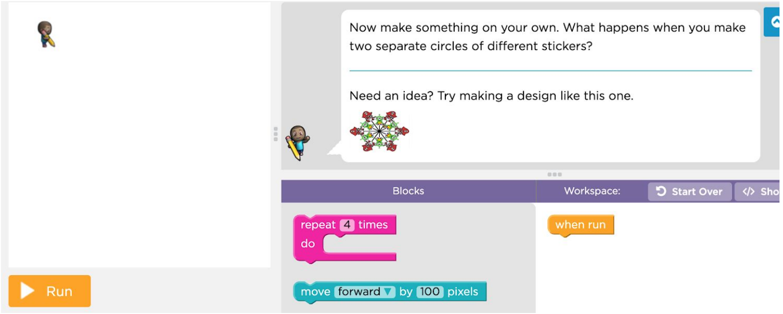
○ A. BB-8 will travel too far in the first loop.
○ B. BB-8 will not travel far enough in the first loop in order to get all of the scrap metal.
○ C. BB-8 will collect all of the scrap metal with no problems.
○ D. I don't know.

View only: 12 / 12 blocks

```
when run
repeat 2 times
do  move forward
turn left ↺
repeat 3 times
do  move forward
turn right ↻
repeat 3 times
```

Free-Play

Many of our skill-building and application lessons end with a free-play level. This is not only a great place to point students who are progressing quickly, but a place for all students to really express their creativity. While students are not actually required to write any code in these levels, doing so will allow them to apply what they have learned in a way that is more meaningful to them. Encourage your students to take their time here.

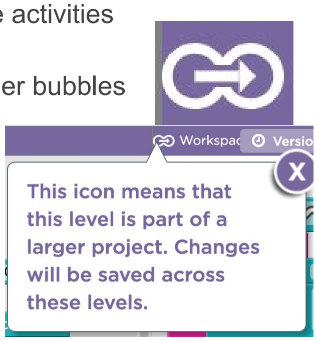


Now make something on your own. What happens when you make two separate circles of different stickers?

Need an idea? Try making a design like this one.

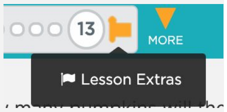
Mini-Project

Mini-projects are made up of a series of online activities in which students will write code to build something gradually over the course of the main activity of a lesson. These activities look similar to practice puzzles but have a few important differences. Most importantly, any code that you write in one bubble is carried over to the other bubbles within the same mini-project progression. This means that students (and the teacher) will see the same program when clicking ahead (or behind) within the project. Because these activities are designed to encourage creativity, expect each student to create something a little different. The open-ended nature of these projects means that Code Studio will not offer immediate feedback about the correctness of the student's work. Pressing the orange "Finish" button will allow the student to move on to the next step and turn the associated bubble at the top of Code Studio green.



This icon means that this level is part of a larger project. Changes will be saved across these levels.

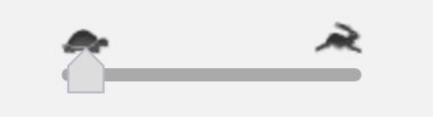
Lesson Extras



Because of the structure of concept chunks and the flow between unplugged, skill-building, and application lessons, it is usually ideal to keep the entire class together on the same lesson instead of letting students skip ahead to new concepts. For that reason, we have provided an area at the end of each lesson where students can use the concepts that they've learned in new and interesting ways. These puzzles are considered "optional." They are a sandbox to keep students who finish early challenged and working on tasks relevant to the core CS concept being taught. In addition, the Lesson Extras can be used to add an additional day onto skill-building lessons, offering the opportunity for students to come back and create personal and authentic projects with their newest concepts.

Code Studio Debugging Features

Some of our puzzles feature special tools to help students debug code on their own.

Debugging Feature	How It Works
Step Button *Found in Maze puzzles* (Includes skins: Angry Birds, Ice Age, Plants vs. Zombies)	With the “Step” button, it is possible to go through a program block by block to see what happens each step of the way. This is a helpful tool when the code moves too quickly to understand where things get off course. To use the “Step” button, simply click on “Step” instead of “Run.” The code will run exactly one block before coming to a rest again. To continue through the code, keep pressing “Step” until you have completed your program or found your bug.
Speed Slider  *Found in Artist puzzles* (Includes Frozen skin)	In Artist puzzles, the “Step” button is replaced by a speed slider. For a similar effect, try moving the slider to the far left and watching the artist go through each step very slowly.

Implementation and Planning

This section offers suggestions for implementing a CS Fundamentals course in an elementary school classroom. CS Fundamentals courses are designed for flexibility in implementation and have been successfully used in a variety of formats, including in the classroom alongside other subjects, as a rotating special, in computer lab time, or as stations.

Lesson Pacing

In general, all students should move lesson to lesson at a pace set by the teacher. This is easy to do in a teacher-led unplugged lesson but can be trickier in skill-building and application lessons in which students move through activities at different speeds. The two challenges here are in recognizing when a class is ready to move on and knowing how to make sure all students are working on something appropriate for them within the same concept chunk.

Unplugged Lessons: These lessons involve the full class learning a concept together and are a great way to kick off a concept together as a whole class.

Skill-Building Lessons: These lessons contain optional lesson extras as well as challenge puzzles that students can actually choose to skip. If a student skips a challenge puzzle or lesson extra but moves through the rest of the lesson quickly, encourage them to go back and try the challenge and lessons extras. It is not necessary for all students to complete all challenges and lesson extras before moving the class onto the next lesson. If a student's progress shows that they haven't finished a full lesson yet, consider seeing if they are stuck at a challenge puzzle or something more basic and responding appropriately. For some concept chunks there are multiple skill-building lessons. When this is the case, it is not required that all students do every level of every skill-building lesson. You should use your judgement about when students are ready to move on to the application lessons or to the next concept chunk.

Application Lessons: Due to the more open-ended nature of these lessons, students can actually pass some levels in these lessons just by clicking "Finish." This should be discouraged as these lessons are one of the best ways for students to demonstrate their learning. They are also a great place to encourage creativity and give students space to apply the concepts they've learned in new ways. If some students move through an application lesson quickly, help them brainstorm ideas or encourage them to get feedback from peers.

Projects: In addition to the open-ended project lesson at the end of most courses, it is also possible for students to create a new project at any time by clicking the "Create" button in the top right corner of Code Studio. If a student has already shown good understanding of a concept, consider having them build a new project using what they know rather than just moving onto the next lesson ahead of the class.

Scheduling The Lessons

The simplest and easiest way to schedule a CS Fundamentals lesson is to dedicate one 45-minute period to each lesson. However, if this does not seem realistic for your situation, it is possible to teach the courses, even with less time. In the CS Fundamentals Curriculum Overview section, we explored how Courses A-F are further divided into concept chunks. These groups of lessons are connected through shared concepts. However, you are not required to teach every lesson or even every chunk. If a concept chunk consists of three or more lessons, consider grouping some lessons into one class period. For example, it may be possible to teach an abridged version of an unplugged lesson in 20 minutes, or you may give students a choice about which skill-building lesson to complete first rather than giving everyone time to complete all skill-building lessons. If you do need to shorten things, be sure to end with an experience in which students get to take ownership of something they make themselves, either in an application lesson or a project.

Approach to Teaching in Common Classroom Scenarios

Here are implementation tips for common situations in elementary schools:

Everyone doing the lesson together

Keeping the class together on the same lesson helps to build a sense of community and prevents struggling students from feeling left behind by their peers. Make sure students feel empowered to travel at their own pace within a lesson and consider what you want to do with students who finish early. See the “lesson pacing” section above for more details and tips.

1:1 computers

Even if you have 1:1 computers, consider grouping students up for pair programming. This setup allows students to gain insight into the problem-solving processes of their peers while helping them to develop collaboration and communication skills.

Always in the computer lab

If your class is regularly held in a computer lab, look for rooms to “visit” for unplugged lessons (like the library or the gym). This will give students room to spread out and feel like they are learning authentically rather than trying to “make do” in the available space.

Only a few computers

Some classrooms have a small number of computers set up in one area, and teachers use these as activity centers. This can be still effective once the class has already gone through the unplugged lessons for a concept together.

Limited time

If you are short on time, choose a concept and teach it thoroughly. In elementary school, the main goal is to teach students that they are capable of learning computer science. If you ditch a deep dive on concepts in favor of a shallow introduction, students might be left feeling as if they don’t understand any of it.

Guide for CS Fundamentals teachers during school closures

In response to school closures due to COVID-19 pandemic, many Code.org classrooms are moving to using our CS Fundamentals courses in a virtual setting or giving their students activities to keep learning at home. Note that the following guidance also applies to longterm school closures caused by other incidents schools may face (e.g., inclement weather).

Resources to keep learning

Computer science is a great way to add some fun to extended time at home. However, the learning will take a different form when some kids are engaged regularly, some have only a little time, some have computers, and some don’t. Rather than trying to teach a synchronous class remotely, many teachers will be giving elementary school students activities they can do to maintain their interest as well as providing structure during their time at home.

We are compiling a list of activities on code.org/athome that are ideal to share with students or families interested in doing some computer science on their own. This list includes activities for students with no computers or internet access.

Teaching CS Fundamentals Remotely

We designed the lessons in CS Fundamentals for use in a classroom with an actively engaged teacher, but many lessons can adapt well to at-home learning. Learning with CS Fundamentals does require an internet-connected device with a modern browser, something we realize not all students can access right now. We have modifications available for remote teaching.

Modifications for Virtual and Socially-Distanced Classrooms

Each of Courses A-F has a modifications document, below, designed to help you create a plan to implement as many of the lessons as possible for your situation. Suggestions are provided for teacher preparation steps, teaching strategies, callouts for tricky lessons, and specific modifications for these lessons.

Modification documents are available on CSF course landing pages.

You can track your students' progress to see where they are. You may be able to use resources like video conferencing or email to assist students and provide feedback.

Where can I get help?
You are highly encouraged to share any questions or insights on our CS Fundamentals Forum where you can connect with other teachers. You can also email us at support@code.org. We are here to help!

We also recommend you consider CSTA's Resources to Support Teaching During COVID-19 for an extensive set of options for continuing to teach computer science during school closures.

Tech Requirements and Required Materials

Technical Requirements

A computing device and an Internet connection

We work hard to build an environment that supports all modern web browsers on desktops and mobile devices. This includes Internet Explorer 11+ and the latest versions of Firefox, Chrome, Safari, and Edge.

Our instructional videos may be affected depending on your school's internet filters. If YouTube is blocked at your school, our video player will attempt to use our non-YouTube player instead. For more details about the IT requirements for accessing and playing our embedded videos, see our IT requirements page at code.org/educate/it. We've made all of our videos available for download using a link located in the bottom corner. If all fails, some videos have a "Show Notes" tab that provide a storyboard equivalent of the video.

Required Materials / Supplies

One potentially significant cost to consider when teaching this course is printing. Many lessons have handouts that are designed to guide students through activities. While it is not required that all of these handouts be printed, many were designed to be printed, and we highly recommend printing when possible.

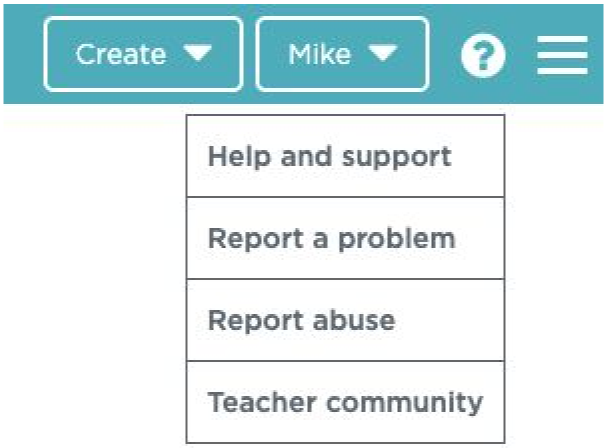
Beyond printing, some lessons call for typical classroom supplies and manipulatives, such as student journals, poster paper, markers, colored pencils, scissors, scrap paper, glue or tape.

The following items are called for in specific lessons as listed:

Course	Lesson	Materials
Course A	Marble Run (optional activity)	Kid friendly marbles or round cereal (1-5/group)*
Course C	Lesson 3: My Robotic Friends Jr.	Plastic cups (10/group of 2-3)*
	Lesson 7: Creating Art with Code	Optional - Protractors (1/student)
	Lesson 8: Binary Bracelets	Markers. Optional – 18 black/18 white beads, 1 pipe cleaner per student
	Lesson 9: My Loopy Robotic Friends Jr.	Paper cups (20/group of 4)
Course D	Lesson 7: Dance Party	Optional - Headphones (1/student)
	Lesson 10: Conditionals with Cards	Deck of cards or something similar (1/group of 4-6)*
	Lesson 16: Binary Images	Optional - Groupings of opposite items to display to students
Course E	Lesson 10: Digital Sharing	Smartphone or tablet
Course F	Lesson 6: The Power of Words	Colored Pencils, string the length of the classroom
	Lesson 7: Envelope Variables	Envelopes (1-4/group of 2-4)
	Lesson 14: For Loop Fun	Dice (3 dice/group of 2-4)*

**These items can easily be re-used between multiple classes*

Getting Help



The curriculum is completely free for anyone to teach anywhere. For support, click on the question mark in the upper right-hand corner of the website.

Here, you'll find our "Help and support" forum where you can email us or find how-to articles. You'll also see a link to our "Teacher community" forums where you can connect to other teachers for support, teaching tips, or best practices.

When you're in a puzzle, you'll see an additional "Report a problem" link for that puzzle. Thank you for helping us find and fix any issues.

Thanks and Acknowledgements

Launched in 2013, Code.org is a non-profit organization dedicated to expanding access to computer science and increasing participation by women and underrepresented students of color. Our vision is that every student in every school has the opportunity to learn computer science. We believe computer science should be part of core curriculum, alongside other courses such as biology, chemistry, or algebra.

Code.org increases diversity in computer science by reaching students of all backgrounds where they are — at their skill-level, in their schools, and in ways that inspire them to keep learning. Read about our efforts to increase diversity in computer science at code.org/diversity. In order to support this vision of diverse and meaningful access to computer science, Code.org has developed a full pathway of learning opportunities that span K-12. The CS Fundamentals curriculum is specifically designed to meet the needs of elementary school students and teachers along that pathway.

As always, it is thanks to our generous donors that we were able to develop and offer this curriculum at no cost to schools, teachers, or students: Microsoft, Infosys Foundation USA, Facebook, Omidyar Network, Google, Ballmer Family Giving, Ali and Hadi Partovi, Bill Gates, The Bill and Melinda Gates Foundation, BlackRock, Jeff Bezos, John and Ann Doerr, Mark Zuckerberg and Priscilla Chan, Quadrivium Foundation, Amazon Web Services, The Marie-Josee and Henry R. Kravis Foundation, Reid Hoffman, Drew Houston, Salesforce, Sean N. Parker Foundation, Smang Family Foundation, Verizon.

Appendix A: Worksheets for Intro Workshop

This appendix contains all necessary handouts for lessons that could be explored in the Intro Workshop. Consult the table of contents below for page numbers:

Unplugged Lesson Implementation Planning Guide

Course ____________________ Lesson __________

Make a Plan

Goal
You have thought about and made a detailed plan for how you might teach an **unplugged lesson** in a way that leverages the CS Fundamental instructional practices and supports students

What's happening	Where can this happen in the lesson?
Instructional approaches to use in this lesson What instructional techniques or approaches do you want to be sure to use in this lesson? Where do you want to use them?	
Role as the teacher When/where in the lesson do you want the whole class to discuss a topic? What role do you plan to take and when?	
Classroom environment and student interaction When do you want students to interact during this lesson? What should that look like?	

What's happening	Where can this happen in the lesson?
Assessing student learning in the lesson How can you assess if students have learned what they need to learn during this lesson?	
Connecting to computer science Where do you think students might need a push to see the connection between this unplugged activity and computer science? How do you plan to support that?	
Making connections Where is there space in this lesson to make connections to the real world/other things students are learning in school?	

Plugged Lesson Implementation Plan

Course ____________________ Lesson ___________

Make a Plan

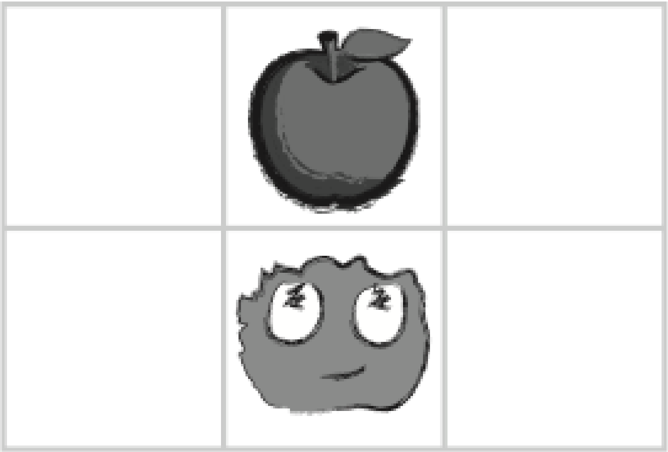
Goal
You have thought about and made a detailed plan for how you might teach the lesson in a way that leverages the CS Fundamental instructional practices and supports students

What's happening	Where can this happen in the lesson?
Connecting to Unplugged lesson How will you scaffold and support students in seeing the connection between the previous unplugged lesson and this plugged lesson?	
Instructional approaches to use in this lesson What instructional techniques or approaches do you want to be sure to use in this lesson and where do you want to use them?	
Role as the teacher When/where in the lesson do you want the whole class to discuss a topic? What role do you plan to take and when?	

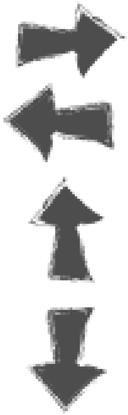
What's happening	Where can this happen in the lesson?
Classroom environment and student interaction When do you want students to interact during this lesson? What should that look like?	
Assessing student learning in the lesson How can you assess if students have learned what they need to learn during this lesson?	
Supporting debugging Where do you think students might need to engage in debugging during this lesson (which levels/activities)? How can you support them with that debugging?	
Making connections Where is there space in this lesson to make connections to the real world/other things students are learning in school?	

Course A Lesson 3 - Happy Maps

1 Happy Map 1

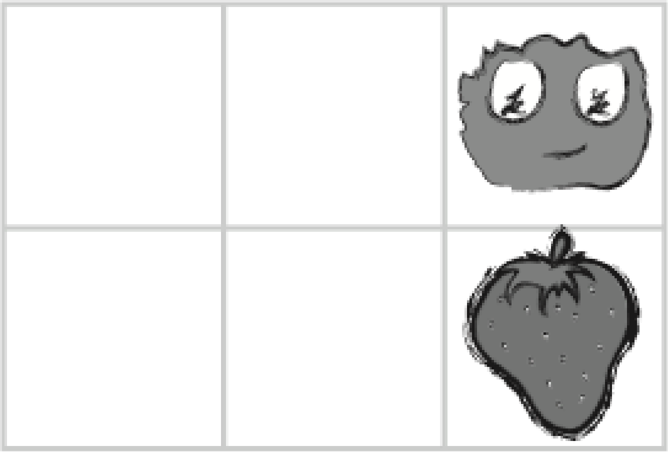


Which way should the Flurb step to get to the supplies?

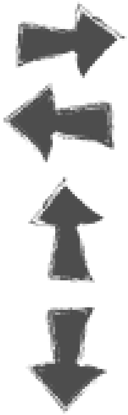


Revision 161003.1a

2 Happy Map 2



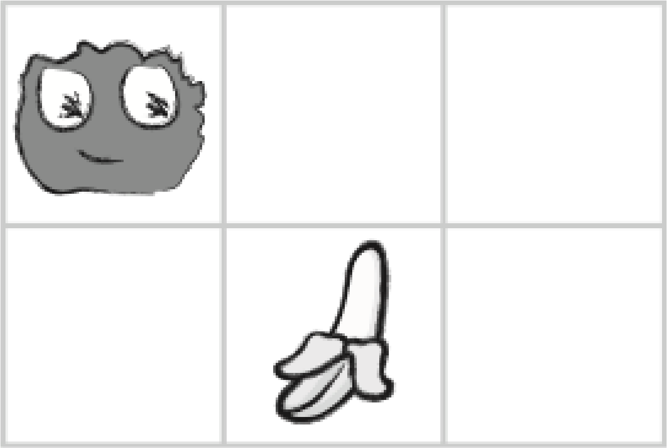
Which way should the Flurb step to get to the supplies?



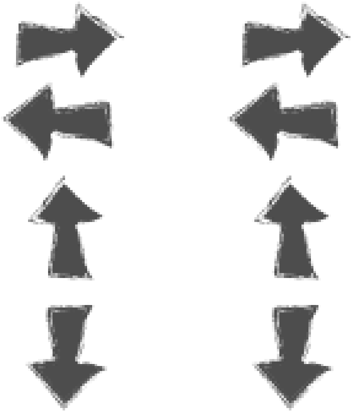
Course A Lesson 3 - Happy Maps

3

Happy Map 3



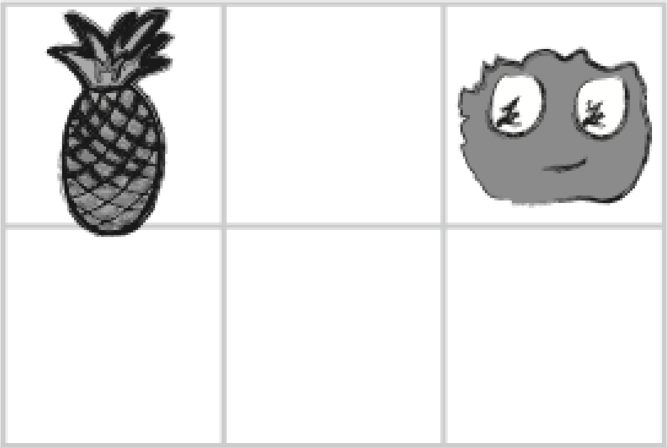
Which two ways should the Flurb step to get to the supplies?



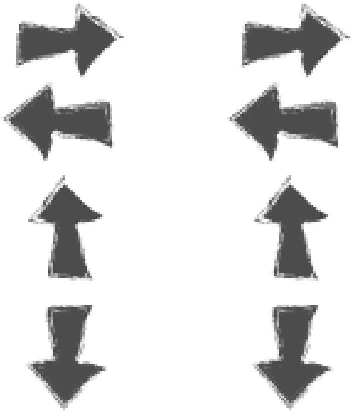
Revision 161003.1a

4

Happy Map 4



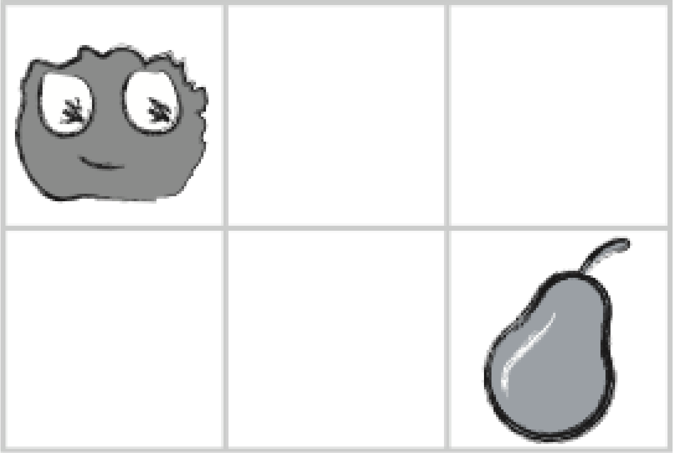
Which two ways should the Flurb step to get to the supplies?



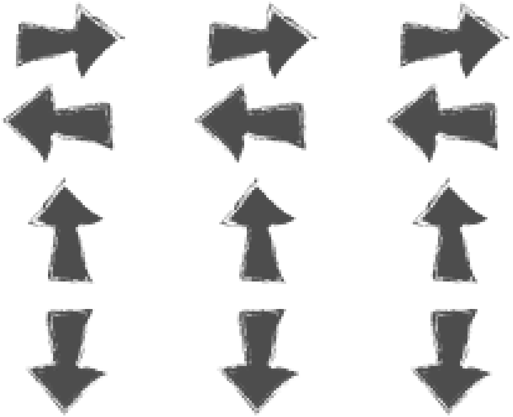
Course A Lesson 3 - Happy Maps

5

Happy Map 5



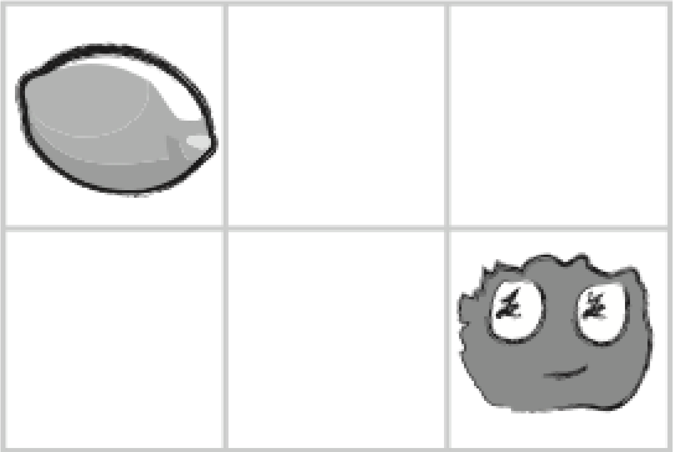
What should the Flurb do to get to the supplies?



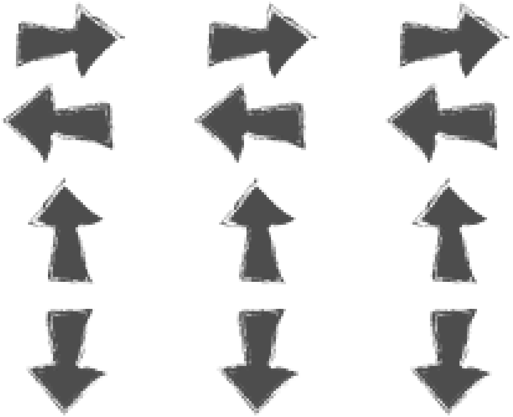
Revision 161003.1a

6

Happy Map 6



What should the Flurb do to get to the supplies?

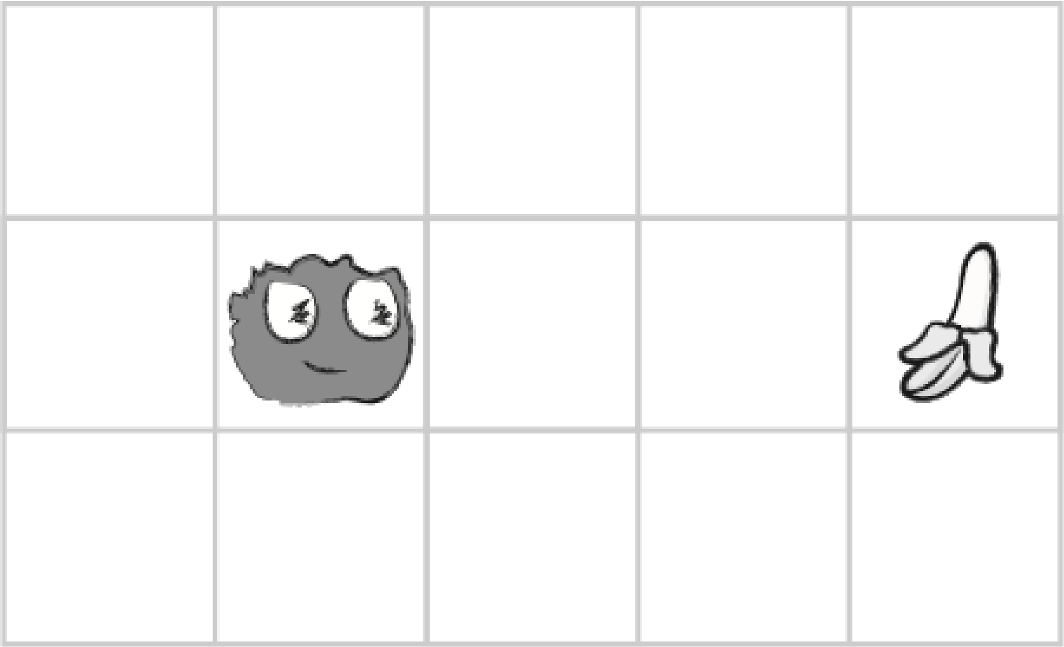


Course A Lesson 3 - Happy Maps

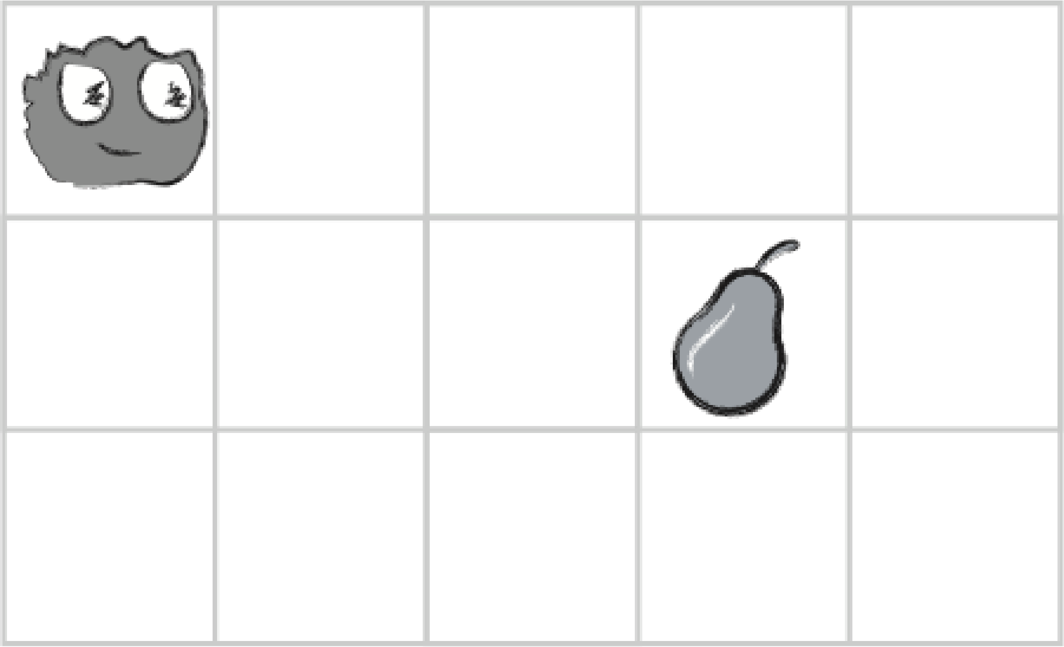
5 Happy Map Blank

Course A Lesson 7 - Happy Loops

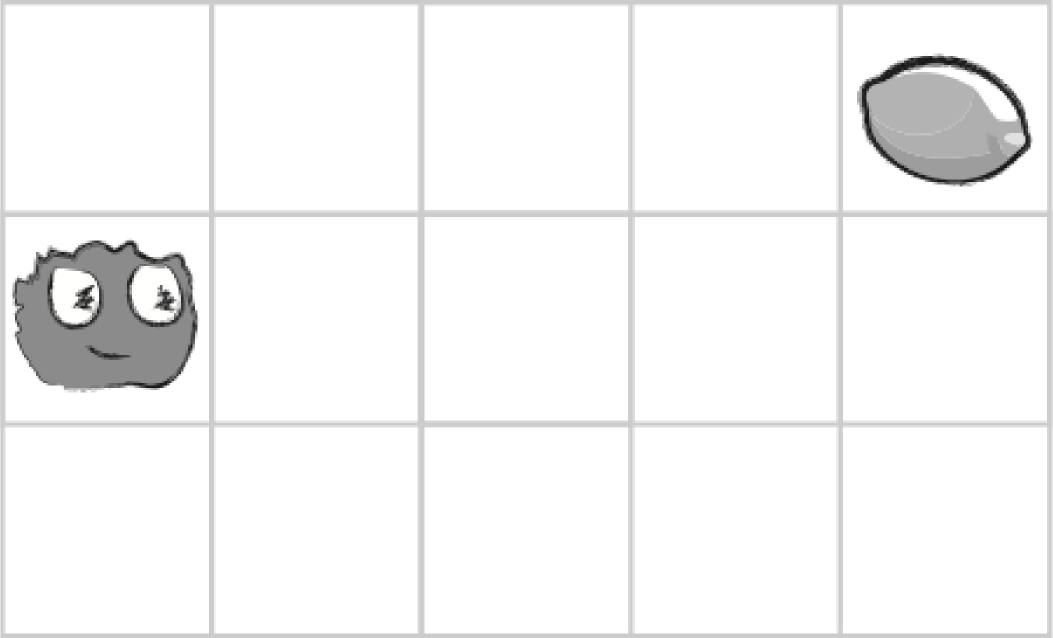
1 Happy Map XL 1



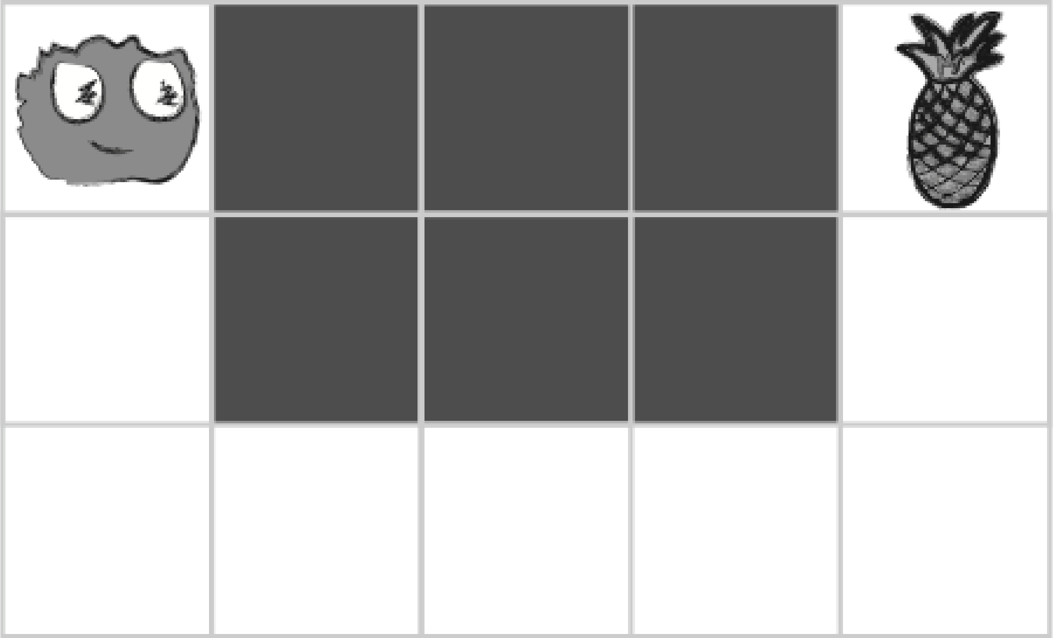
2 Happy Map XL 2



3 Happy Map XL 3



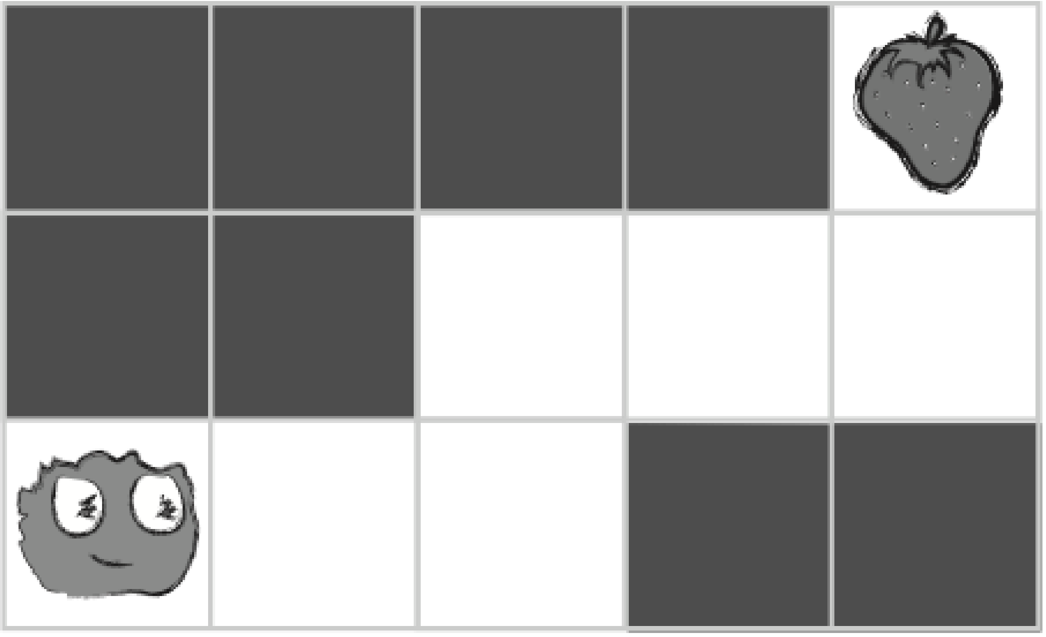
4 Happy Map XL 4



Course A Lesson 7 - Happy Loops

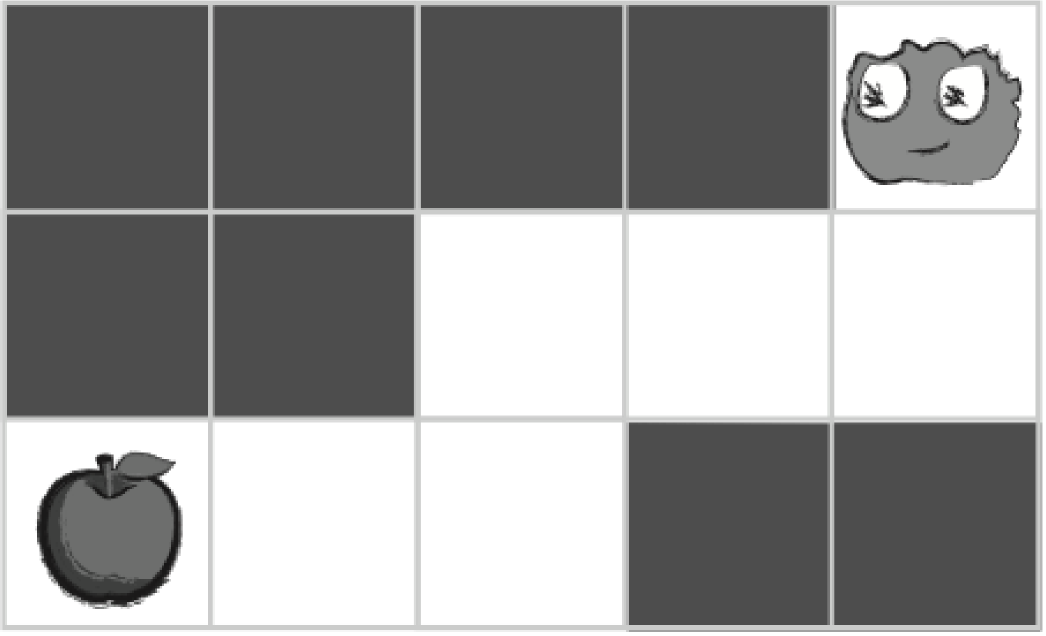
5

Happy Map XL 5



6

Happy Map XL 6



Course A Lesson 11 and Course C Lesson 14 - The Big Event (Jr.)

U

Unplugged

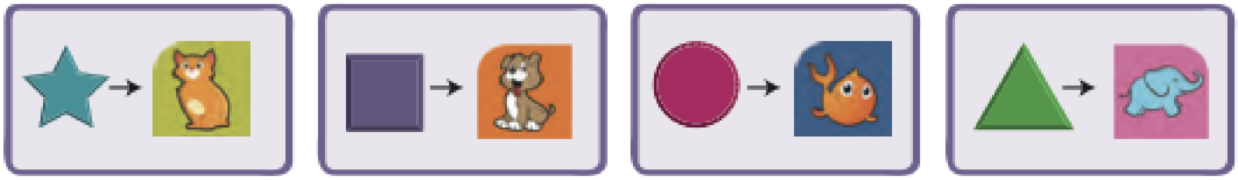
Name: ______________________ Date: ______________

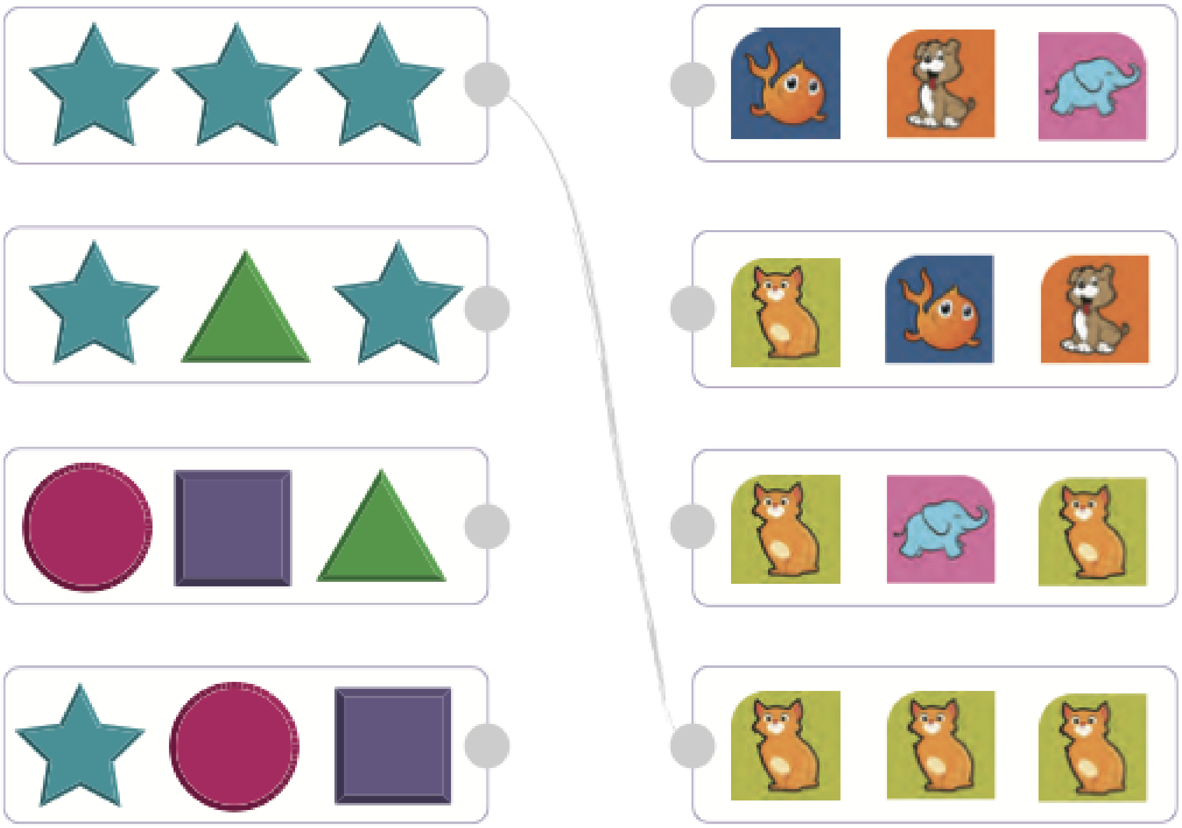
The Big Event

Controlling by Events Assessment

You've been given a magical controller that changes the picture on the frame on your desk.

Take a look below to see what each button does. Can you figure out which series of button events will cause your frame to show the pictures on the right? Draw a line from each set of pictures to the button combination that causes it. The first one has been done for you.





Name: ____________________ Date: ____________

U

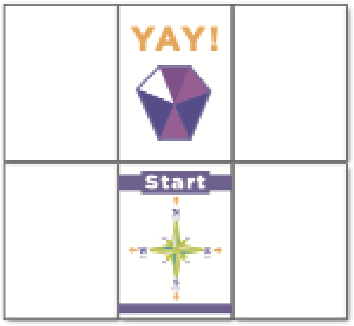
Unplugged

Move It Move It

Debugging

Each of these algorithms has a mistake. Can you find the mistake and cross it out?

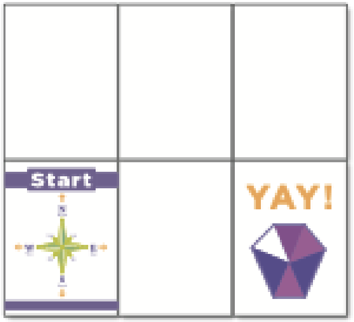
1)



Move North ↑

Move North ↑

2)

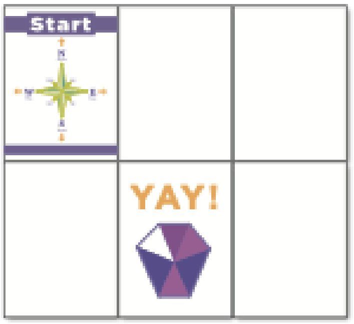


Move East →

Move North ↑

Move East →

3)

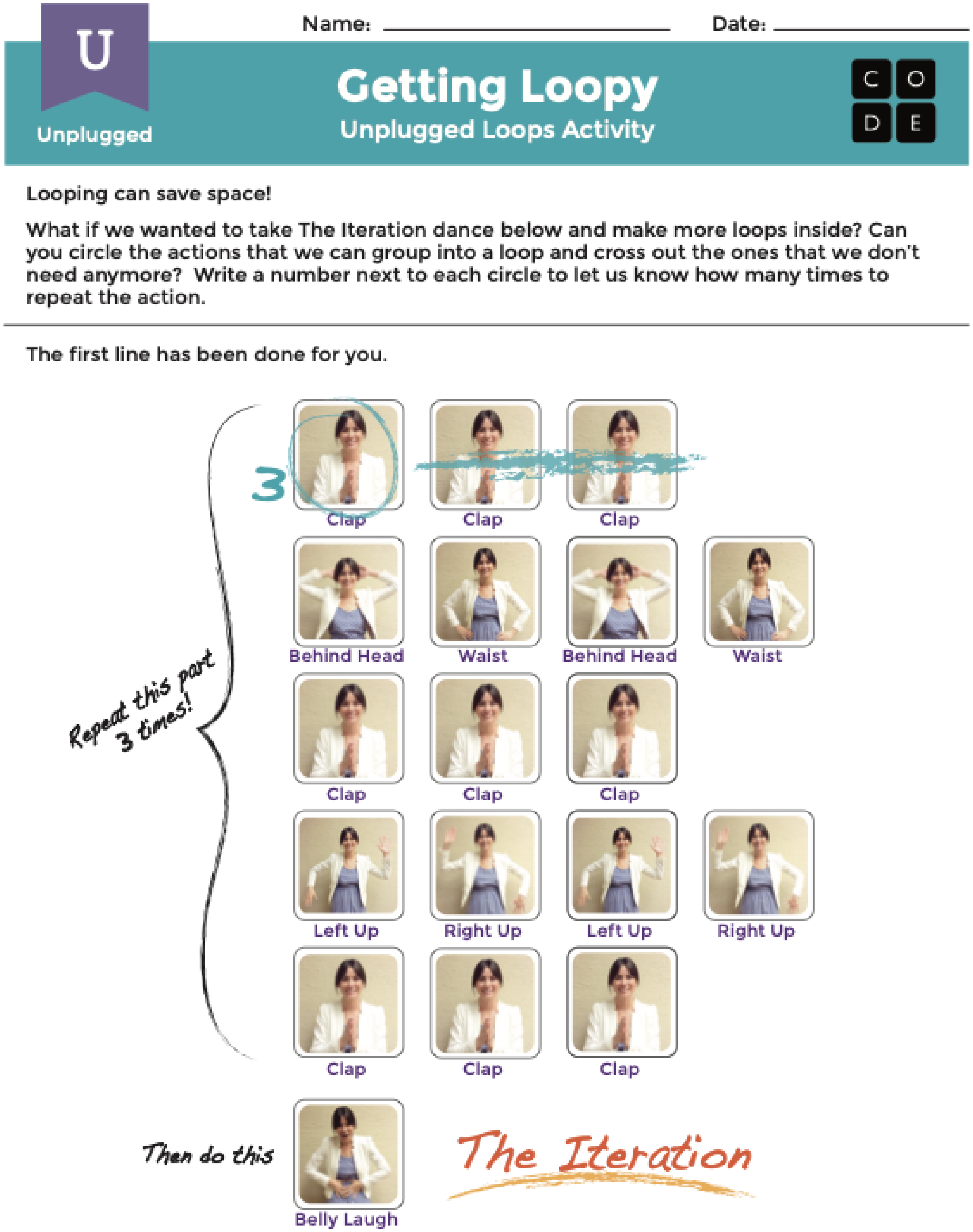


Move South ↓

Move East →

Move South ↓

Course B Lesson 6 - Getting Loopy



Name: ______________________ Date: ______________

U

Unplugged

Getting Loopy

Unplugged Loops Activity

Looping can save space!

What if we wanted to take The Iteration dance below and make more loops inside? Can you circle the actions that we can group into a loop and cross out the ones that we don't need anymore? Write a number next to each circle to let us know how many times to repeat the action.

The first line has been done for you.

Repeat this part 3 times!

3 Clap ~~Clap~~ ~~Clap~~

Behind Head Waist Behind Head Waist

Clap Clap Clap

Left Up Right Up Left Up Right Up

Clap Clap Clap

Then do this

Belly Laugh

The Iteration

Course D Lesson 1 - Graph Paper Programming

Choose one of the images below. Don't let your partner see which one you pick!

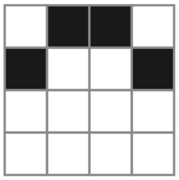


Image 1

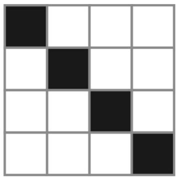


Image 2

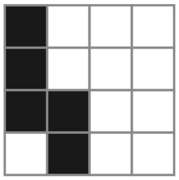


Image 3

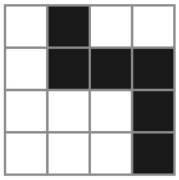


Image 4

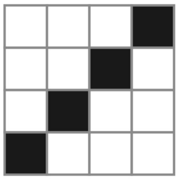


Image 5

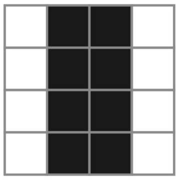


Image 6

1) Write a program. (Use → ← ↑ ↓ ✎)

Step 1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

2) Trade this worksheet with a partner.

3) Draw! Follow your partner's program:

☆			

Play Again!

1. Write a program. (Use → ← ↑ ↓ ✎)

Step 1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

2) Trade this worksheet with a partner.

3) Draw! Follow your partner's program:

☆			

Course D Lesson 1 - Graph Paper Programming Assessment

Name: ______________________ Date: ______________

U

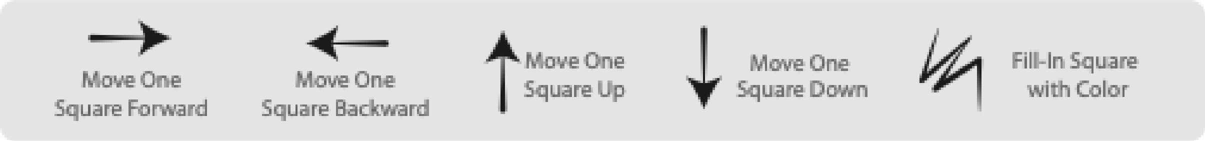
Unplugged

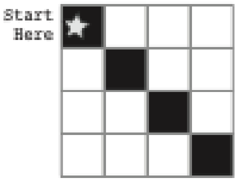
Graph Paper Programming

Assessment Worksheet

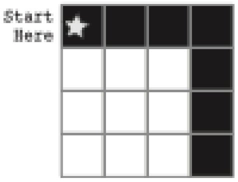
You have just learned how to create algorithms and programs from drawings, and how to draw an image from a program that someone gives to you. During the lesson, you worked with other people to complete your activities. Now you can use the drawings and programs below to practice by yourself.

Use the symbols below to write a program that would draw each image.

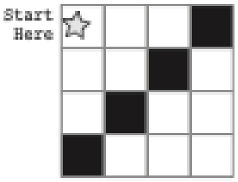




Step 1	2	3	4	5	6	7	8	9	10
Step 11	12	13	14	15	16	17	18	19	20

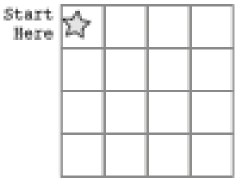


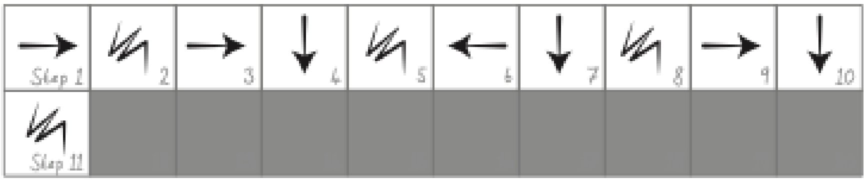
Step 1	2	3	4	5	6	7	8	9	10
Step 11	12	13	14	15	16	17	18	19	20



Step 1	2	3	4	5	6	7	8	9	10
Step 11	12	13	14	15	16	17	18	19	20

Now, read the program below and draw the image that it describes.



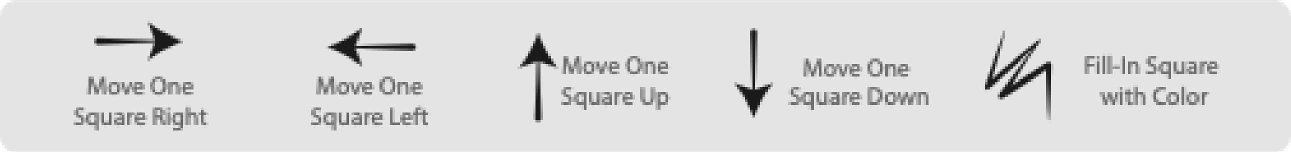


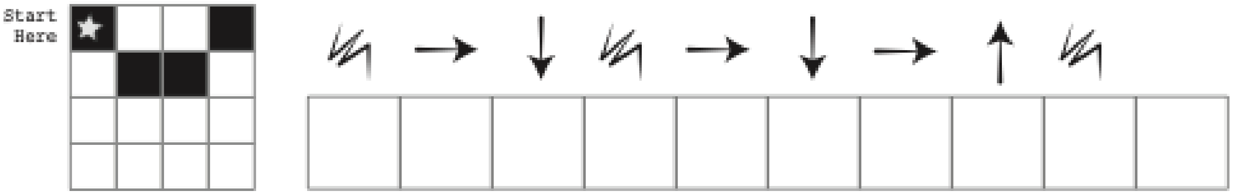
Course D Lesson 3 - Relay Programming Assessment

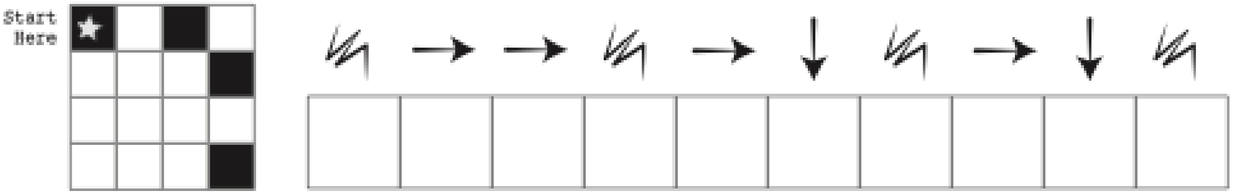
Sometimes when you are coding in groups, someone will make an error that will affect everyone.

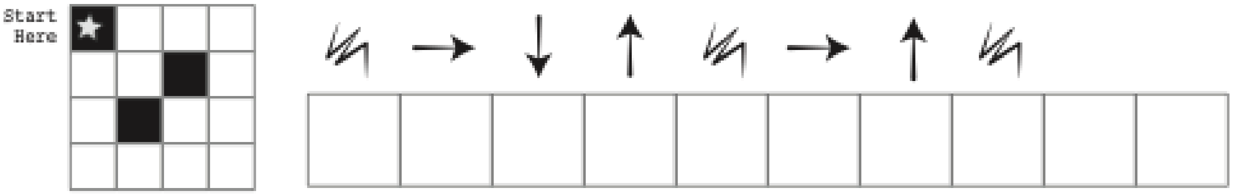
Somebody has already written programs for the images below, but each one has a mistake! Figure out what the programs are *supposed* to look like, and circle the error in each one. Then, draw the correct symbol in the box beneath.

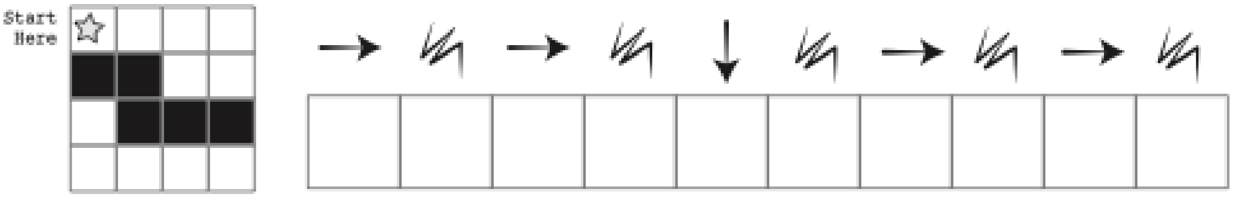
Each program should use the symbols below to draw the image to its left.











Course D Lesson 11 - Conditionals with Cards

U
Unplugged

Name: ____________________ Date: __________

Conditionals with Cards

Assessment Activity

Look at the program below.

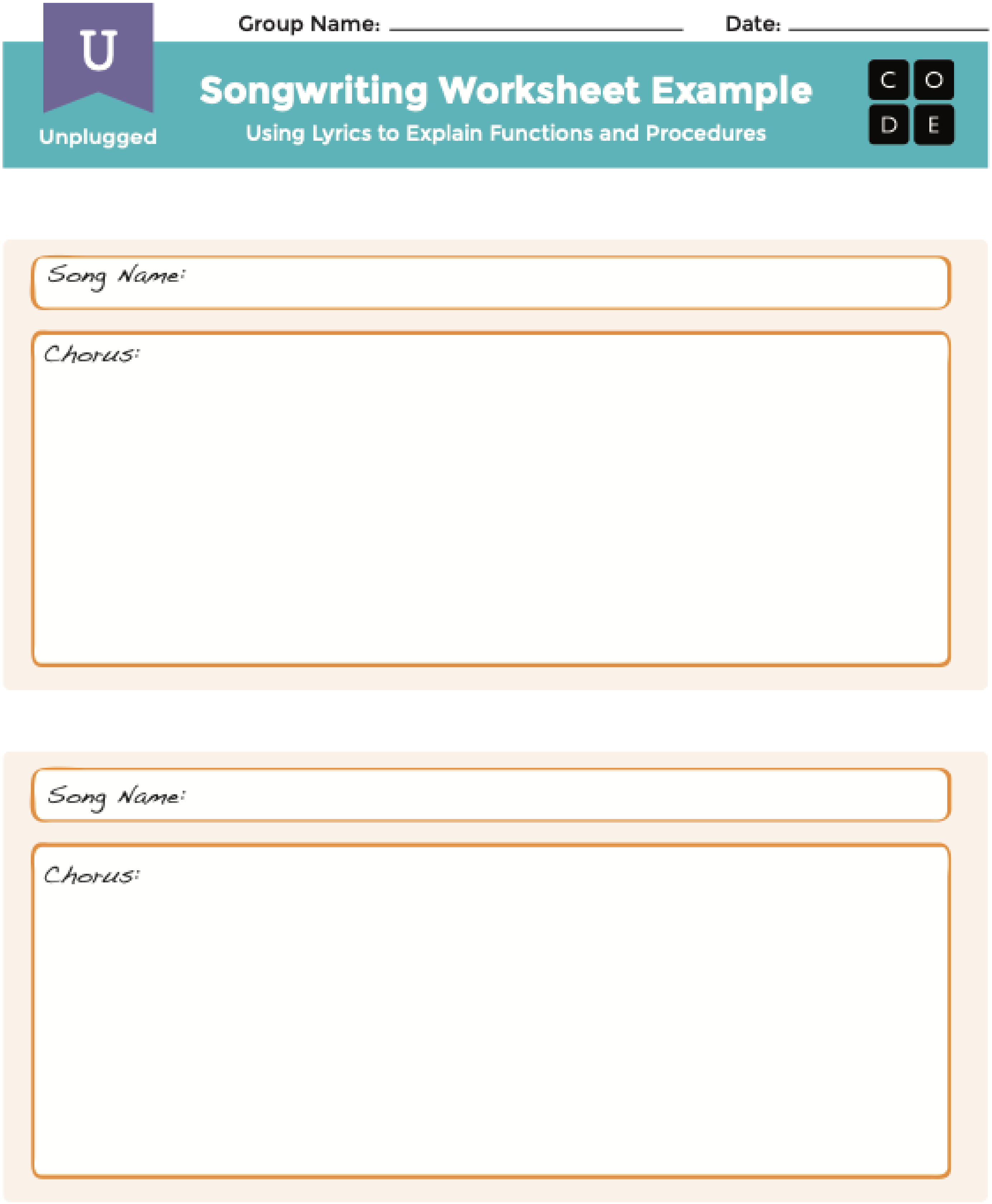
The steps below show each team taking turns to play the Conditionals Game. See if you can figure out what happens for each draw. Write down the score during each round along the way. After three rounds, circle the winner.

```
If (CARD is lower than 5)
    If ( CARD is BLACK)
        Award YOUR team the same
        number of points on the card.

    Else
        Award OTHER team 1 point.

Else
    If ( CARD is HEARTS)
        Award YOUR team 1 point
```

Here's how the game went:

	TEAM #1	END OF ROUND SCORE	TEAM #2	END OF ROUND SCORE
		0		0
ROUND #1	3♠	____	7♥	____
ROUND #2	4♥	____	4♣	____
ROUND #3	9♣	____	5♦	____

Course E Lesson 14 - Functions: Songwriting



Group Name: ______________________ Date: ______________

U

Unplugged

Songwriting Worksheet Example

Using Lyrics to Explain Functions and Procedures

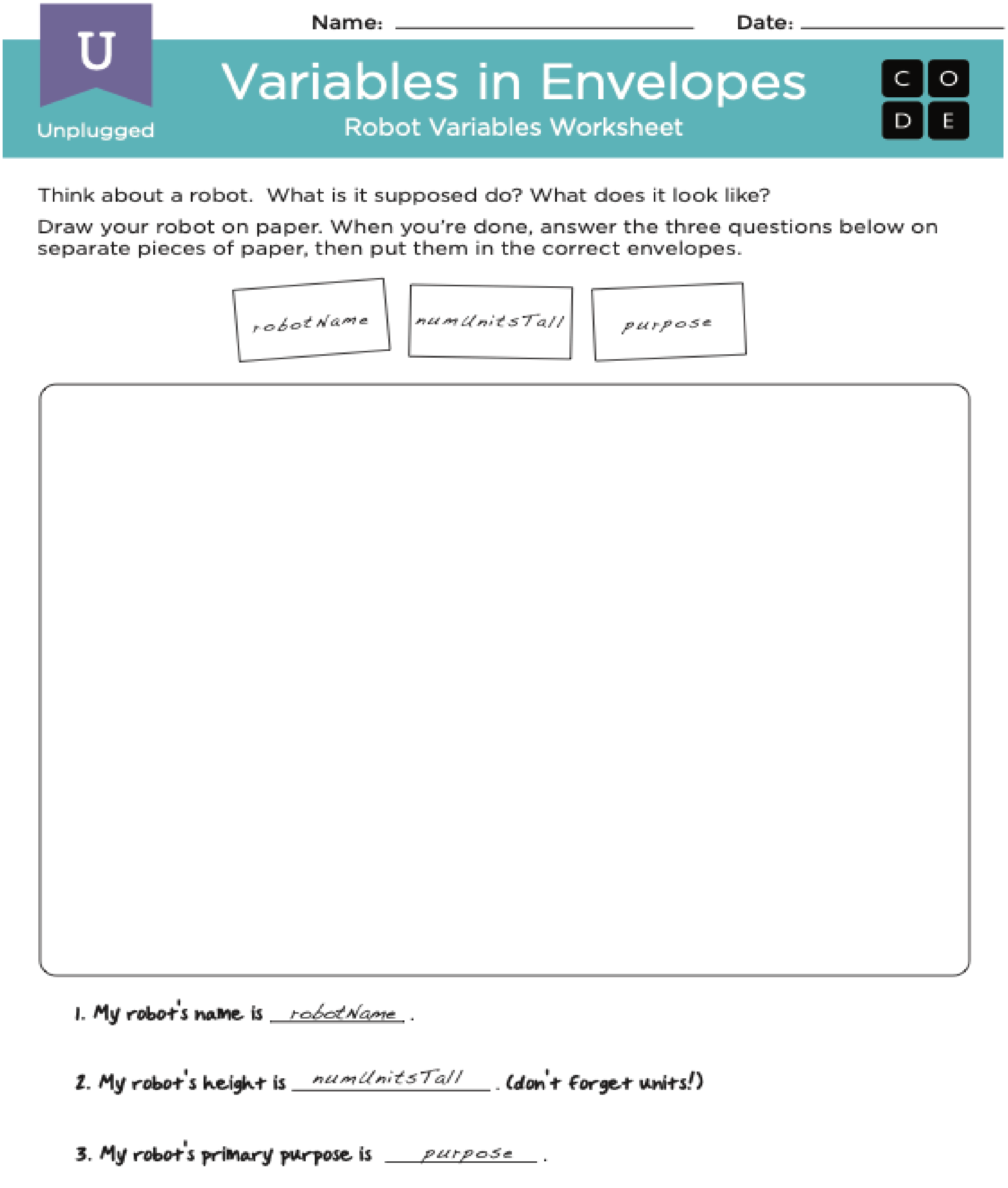
Song Name:

Chorus:

Song Name:

Chorus:

Course F Lesson 7 - Envelope Variables



Name: ______________________ Date: ______________

U

Unplugged

Variables in Envelopes

Robot Variables Worksheet

Think about a robot. What is it supposed do? What does it look like?

Draw your robot on paper. When you're done, answer the three questions below on separate pieces of paper, then put them in the correct envelopes.

robotName

numUnitsTall

purpose

1. My robot's name is robotName.

2. My robot's height is numUnitsTall. (don't forget units!)

3. My robot's primary purpose is purpose.

Course F Lesson 7 - Envelope Variable

Name: ____________________ Date: ______________

U

Unplugged

Variables in Envelopes

Variables Assessment Worksheet

Given the value of each variable envelope, fill-in the blanks to finish the sentence.

color = pink

petalNumber = 22

animal = monkey

bestSport = golf

hobby = coding

When I grow up, I want to own a guard ____________.
animal

I found a flower with ____________ petals, so I picked it.
petalNumber

My dad just painted his house ____________ to match his car.
color

I love ____________ . I do it every evening.
hobby

There is no such thing as ________ rivers, so if you find one, don't swim in it!
color

The best sport in the world is ____________, do you agree?
bestSport

Variable envelopes can also contain number values. Use these envelopes and the provided equations to figure out the magic numbers below.

numOne = 2

numTwo = 5

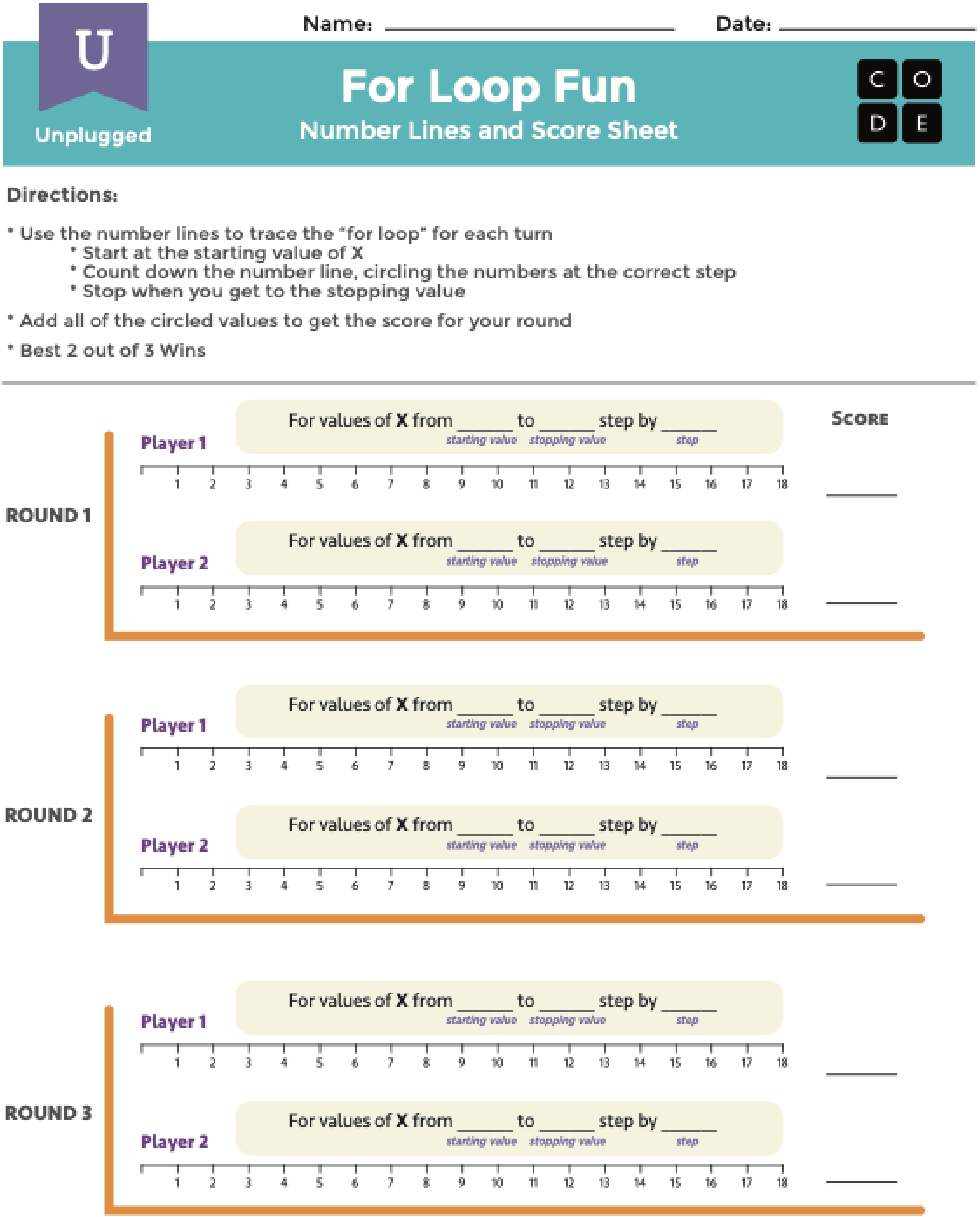
numThree = 7

□ = ________ - ________
magicNumberA numThree numOne

□ = ________ * ________
magicNumberB numTwo numOne

□ = ________ + ________ * ________
magicNumberC numOne numTwo magicNumberB

Course F Lesson 14 - For Loop Fun



U

Unplugged

Name: ____________________ Date: ______________

For Loop Fun

Number Lines and Score Sheet

Directions:

- Use the number lines to trace the "for loop" for each turn
    - Start at the starting value of X
    - Count down the number line, circling the numbers at the correct step
    - Stop when you get to the stopping value
- Add all of the circled values to get the score for your round
- Best 2 out of 3 Wins

Score

ROUND 1

Player 1

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

Player 2

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

ROUND 2

Player 1

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

Player 2

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

ROUND 3

Player 1

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

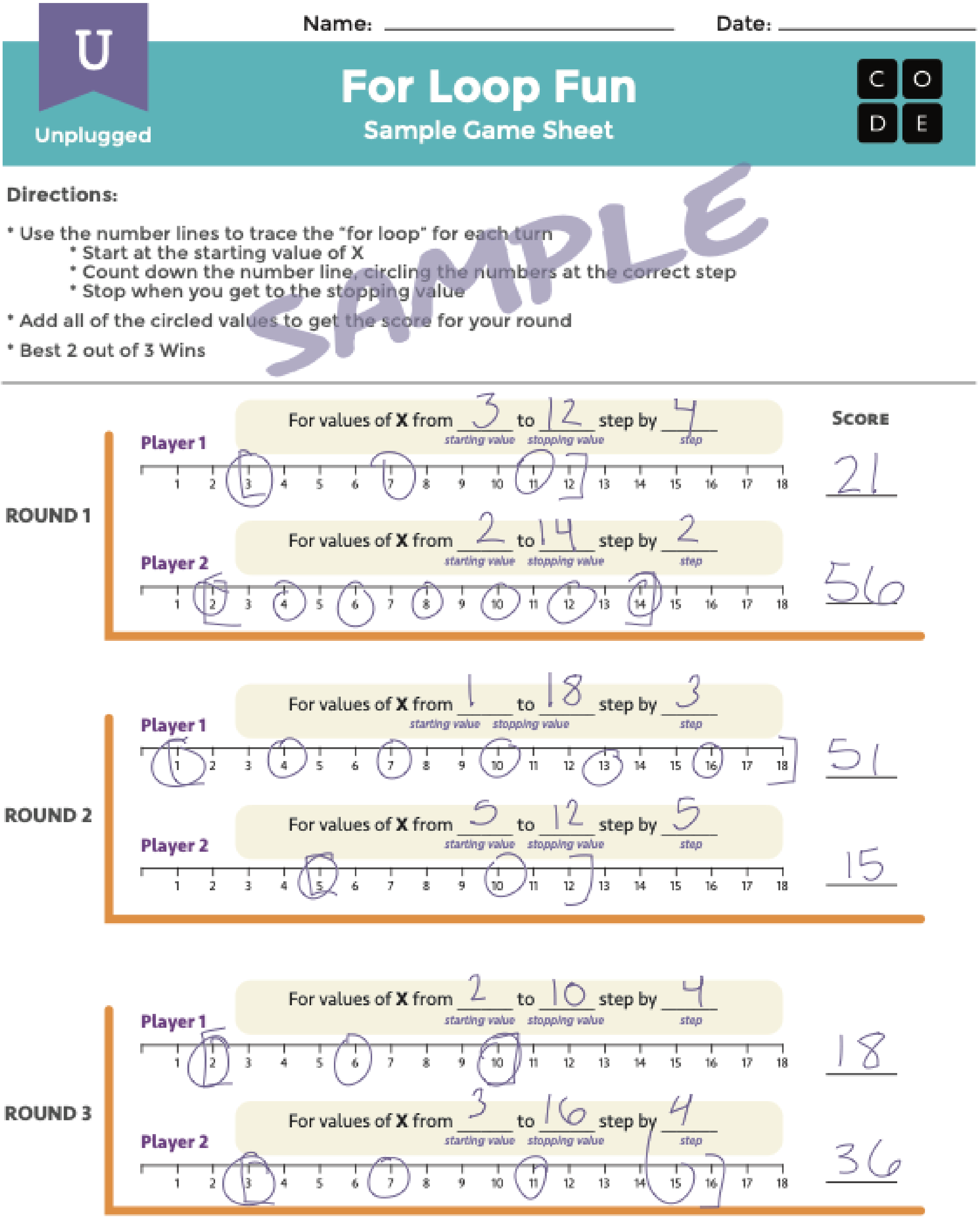
Player 2

For values of **X** from ______ to ______ step by ______
starting value *stopping value* *step*

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

______

Course F Lesson 14 - For Loop Fun



Name: ______________ Date: ______________

U

Unplugged

For Loop Fun

Sample Game Sheet

Directions:

- Use the number lines to trace the "for loop" for each turn
  - Start at the starting value of X
  - Count down the number line, circling the numbers at the correct step
  - Stop when you get to the stopping value
- Add all of the circled values to get the score for your round
- Best 2 out of 3 Wins

ROUND 1

Player 1

For values of **X** from 3 (starting value) to 12 (stopping value) step by 4 (step)

1 2 [✓ 3] 4 5 6 [✓ 7] 8 9 10 [✓ 11] 12 13 14 15 16 17 18

SCORE: 21

Player 2

For values of **X** from 2 (starting value) to 14 (stopping value) step by 2 (step)

1 [✓ 2] 3 [✓ 4] 5 [✓ 6] 7 [✓ 8] 9 [✓ 10] 11 [✓ 12] 13 [✓ 14] 15 16 17 18

SCORE: 56

ROUND 2

Player 1

For values of **X** from 1 (starting value) to 18 (stopping value) step by 3 (step)

[✓ 1] 2 3 [✓ 4] 5 6 [✓ 7] 8 9 [✓ 10] 11 12 [✓ 13] 14 15 [✓ 16] 17 18

SCORE: 51

Player 2

For values of **X** from 5 (starting value) to 12 (stopping value) step by 5 (step)

1 2 3 4 [✓ 5] 6 7 8 9 [✓ 10] 11 12 13 14 15 16 17 18

SCORE: 15

ROUND 3

Player 1

For values of **X** from 2 (starting value) to 10 (stopping value) step by 4 (step)

1 [✓ 2] 3 4 5 [✓ 6] 7 8 9 [✓ 10] 11 12 13 14 15 16 17 18

SCORE: 18

Player 2

For values of **X** from 3 (starting value) to 16 (stopping value) step by 4 (step)

1 2 [✓ 3] 4 5 6 [✓ 7] 8 9 10 [✓ 11] 12 13 14 [✓ 15] 16 17 18

SCORE: 36

Appendix B: Worksheets for Deep Dive Workshop

If you are a US teacher and you'd like to attend a free training on our K-5 Computer Science curriculum, you can find links to local workshops by visiting https://code.org/professional-development-workshops.

Session: Scavenger Hunt

Instructions for this session

The goal of this session is to give you space to learn about pieces of the website and the organization of Code.org resources with which you might not currently be familiar. This activity is broken into three scavenger hunts that progress in order (e.g., you need to know the things in hunt 1 before moving on to hunt 2). If you have some comfort navigating the code.org resources already, feel free to start the hunt on whatever category covers topics with which you are unfamiliar. If you feel very comfortable with all topics, partner up with someone new and support them!

1. Creating and managing sections:

- Creating a classroom section
- Add students to your section
- Changing a section's assigned course
- Moving students between sections

2. Navigating and accessing course materials:

- Hiding and showing lessons for students
- Accessing lesson plans

3. Viewing and assessing student work:

- Viewing student progress and work
- Viewing sample solutions

Resources

All of the topics explored in this scavenger hunt have support articles associated with them. If at any time you want help finding answers, see the support resources linked at the top of the shared notes ("code.org resources") or visit support.code.org and use the search box to find what you need.

1. Creating and managing sections

Category	What to Find	Answer / notes to self
Creating a classroom section	There are 4 types of account logins to choose from when creating a section. What are they?	
	When might you use each of the login types?	
	When setting up a classroom section, there is an option to enable "lesson extras." What are those?	
	Pause and practice: If you haven't already, go create a section of your own for your students using the login type and course that best suits their needs.	
Add students to your section	Which login types require you to add students to the section yourself?	
	For "personal logins" (in which students use their own email addresses), where do students need to go to type in your section code?	
	BONUS: Where can you see all of the sections you have JOINED on code.org?	

Changing a section's assigned course	If you decide you want to teach a section of students a CS Fundamentals course that is different from what you originally assigned it, what can you do?	
	When you assign a new CS Fundamentals course to a section, what happens under "my courses" on the dashboard for you and the students in that section?	
	Pause and Practice: If you haven't already, grab a buddy and have them join one of your sections. Once they've joined, practice changing the assigned course and see what happens on their dashboard.	
Moving students between sections	How do you move students from one classroom section to another?	
	Do you have to move students one at a time, or can you move multiple students at once?	
	Pause and Practice: If you haven't already, move your partner to a different section and see what happens.	

2. Navigating and Accessing Course Materials

Category	What to Find	Answer / notes to self
Hiding and showing lessons (for students)	Where do you go to hide and show lessons? How does it work?	
	When you show and hide lessons, are all of your students impacted or just some?	
	Is it possible to hide lessons for someone with a teacher account?	
	Pause and Practice: If you haven't already, practice hiding and showing lessons for one of your sections. As a reminder, you can always toggle on 'student view' to see what the course looks like when lessons are hidden.	
Accessing Lesson Plans	How do you access lesson plans for CS Fundamentals? (There are two ways.)	
	Pause and Practice: If you haven't looked over the curriculum page (which houses all of the lesson plans for a given course), take a minute or two and check in out now.	

3. Viewing and Assessing Student Work

Category	What to Find	Answer / notes to self
Viewing student progress and work	What are the two ways to view student work on code.org?	
	Pause and Practice: Do you have any students in your sections who have made progress that you can view?	
Viewing sample solutions	Where can you find the 'see a solution' button on programming levels?	
	How do you exit the solution once you're looking at it?	
	Pause and Practice: If you haven't already, try out viewing a sample solution for yourself!	

Session: Lesson Implementation Planning

Lesson Implementation Planning Process

Step 1

Choose what type of lesson you want to plan today. Here are recommendations for what type of lesson to plan, based on your experience level:

Fairly new to CS Fundamentals? We recommend you choose one of these lesson types, focusing on the type you haven't spent much time planning in the past:

- **Unplugged** — explore concepts away from the computer
- **Skill-Building** — the style of lesson we saw in the model lesson earlier

Have you been teaching CS Fundamentals **for a while**? We recommend you plan

- **Application** — levels build off of each other up to a final product (typically a game)

Have a **ton of experience** with CS Fundamentals? We recommend you plan

- **End of Course projects** — entirely student directed projects about a topic of the students' choice

Step 2

Choose which course you're going to explore today, based on the grades you teach:

If you teach a **single grade**, choose the course associated with that grade.

If you teach CS Fundamentals to **multiple grades**, choose which course to explore today based on considerations like:

- Which course you have the least experience with
- Which grade you spend the most time with

Step 3

Find your lesson recommendation below by matching up the lesson type and course you want to explore:

	Unplugged	**Plugged —** Skill-Building	**Plugged —** Application	**Plugged —** End of Course Projects
Course A *Kindergarten*	A.7 - Happy Loops	A.10: Ocean Scene with Loops	A.12: On the Move with Events	*There are currently no open ended projects written into courses A-B, but if you're interested in developing an open ended project you can start with the materials in lesson C.18*
Course B *First Grade*	B.10 - The Right App	B.9: Drawing Gardens with Loops	B.12: A Royal Battle with Events	
Course C *Second Grade*	C.8: Binary Bracelets	C.13: Sticker Art with Loops	C.16: Chase Game with Events	C.18 - End of Course Project
Course D *Third Grade*	D.16: Binary Images	D.17: Binary Images with Artist	D.6 - Build a Star Wars Game	D.19 - End of Course Project
Course E *Fourth Grade*	E.14 - Songwriting	E.17: Functions with Artist	E.13 - Nested Loops in Frozen	E.19 - End of Course Project
Course F *Fifth Grade*	F.7 - Envelope Variables	F.8 - Changing Variables with Artist	F.18 - Virtual Pet	F.19 - End of Course Project

Step 4	Access your lesson plan materials and read the lesson • All lesson plans are available at curriculum.code.org/csf • Access all digital materials via the code.org website: code.org > course catalog> "learn more" on elementary school tile > scroll down to course tiles.
Step 5	Read your lesson and prepare to make an implementation plan. Once you have read the lesson plan, you are ready to use the Code.org Lesson Implementation Planning Guide as a template to help you think about how you will teach this lesson.

CS Fundamentals Lesson Implementation Plan Template

Course ____________________ Lesson __________

Make a Plan

Goal
You have thought about and made a detailed plan for how you might teach the lesson in a way that leverages the CS Fundamental instructional practices and supports students

What's happening	Where can this happen in the lesson?
Instructional approaches to use in this lesson What instructional techniques or approaches do you want to be sure to use in this lesson and where do you want to use them?	
Role as the teacher When/where in the lesson do you want the whole class to discuss a topic? What role do you plan to take and when?	

What's happening	Where can this happen in the lesson?
Classroom environment and student interaction When do you want students to interact during this lesson? What should that look like?	
Assessing student learning in the lesson How can you assess if students have learned what they need to learn during this lesson?	
Supporting debugging Where do you think students might need to engage in debugging during this lesson (which levels/activities?) How can you support them with that debugging?	
Making connections Where is there space in this lesson to make connections to the real world/other things students are learning in school?	

Session: Implementation Unconference

Reflect and write

How much of CS Fundamentals do you want to teach to your students?

Now that we've spent more time digging into CS Fundamentals, what is top of mind as a barrier for making CS Fundamentals work in your classroom?

Think both in terms of getting as far as you might like and in terms of making those lessons as effective as possible.

What goals do you have for your CS Fundamentals Class?

Appendix C: Notes

Use the following pages to jot down questions, ideas, and reflections as you engage with curriculum at workshops and while teaching.