Table of Contents

Welcome to Computer Science Discoveries **1**

Code.org Values and Philosophy **3**
- Curriculum Values 3
- Pedagogical Approach To Our Values 4

CS Discoveries Curriculum Overview **5**
- Unit 1 - Problem Solving and Computing 5
- Unit 2 - Web Development 8
- Unit 3 - Interactive Animations and Games 12
- Unit 4 - The Design Process 16
- Unit 5 - Data and Society 19
- Unit 6 - Physical Computing 22

Learning Tools **25**

Unplugged and Plugged Activities **27**

Teaching and Learning Strategies **28**
- Student Practices 33
- Problem Solving Process 33
- The Problem Solving Process for Programming 34
- The Problem Solving Process for Design 34
- The Problem Solving Process for Data 34

Course Resources **35**
- Code.org Website 35
- Course Overview Pages 36
- Unit Overview Pages 37
- Lesson Structure and Iconography 40

Assessment and Feedback **43**

Planning for the Year **45**
- Pacing 45
- Tech Requirements and Required Materials 45

Appendix A: Professional Learning Handouts **46**
- Build your Action Plan: Getting to Fall 47
- Pacing and Planning: Instructional Units 48

Appendix B: CS Discoveries Getting Started and Implementation Options **49**
- Getting Started 50
- Intended Implementation Options 52

Appendix C: Additional Guides and Resources **53**
- Guide to Debugging - Student Facing 54
- Guide to Debugging - Teacher Facing 55
- Guide to Differentiation - Teacher Facing 58
- Guide to Resources - Student Facing 60
- Guide to Resources - Teacher Facing 61

Welcome to Computer Science Discoveries

CS Discoveries is an introductory computer science course that empowers students to create authentic artifacts and engage with computer science as a medium for creativity, communication, problem solving, and fun.

Semester 1 - Exploration and Expression

The first semester of CS Discoveries introduces students to computer science as a vehicle for problem solving, communication, and personal expression. As a whole, this semester focuses on the visible aspects of computing and computer science, and encourages students to see where computer science exists around them and how they can engage with it as a tool for exploration and expression.

Unit 1: Problem Solving and Computing	Students learn the problem-solving process, the input-output-store-process model of a computer, and how computers help humans solve problems. Students end the unit by proposing their own app to solve a problem.
Unit 2: Web Development	Students learn to create websites using HTML and CSS inside Code.org's Web Lab environment. Throughout the unit, students consider questions of privacy and ownership on the internet as they develop their own personal websites.
Unit 3: Interactive Animations and Games	Students learn fundamental programming constructs and practices in the JavaScript programming language while developing animations and games in Code.org's Game Lab environment. Students end the unit by designing their own animations and games.

Semester 2 - Innovation and Impact

Where the first semester centers on the immediately observable and personally applicable elements of computer science, the second semester asks students to look outward and explore the impact of computer science on society. Students will see how a thorough user-centered design process produces a better application, how data is used to address problems that affect large numbers of people, and how physical computing with circuit boards allows computers to collect input and return output in a variety of ways.

Unit 4: The Design Process	Students apply the problem solving process to the problems of others, learning to empathize with the needs of a user and design solutions to address those needs. During the second half of the unit, students form teams to prototype an app of their own design, first on paper and eventually in Code.org's App Lab environment.
Unit 5: Data and Society	Students explore different systems used to represent information in a computer and the challenges and tradeoffs posed by using them. In the second half of the unit, students learn how collections of data are used to solve problems and how computers help to automate the steps of this process.
Unit 6: Physical Computing	Students use Code.org's App Lab environment, in conjunction with the Adafruit Circuit Playground, to explore the relationship between hardware and software. Throughout the unit, students develop prototypes that mirror existing innovative computing platforms, before ultimately designing and prototyping one of their own.

Standards and Learning Framework

CS Discoveries was written using both the K-12 Framework for Computer Science and the CSTA standards as guidance. Lists of connected standards can be found within each lesson where they are addressed. An overview of all of the standards for the course can be viewed at curriculum.code.org/csd/standards/

In addition to the CSTA standards and K-12 Framework, CS Discoveries was written with a learning framework, which is written for each unit and outlines the expected student outcomes that are assessed throughout the unit and in that unit's major projects and post-project test. These outcomes are organized into concept clusters to help students and teachers understand the broad goals of each unit.

Tools Across the Course

CS Discoveries introduces students to tools and programming languages that are accessible for beginners while offering more advanced students opportunities to create sophisticated projects. All of the tools below are integrated directly into the Code.org website, allowing teachers to have visibility into all student work and progress.

Unit 1 ***Problem Solving and Computing***	*No special learning tools*
Unit 2 ***Web Development***	**Web Lab** — A browser-based tool for creating and publishing HTML and CSS web sites.
Unit 3 ***Interactive Animations and Games***	**Game Lab** — A browser-based JavaScript programming environment designed to create sprite-based drawings, animations and games. Enables students to switch between programming in blocks or text.
Unit 4 ***The Design Process***	**App Lab** — A browser-based JavaScript programming environment for creating interactive apps. Enables students to switch between programming in blocks or text.
Unit 5 ***Data and Society***	*No special learning tools*
Unit 6 ***Physical Computing***	**Circuit Playground** — Adafruit's low-cost microcontroller featuring multiple integrated sensors and output devices. **Maker Toolkit** — A collection of commands that extends App Lab's capabilities to allow students to easily program the Circuit Playground and directly from App Lab.

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 so 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 lead to attrition of 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 residue of this is that we never 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 - 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, all 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

Our lessons almost always call for students to interact with other students in the class in some way. 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 classroom 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; and we strive to include projects for teams in which everyone must play a critical role.

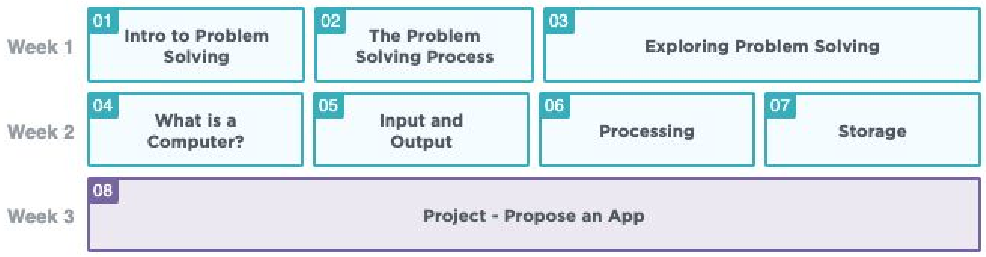
CS Discoveries Curriculum Overview

The following pages provide an overview of each of the 6 units in the CS Discoveries curriculum. For each unit, there is a one page description of the unit, timeline, big questions answered throughout the unit, learning goals, major projects, and tools, and the 1 - 3 pages that follow the overview outline each lesson of the unit.

Unit 1 - Problem Solving and Computing

Overview and Timeline

The Problem Solving and Computing unit is a highly interactive and collaborative introduction to the field of computer science, as framed within the broader pursuit of solving problems. Through a series of puzzles, challenges, and real world scenarios, students are introduced to a problem solving process that they will return to repeatedly throughout the course. Students then learn how computers input, output, store, and process information to help humans solve problems within the context of apps. The unit concludes with students designing an app that helps solve a problem of their choosing.



Week 1	01 Intro to Problem Solving	02 The Problem Solving Process	03 Exploring Problem Solving	
Week 2	04 What is a Computer?	05 Input and Output	06 Processing	07 Storage
Week 3	08 Project - Propose an App			

Big Questions

Chapter 1 - The Problem Solving Process

- What strategies and processes can I use to become a more effective problem solver?

Chapter 2 - Computers and Problem Solving

- How do computers help people to solve problems?
- How do people and computers approach problems differently?
- What does a computer need from people in order to solve problems effectively?

Unit Goals

By the end of the unit, students should be able to identify the defined characteristics of a computer and how it is used to solve information problems. They should be able to use a structured problem solving process to address problems and design solutions that use computing technology. The unit also serves to build a collaborative classroom environment where students view computer science as relevant, fun, and empowering.

Alternate Lessons

Alternate versions of Lessons 1 and 3 are available, and they target the same learning goals. Teachers may choose which lesson to implement based on the availability of supplies or student interests. Chapter 1 of Unit 1 can be used as a review for students who have already taken a portion of the Computer Science Discoveries course in a different grade level or semester. In this case, the teacher can substitute in a different lesson from the one that students have already completed.

Major Projects

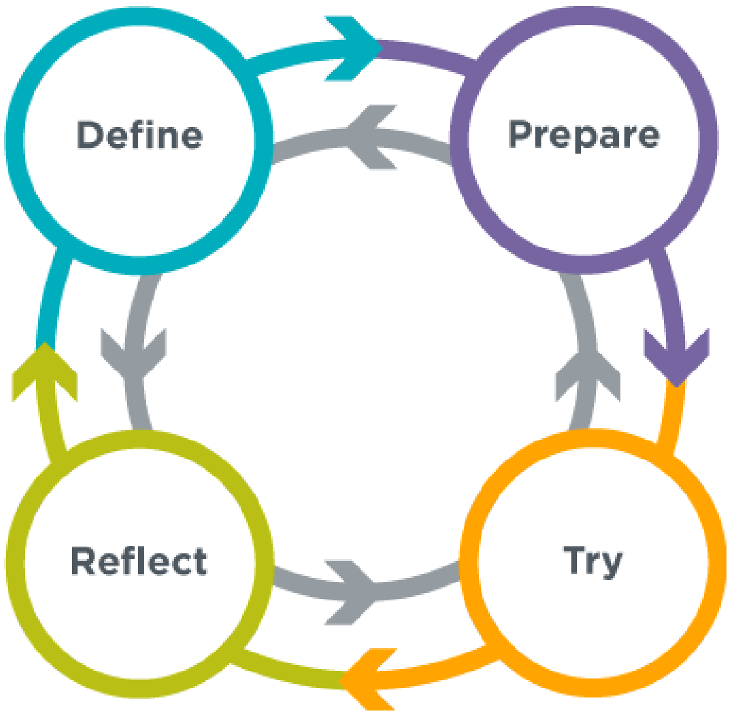
- **Lesson 8: Project - Propose an App**

To conclude their study of the problem solving process and the input/output/store/process model of a computer, students will propose an app designed to solve a problem of their choosing. To learn more about this project, check out the description in this unit's lesson progression.

Lesson Progression: Unit 1 - Problem Solving and Computing

Chapter 1 The Problem Solving Process	**Lesson 1 Alternate Lessons (Select One)**	**Lesson 1: Intro to Problem Solving - Aluminum Boats** The class works in groups to design aluminum foil boats that will support as many pennies as possible. At the end of the lesson, groups reflect on their experiences with the activity and make connections to the types of problem solving they will be doing for the rest of the course.
		Lesson 1: Exploring Problem Solving - Newspaper Table The class works in groups to design newspaper tables that will support as many books as possible. At the end of the lesson, groups reflect on their experiences with the activity and make connections to the types of problem solving they will be doing for the rest of the course.
		Lesson 1: Exploring Problem Solving - Spaghetti Bridge The class works in groups to design spaghetti bridges that will support as many books as possible. At the end of the lesson, groups reflect on their experiences with the activity and make connections to the types of problem solving they will be doing for the rest of the course.
	Lesson 2: The Problem Solving Process This lesson introduces the formal problem solving process that the class will use over the course of the year: Define - Prepare - Try - Reflect. The class relates these steps to the problem from the previous lesson, then to a problem they are good at solving, then to a problem they want to improve at solving. At the end of the lesson, the class collects a list of generally useful strategies for each step of the process to put on posters that will be used throughout the unit and year.	
	Lesson 3 Alternate Lessons (Select One)	**Lesson 3: Exploring Problem Solving** In this lesson, the class applies the problem solving process to three different problems: a word search, a seating arrangement for a birthday party, and planning a trip. The problems grow increasingly complex and poorly defined to highlight how the problem solving process is particularly helpful when tackling these types of problems.
		Lesson 3: Exploring Problem Solving - Animal Theme In this lesson, the class applies the problem solving process to three different problems: a tangram puzzle, choosing a pet according to criteria, and planning a pet adoption event. The problems grow increasingly complex and poorly defined to highlight how the problem solving process is particularly helpful when tackling these types of problems.
		Lesson 3: Exploring Problem Solving - Games Theme In this lesson, the class applies the problem solving process to three different problems: a maze, a logic puzzle, and planning field day activity. The problems grow increasingly complex and poorly defined to highlight how the problem solving process is particularly helpful when tackling these types of problems.
Chapter 2 Computers and Problem Solving	**Lesson 4: What is a Computer?** In this lesson, the class develops a preliminary definition of a computer. After brainstorming the possible definitions for a computer, the class works in groups to sort pictures into "is a computer" or "is not a computer" categories on poster paper, and explain their motivations for choosing some of the most difficult categorizations. The teacher then introduces a definition of the computer and allows groups to revise their posters according to the new definition.	

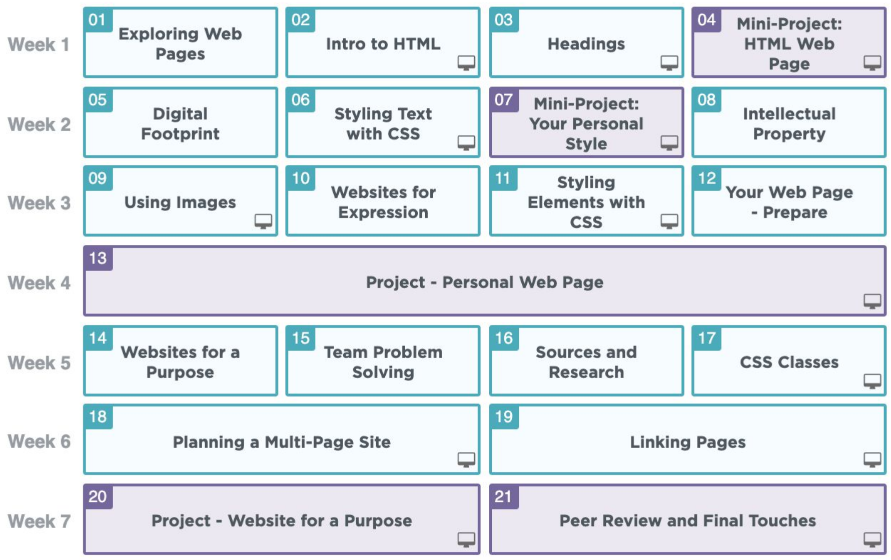
Chapter 2 Computers and Problem Solving (continued)	**Lesson 5: Input and Output** In this lesson, the class considers how computers get and give information to the user through inputs and outputs. After first considering what information they would need to solve a "thinking problem", the class identifies the inputs and outputs to that process. Afterwards, they explore a series of apps and determine the inputs and outputs for each one.
	Lesson 6: Processing This lesson introduces four types of processing that the class will use throughout the course. Through a series of apps, the class explores how processing is used to turn input into output. In the end, the class brainstorms more types of app processing that would be useful.
	Lesson 7: Storage This lesson covers the last part of the chapter's model of computing: storage. The class interacts with several different apps, determining which information should be stored for later and why. The input-output-storage-processing model of computing is then presented in full, and the class reflects on how various apps use each of the components, and how the model impacts whether or not something should be considered a computer.
	Lesson 8: Project - Propose an App To conclude the study of the problem solving process and the input/output/store/process model of a computer, the class proposes apps designed to solve real world problems. This project is completed across multiple days and culminates in a poster presentation where students highlight the features of their apps. The project is designed to be completed in pairs, though it can be completed individually.



Unit 2 - Web Development

Overview and Timeline

In Web Development, students are empowered to create and share content on their own web pages. They begin by thinking about the role of the web and how it can be used as a medium for creative expression. As students develop their pages and begin to see themselves as programmers, they are encouraged to think critically about the impact of sharing information online and how to be more critical consumers of content. They are also introduced to problem solving as it relates to programming while they learn valuable skills such as debugging, using resources, and teamwork. At the conclusion of the unit, students will have created a personal website they can publish and share.



Week 1	01 Exploring Web Pages	02 Intro to HTML	03 Headings	04 Mini-Project: HTML Web Page
Week 2	05 Digital Footprint	06 Styling Text with CSS	07 Mini-Project: Your Personal Style	08 Intellectual Property
Week 3	09 Using Images	10 Websites for Expression	11 Styling Elements with CSS	12 Your Web Page - Prepare
Week 4	13 Project - Personal Web Page			
Week 5	14 Websites for a Purpose	15 Team Problem Solving	16 Sources and Research	17 CSS Classes
Week 6	18 Planning a Multi-Page Site		19 Linking Pages	
Week 7	20 Project - Website for a Purpose		21 Peer Review and Final Touches	

Big Questions

Chapter 1 - Creating Web Pages

- Why do people create websites?
- How can text communicate content and structure on a web page?
- How do I safely and appropriately make use of the content published on the internet?
- What strategies can I use when coding to find and fix issues?

Chapter 2 - Multi-Page Websites

- How can websites be used to address problems in the world?
- What strategies can teams use to work better together?
- How do I know what information can be trusted online?

Unit Goals

By the end of the unit, students should be able to create a digital artifact that uses multiple computer languages to control the structure and style of their content, and view computer science as a tool for personal expression. They should understand that different programming languages allow them to solve different problems, and that these solutions can be generalized across similar problems. Lastly, they should understand their role and responsibilities as both creators and consumers of digital media.

Major Projects

- **Lesson 13: Project - Personal Web Page**
- **Lesson 20: Project - Website for a Purpose**

Throughout the unit, students use their developing skills to create a multi-page website, and have several opportunities to share out and engage in peer review at the end of each chapter. These projects emphasize many of the core practices of this course as students will need to tap into their creativity, problem solving skills, and persistence to complete their websites. To learn more about these projects, check out the descriptions in this unit's lesson progression.

Tools

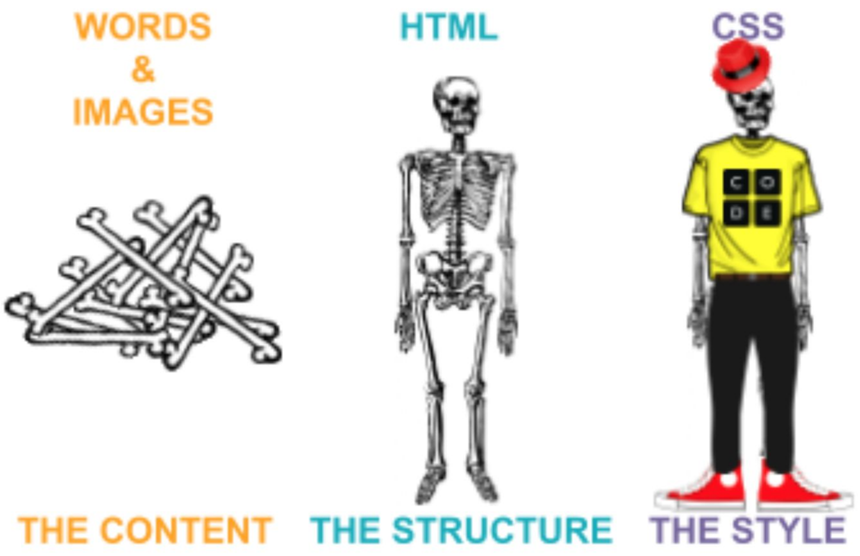
This unit uses **Web Lab**. For more detailed information, see the **Learning Tools** section of this curriculum guide

Lesson Progression: Unit 2 - Web Development

Chapter 1 Creating Web Pages	**Lesson 1: Exploring Websites** This lesson covers the purposes that a web page might serve, both for the users and the creators. The class explores a handful of sample web pages and describes how each of those pages is useful for users and how they might also serve their creators.
	Lesson 2: Intro to HTML This lesson introduces HTML as a solution to the problem of how to communicate both the content and structure of a website to a computer. The lesson begins with a brief unplugged activity that demonstrates the challenges of effectively communicating the structure of a web page. Then, the class looks at an HTML page in Web Lab and discusses how HTML tags help solve this problem, before using HTML to write their first web pages of the unit.
	Lesson 3: Headings This lesson continues the introduction to HTML tags, this time with headings. The class practices using heading tags to create page and section titles and learns how the different heading elements are displayed by default.
	Lesson 4: Mini-Project: HTML Web Page In this lesson, the class creates personal web pages on a topic of their choice. The lesson starts with a review of HTML tags. Next, the class designs web pages, first identifying the tags needed to implement them, and then creating the pages in Web Lab.
	Lesson 5: Digital Footprint This lesson takes a step back from creating the personal website to talk about the personal information that people choose to share digitally. The class begins by discussing what types of information they have shared on various websites, then they look at several sample social media pages to see what types of personal information could be shared intentionally or unintentionally. Finally, the class comes up with a set of guidelines to follow when putting information online.
	Lesson 6: Styling Text with CSS This lesson introduces CSS as a way to style elements on the page. The class learns the basic syntax for CSS rule-sets and then explores properties that impact HTML text elements. Finally, they discuss the differences between content, structure, and style when making a personal web page.
	Lesson 7: IMini-Project: Your Personal Style In this lesson, students create their own styled web pages. The lesson starts with a review of the CSS. They then design the web page, identify which CSS properties they will need, and create their web pages in Web Lab.
	Lesson 8: Intellectual Property Starting with a discussion of their personal opinions on how others should be allowed to use their work, the class explores the purpose and role of copyright for both creators and users of creative content. They then move on to an activity exploring the various Creative Commons licenses as a solution to the difficulties of dealing with copyright.
	Lesson 9: Using Images The class starts by considering the ethical implications of using images on websites, specifically in terms of intellectual property. They then learn how to add images to their web pages using the <img> tag and how to cite the image sources appropriately.

Chapter 1 Creating Web Pages (continued)	**Lesson 10: Websites for Expression** This lesson introduces websites as a means of personal expression. Students first discuss the different ways that people express and share their interests and ideas, then they look at a few exemplar websites made by students from a previous course. Finally, everyone brainstorms and shares a list of topics and interests to include in a personal website, creating a resource for developing a personal website in the rest of the unit.
	Lesson 11: Styling Elements with CSS This lesson continues the introduction to CSS style properties, this time focusing more on non-text elements. The class begins by investigating and modifying the new CSS styles on a Desserts of the World page. Afterwards, everyone applies this new knowledge to their personal websites.
	Lesson 12: Your Web Page - Prepare In this lesson, students engage in the "prepare" stage of the problem solving process by deciding what elements and style their web pages will have. They review the different HTML, CSS, and digital citizenship guidelines, then design and plan their pages, as well as download and document the images they will need. Afterwards, they reflect on how their plan will ensure that the website does what it is designed to do.
	Lesson 13: Project - Personal Web Page After quickly reviewing the debugging process, the class goes online to create the pages that they have planned out in previous lessons, with the project guides as a reference. Afterwards, they engage in a structured reflection and feedback process before making any final updates.
Chapter 2 Multi-page Websites	**Lesson 14: Websites for a Purpose** In this lesson, students explore the different reasons people make websites. After brainstorming various reasons that they visit websites, they investigate sample web sites that have been created to address a particular problem and decide what different purposes those websites might serve for the creators. The class then thinks of problems they might want to solve with their own websites.
	Lesson 15: Team Problem Solving Teams work together to set group norms and brainstorm what features they would like their websites to have. The class starts by reflecting on what makes teams successful. Teams then make plans for how they will interact and achieve success in their own projects before brainstorming ideas for their website projects.
	Lesson 16: Sources and Research This lesson covers how to find relevant and trustworthy information online. After viewing and discussing a video about how search engines work, students search for information relevant to their sites, then analyze the sites for credibility to decide which are appropriate to use on their own website.
	Lesson 17: CSS Classes This lesson introduces CSS classes, which allow web developers to treat groups of elements they want styled differently than other elements of the same type. Students first investigate and modify classes on various pages, then create their own classes and use them to better control the appearance of their pages. Teams then reflect on how they could use this skill to improve their websites.
	Lesson 18: Planning a Multi-Page Site The class works in teams to plan out the final web sites, including a sketch of each page. They then download the media that they will need for their sites. At the end of the activity, they decide how the work will be distributed among them and report whether the entire team agreed to the plan.

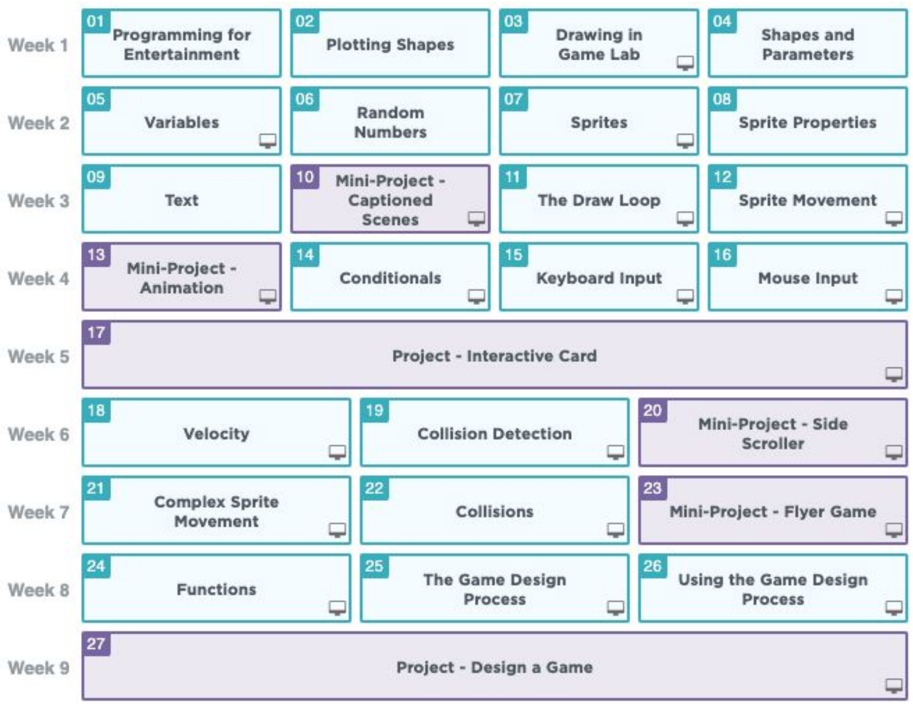
Chapter 2 Multi-page Websites (continued)	**Lesson 19: Linking Pages** The class begins this lesson by looking online for the internet's first web page and discussing how its use of links was what started the web. They then transition to Web Lab where they learn how to make their own links, as well as good conventions that make it easier for users to navigate on a page. Last, they reflect on their team project and what their personal goals are for the final stretch.
	Lesson 20: Project - Website for a Purpose In this lesson, teams are finally able to code the pages that they have been planning. Using the project guide, the team works together and individually to code all of their pages, then puts all of the work together into a single site.
	Lesson 21: Peer Review and Final Touches This lesson focuses on the value of peer feedback. The class first reflects on what they are proud of, and what they would like feedback on. They then give and get that feedback through a structured process that includes the project rubric criteria. Afterwards, everyone puts the finishing touches on their sites and reflects on the process before a final showcase.



Unit 3 - Interactive Animations and Games

Overview and Timeline

In the Interactive Animations and Games unit, students build on their coding experience as they create programmatic images, animations, interactive art, and games. Starting off with simple, primitive shapes and building up to more sophisticated sprite-based games, students become familiar with the programming concepts and the design process computer scientists use daily. They then learn how these simpler constructs can be combined to create more complex programs. In the final project, students develop a personalized, interactive program. Along the way, they practice design, testing, and iteration, as they come to see that failure and debugging are an expected and valuable part of the programming process.



Week 1	01 Programming for Entertainment	02 Plotting Shapes	03 Drawing in Game Lab	04 Shapes and Parameters
Week 2	05 Variables	06 Random Numbers	07 Sprites	08 Sprite Properties
Week 3	09 Text	10 Mini-Project - Captioned Scenes	11 The Draw Loop	12 Sprite Movement
Week 4	13 Mini-Project - Animation	14 Conditionals	15 Keyboard Input	16 Mouse Input
Week 5	17 Project - Interactive Card			
Week 6	18 Velocity	19 Collision Detection	20 Mini-Project - Side Scroller	
Week 7	21 Complex Sprite Movement	22 Collisions	23 Mini-Project - Flyer Game	
Week 8	24 Functions	25 The Game Design Process	26 Using the Game Design Process	
Week 9	27 Project - Design a Game			

Big Questions

Chapter 1 - Images and Animations

- What is a computer program?
- What are the core features of most programming languages?
- How does programming enable creativity and individual expression?
- What practices and strategies will help me as I write programs?

Chapter 2 - Building Games

- How do software developers manage complexity and scale?
- How can programs be organized so that common problems only need to be solved once?
- How can I build on previous solutions to create even more complex behavior?

Unit Goals

By the end of the unit, students should be able to create an interactive animation or game that includes basic programming concepts such as control structures, variables, user input, and randomness. They should manage this task by working with others to break it down using objects (sprites) and functions. Throughout the process, they should give and respond constructively to peer feedback, and work with their teammates to complete a project. Students should leave this unit viewing themselves as computer programmers, and see programming as a fun and creative form of expression.

Major Projects

- **Lesson 17: Project - Interactive Card**
- **Lesson 27: Project - Design a Game**

There are two major projects in this unit, which are at the end of each chapter. Both offer students an opportunity to demonstrate what they've learned while leveraging creativity and peer feedback. To learn more about these projects, check out the descriptions in this unit's lesson progression.

Tools

This unit uses **Game Lab**. For more detailed information, see the **Learning Tools** section of this curriculum guide.

Lesson Progression: Unit 3 - Interactive Animations and Games

Chapter 1 Images and Animations	**Lesson 1: Programming for Entertainment** The class is asked to consider the "problems" of boredom and self expression, and to reflect on how they approach those problems in their own lives. From there, they will explore how Computer Science in general, and programming specifically, plays a role in either a specific form of entertainment or as a vehicle for self expression.
	Lesson 2: Plotting Shapes This lesson explores the challenges of communicating how to draw with shapes and use a tool that introduces how this problem is approached in Game Lab.The class uses a Game Lab tool to interactively place shapes on Game Lab's 400 by 400 grid. Partners then take turns instructing each other how to draw a hidden image using this tool, which accounts for many of the challenges of programming in Game Lab.
	Lesson 3: Drawing in Game Lab The class is introduced to Game Lab, the programming environment for this unit, and begins to use it to position shapes on the screen. The lesson covers the basics of sequencing and debugging, as well as a few simple commands. At the end of the lesson, students will be able to program images like the ones they made with the drawing tool in the previous lesson.
	Lesson 4: Shapes and Parameters In this lesson, students continue to develop a familiarity with Game Lab by manipulating the width and height of the shapes they use to draw. The lesson kicks off with a discussion that connects expanded block functionality (e.g. different sized shapes) with the need for more block inputs, or "parameters." Finally, the class learns to draw with versions of ellipse() and rect() that include width and height parameters and to use the background() block.
	Lesson 5: Variables This lesson introduces variables as a way to label a number in a program or save a randomly generated value. The class begins the lesson with a very basic description of the purpose of a variable and practices using the new blocks, then completes a level progression that reinforces the model of a variable as a way to label or name a number.
	Lesson 6: Random Numbers Students are introduced to the randomNumber() block and how it can be used to create new behaviors in their programs. They then learn how to update variables during a program and use those skills to draw randomized images.
	Lesson 7: Sprites In order to create more interesting and detailed images, the class is introduced to the sprite object. The lesson starts with a discussion of the various information that programs must keep track of, then presents sprites as a way to keep track of that information. Students then learn how to assign each sprite an image, which greatly increases the complexity of what can be drawn on the screen.
	Lesson 8: Sprite Properties Students extend their understanding of sprites by interacting with sprite properties. The lesson starts with a review of what a sprite is, then moves on to Game Lab for more practice with sprites, using their properties to change their appearance. The class then reflects on the connections between properties and variables.
	Lesson 9: Text This lesson introduces Game Lab's text commands, giving students more practice using the coordinate plane and parameters. At the beginning of the lesson, they are asked to caption a cartoon created in Game Lab. They then move onto Code Studio where they practice placing text on the screen and controlling other text properties, such as size.

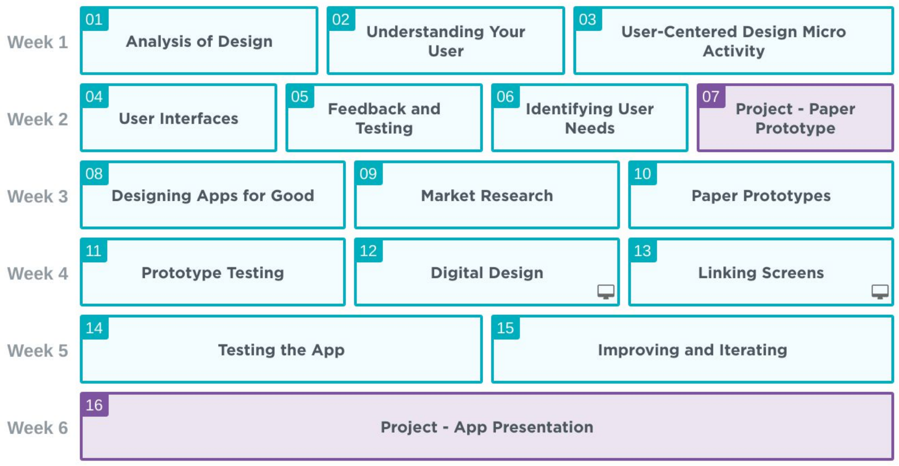
Chapter 1 Images and Animations (continued)	**Lesson 10: Mini-Project - Captioned Scenes** After a quick review of the code learned so far, the class is introduced to the first creative project of the unit. Using the problem solving process as a model, students define the scene that they want to create, prepare by thinking of the different code they will need, try their plan in Game Lab, then reflect on what they have created. In the end, they also have a chance to share their creations with their peers.
	Lesson 11: The Draw Loop This lesson introduces the draw loop, one of the core programming paradigms in Game Lab. Students learn how to combine the draw loop with random numbers to manipulate some simple animations first with dots and then with sprites.
	Lesson 12: Sprite Movement In this lesson, the class learns how to control sprite movement using a construct called the counter pattern, which incrementally changes a sprite's properties. After brainstorming different ways that they could animate sprites by controlling their properties, students explore the counter pattern in Code Studio, using the counter pattern to create various types of sprite movements.
	Lesson 13: Mini-Project - Animation In this lesson, the class is asked to combine different methods from previous lessons to create an animated scene. Students first review the types of movement and animation that they have learned, and brainstorm what types of scenes might need that movement. They then begin to plan out their own animated scenes, which they create in Game Lab.
	Lesson 14: Conditionals This lesson introduces students to booleans and conditionals, which allow a program to run differently depending on whether a condition is true. The class starts by playing a short game in which they respond according to whether particular conditions are met. They then move to Code Studio where they learn how the computer evaluates boolean expressions, and how they can be used to structure a program.
	Lesson 15: Keyboard Input Following the introduction to booleans and if statements in the previous lesson, students are introduced to a new block called keyDown(), which returns a boolean and can be used in conditionals statements to move sprites around the screen. By the end of this lesson they will have written programs that take keyboard input from the user to control sprites on the screen.
	Lesson 16: Mouse Input The class continues to explore ways to use conditional statements to take user input. In addition to the keyboard commands learned yesterday, they learn about several ways to take mouse input. They also expand their understanding of conditional to include `else`, which allows for the computer to run a certain section of code when a condition is true, and a different section of code when it is not.
	Lesson 17: Project - Interactive Card In this cumulative project for Chapter 1, students plan for and develop an interactive greeting card using all of the programming techniques they've learned to this point.
Chapter 2 Building Games	**Lesson 18: Velocity** After a brief review of how the counter pattern is used to move sprites, the class is introduced to the idea of hiding those patterns in a single block, in order to help manage the complexity of programs. They then head to Code Studio to try out new blocks that set a sprite's velocity directly, and look at the various ways that they are able to code more complex behaviors in their sprites.
	Lesson 19: Collision Detection In this lesson, the class learns about collision detection on the computer. Working in pairs, they explore how a computer could use math, along with the sprite location and size properties,h to detect whether two sprites are touching. They then use the isTouching() block to create different effects when sprites collide and practice using the block to model various interactions.

Chapter 2 Building Games (continued)	**Lesson 20: Mini-Project - Side Scroller** Students use what they have learned about collision detection and setting velocity to create simple side scroller games. After looking at a sample side scroller game, they brainstorm what sort of side scroller they would like, then use a structured process to program the game in Code Studio.
	Lesson 21: Complex Sprite Movement The class learns to combine the velocity properties of sprites with the counter pattern to create more complex sprite movement. After reviewing the two concepts, they explore various scenarios in which velocity is used in the counter pattern, and observe the different types of movement that result. In particular, students learn how to simulate gravity. They then reflect on how they were able to get new behaviors by combining blocks and patterns that they already knew.
	Lesson 22: Collisions In this lesson, the class programs their sprites to interact in new ways. After a brief review of how they used the isTouching block, students brainstorm other ways that two sprites could interact. They then use isTouching to make one sprite push another across the screen before practicing with the four collision blocks (collide, displace, bounce, and bounceOff).
	Lesson 23: Mini-Project - Flyer Game Students use what they have learned about simulating gravity and the different types of collisions to create simple flyer games. After looking at a sample flyer game, they brainstorm what sort of flyer they would like, then use a structured process to program the game in Code Studio.
	Lesson 24: Functions This lesson covers functions as a way for students to organize their code, make it more readable, and remove repeated blocks of code. The class learns that higher level or more abstract steps make it easier to understand and reason about steps, then begins to create functions in Game Lab.
	Lesson 25: The Game Design Process This lesson introduces the process that students will use to design games for the remainder of the unit. This process is centered around a project guide that asks students to define their sprites, variables, and functions before they begin programming their game. They walk through this process in a series of levels. At the end of the lesson, students have an opportunity to make improvements to the game to make it their own.
	Lesson 26: Using the Game Design Process In this multi-day lesson, the class uses the problem solving process from Unit 1 to create a platform jumper game. After looking at a sample game, they define what their games will look like and use a structured process to build them. Finally, the class reflects on how the games could be improved, and implements those changes.
	Lesson 27: Project - Design a Game Students plan and build original games using the project guide from the previous two lessons. Working individually or in pairs, they plan, develop, and give feedback on the games. After incorporating the peer feedback, students share out their completed games.

Unit 4 - The Design Process

Overview and Timeline

The Design Process unit transitions students from thinking about computer science as a tool to solve their own problems towards considering the broader social impacts of computing. Through a series of design challenges, students are asked to consider and understand the needs of others while developing a solution to a problem. The second half of the unit consists of an iterative team project, during which students have the opportunity to identify a need that they care about, prototype solutions both on paper and in App Lab, and test their solutions with real users to get feedback and drive further iteration.



Week 1	01 Analysis of Design	02 Understanding Your User	03 User-Centered Design Micro Activity	
Week 2	04 User Interfaces	05 Feedback and Testing	06 Identifying User Needs	07 Project - Paper Prototype
Week 3	08 Designing Apps for Good	09 Market Research	10 Paper Prototypes	
Week 4	11 Prototype Testing	12 Digital Design	13 Linking Screens	
Week 5	14 Testing the App	15 Improving and Iterating		
Week 6	16 Project - App Presentation			

Big Questions

Chapter 1: User Centered Design

- How do computer scientists identify the needs of their users?
- How can we ensure that a user's needs are met by our designs?
- What processes will best allow us to efficiently create, test, and iterate upon our design?

Chapter 2: App Prototyping

- How do teams effectively work together to develop software?
- What roles beyond programming are necessary to design and develop software?
- How do designers incorporate feedback into multiple iterations of a product?

Unit Goals

By the end of the unit, students should see the design process as a form of problem solving that prioritizes the needs of a user. They should be able to identify user needs and assess how well different designs address them. In particular, they know how to develop paper and digital prototypes, gather and respond to feedback about a prototype, and consider ways different user interfaces do or do not affect the usability of their apps. Students should leave the unit with a basic understanding of other roles in software development, such as product management, marketing, design, and testing, and how to use what they have learned about computer science as a tool for social impact.

Major Projects

- **Lesson 7: Project - Paper Prototype**
- **Lesson 16: Project - App Presentation**

Students are encouraged to focus on the design process in the two major projects in this unit, which are at the end of each chapter. To learn more about these projects, check out the descriptions in this unit's lesson progression.

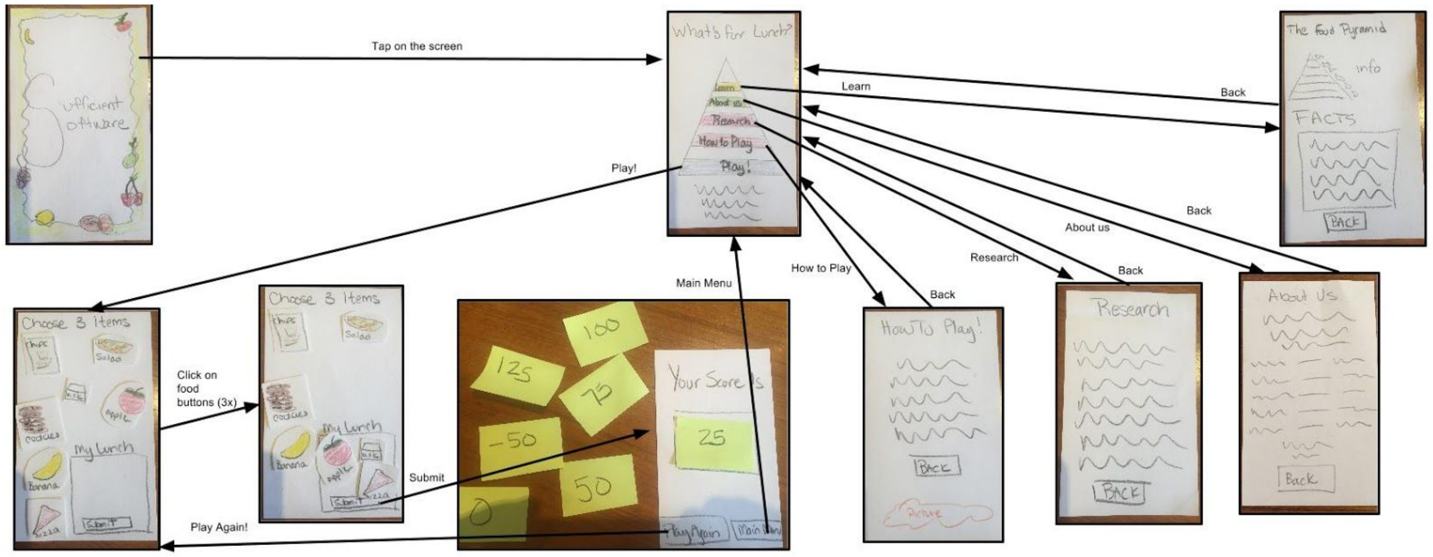
Tools

This unit uses the **App Lab**. For more detailed information, see the **Learning Tools** section in this curriculum guide.

Lesson Progression: Unit 4 - The Design Process

Chapter 1 User Centered Design	**Lesson 1: Analysis of Design** The class explores a variety of different teapot designs to consider design choices. Building on this, students explore the relationship between users, their needs, and the design of objects they use.
	Lesson 2: Understanding Your User Using user profiles, students explore how different users might react to a variety of products. Role playing as a different person, each member of the class will get to experience designs through someone else's eyes.
	Lesson 3: User-Centered Design Micro Activity In small groups, students use the design process to come up with ideas for smart clothing. From brainstorming, to identifying users, to finally proposing a design, this activity serves as the first of several opportunities in this unit for students to practice designing a solution for the needs of others.
	Lesson 4: User Interfaces In this lesson, students get to see how a paper prototype can be used to test and get feedback on software before writing any code. To help out a developer with their idea, the class tests and provides an app prototype made of paper.
	Lesson 5: Feedback and Testing Users have been testing an app, and they have lots of feedback for the developer. The class needs to sort through all of this feedback, identify the common themes and needs, and start revising the prototype to make it better meet the users' needs.
	Lesson 6: Identifying User Needs Up to this point, the users that the class has considered have all been remote, and the only information from users has come through text or role playing. Now students get to rely on each other as potential users, as pairs interview each other to identify needs that could be addressed by developing an app.
	Lesson 7: Project - Paper Prototype Using the interview information from the previous lesson, students come up with app ideas to address the needs of their users. To express those ideas, and test out their effectiveness, each student creates and tests paper prototypes of their own.
Chapter 2 App Prototyping	**Lesson 8: Designing Apps for Good** To kick off the app design project, the class organizes into teams and starts exploring app topics. Several examples of socially impactful apps serve as inspiration for the project.
	Lesson 9: Market Research In this lesson, students dive into app development by exploring existing apps that may serve similar users. In groups, they identify a handful of apps that address the same topic they are working on, and use those apps to help refine the app idea they will pursue.
	Lesson 10: Paper Prototypes Paper prototypes allow developers to quickly test ideas before investing a lot of time writing code. In this lesson, teams explore some example apps created in App Lab and use these examples to help inform the first paper prototypes of their apps.
	Lesson 11: Prototype Testing In this lesson, teams test out their paper prototypes with other members of the class. As one student role plays as the computer, one narrates, and the rest observe, teams will get immediate feedback on their app designs, which will inform the next version of their app prototypes.

Chapter 2 App Prototyping *(continued)*	**Lesson 12: Digital Design** Having developed, tested, and gathered feedback on a paper prototype, teams now move to App Lab to build the next iteration of their apps. Using the drag-and-drop Design Mode, each team member builds out at least one page of their team's app, responding to the feedback they received in the previous round of testing.
	Lesson 13: Linking Screens Building on the screens that they designed in the previous lesson, teams combine screens into a single app. Simple code can then be added to make button clicks change to the appropriate screen.
	Lesson 14: Testing the App In this lesson, teams run another round of user testing with their interactive prototype. Feedback gathered from this round of testing will inform the final iteration of the app prototypes.
	Lesson 15: Improving and Iterating Using the feedback from the last round of testing, teams implement changes that address the needs of their users. Each team tracks and prioritizes the features they want to add and the bugs they need to fix.
	Lesson 16: Project - App Presentation Each team prepares a presentation to "pitch" the app they've developed. This is the time students can share the struggles, triumphs, and plans for the future.

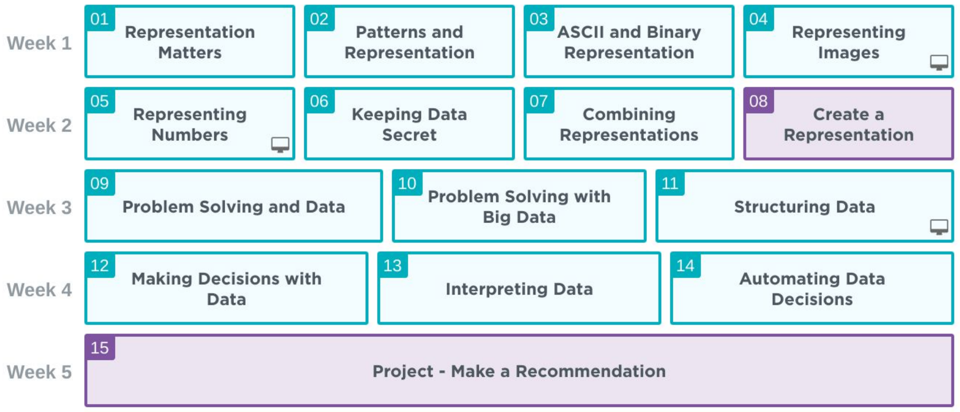


Example of paper prototype from lessons 10 and 11

Unit 5 - Data and Society

Overview and Timeline

The Data and Society unit is about the importance of using data to solve problems and it highlights how computers can help in this process. The first chapter explores different systems used to represent information in a computer and the challenges and tradeoffs posed by using them. In the second chapter, students learn how collections of data are used to solve problems, and how computers help to automate the steps of this process. In the final project, students gather their own data and use it to develop an automated solution to a problem.


Week 1
01 Representation Matters
02 Patterns and Representation
03 ASCII and Binary Representation
04 Representing Images
Week 2
05 Representing Numbers
06 Keeping Data Secret
07 Combining Representations
08 Create a Representation
Week 3
09 Problem Solving and Data
10 Problem Solving with Big Data
11 Structuring Data
Week 4
12 Making Decisions with Data
13 Interpreting Data
14 Automating Data Decisions
Week 5
15 Project - Make a Recommendation

Big Questions

Chapter 1: Representing Information

- Why is representation important in problem solving?
- What features does a representation system need to be useful?
- What is necessary to create usable binary representation systems?
- How can we combine systems together to get more complex information?

Chapter 2: Solving Data Problems

- How does data help us to solve problems?
- How do computers and humans use data differently?
- What parts of the data problem solving process can be automated?
- What kinds of real world problems do computers solve by using data?

Unit Goals

By the end of the unit, students should have a broad understanding of the role of data and data representation in solving information problems. They should be able to explain the necessary components of any data representation scheme, as well as the particulars of binary and the common ways that various types of simple and complex data are represented in binary code. Students should also be able to design and implement a data-based solution to a given problem and determine how the different aspects of this problem solving process could be automated.

Major Projects

- **Lesson 8: Project - Create a Representation**
- **Lesson 15: Project - Make a Recommendation**

There are two major projects in this unit, which are at the end of each chapter. Each project asks students to take a different approach to looking data — the first focuses on building a system to represent complex data, and the second focuses on processing data to make a recommendation. To learn more about these projects, check out the descriptions in this unit's lesson progression.

Lesson Progression: Unit 5 - Data and Society

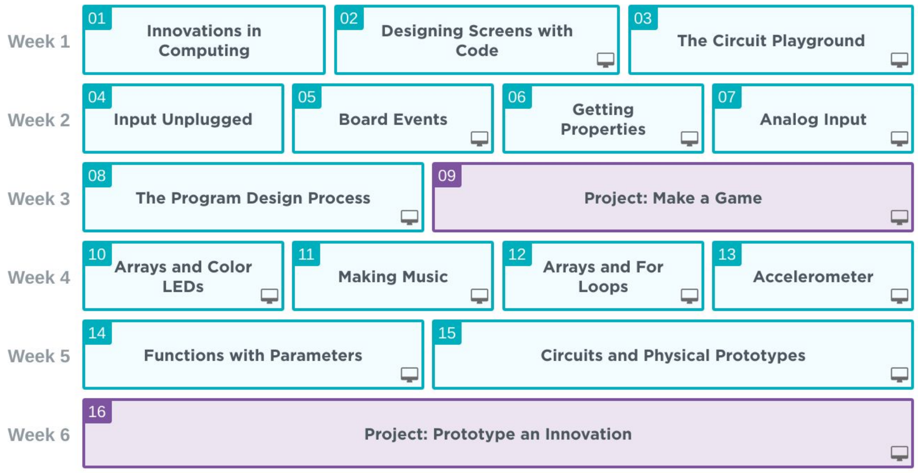
Chapter 1 Representing Information	**Lesson 1: Representation Matters** This first lesson provides an overview of what data is and how it is used to solve problems. Groups use a data set to make a series of meal recommendations for people with various criteria. Afterward, groups compare their responses and discuss how the different representations of the meal data affected how they were able to solve the different problems.
	Lesson 2: Patterns and Representation This lesson looks closer at what is needed to create a system of representation. Groups create systems that can represent any letter in the alphabet using only a single stack of cards. They then create messages with their systems and exchange with other groups to ensure the system worked as intended. Finally, the class discusses commonalities between working systems while recognizing that there are many possible working solutions.
	Lesson 3: ASCII and Binary Representation This lesson introduces students to a formal binary system for encoding information: the ASCII system for representing letters and other characters. At the beginning of the lesson, the teacher introduces the fact that computers must represent information using either "on" or "off." The class then learns about the ASCII system for representing text using binary symbols and practices using this system. Finally, they encode their own messages using ASCII.
	Lesson 4: Representing Images This lesson continues the study of binary representation systems, this time with images. The class is introduced to the concept of splitting images into squares or "pixels," which can then be turned on or off individually to make an entire image. After doing a short set of challenges using the Pixelation Widget, students make connections between the system for representing images and the ASCII system for representing text that they learned about in the previous lesson.
	Lesson 5: Representing Numbers This lesson introduces students to the binary number system. With a set of cards that represent the place values in a binary (base-2) number system, the class turns bits "on" or "off" by turning cards face up and face down, then observes the numbers that result from these different patterns. Eventually, the pattern is extended to a generic 4-bit system.
	Lesson 6: Keeping Data Secret Students have a discussion on the different levels of security they would like for personal data. Once the class has developed an understanding of the importance of privacy, they learn about the process of encrypting information by enciphering a note for a partner and deciphering the partner's note. The class concludes with a discussion about the importance of both physical and digital security.
	Lesson 7: Combining Representations This lesson combines all three types of binary representation systems (ASCII characters, binary numbers, and images) to explore ways to encode more complex types of information in a record. After seeing a series of bits and being asked to decode them, students are introduced to the idea that understanding binary information requires an understanding of both the system that is being used, and the meaning of the information encoded.
	Lesson 8: Project - Create a Representation The class designs structures to represent their perfect day using the binary representation systems they've learned in this chapter. After deciding which pieces of information the record should capture, students decide how a punch card of bytes of information will be interpreted to represent those pieces of information. Afterwards, they use the ASCII, binary number, and image formats they have learned to represent their perfect days and try to decipher what a partner's perfect day is like.

Chapter 2 Solving Data Problems	**Lesson 9: Problem Solving and Data** This lesson covers how the problem solving process can be tailored to deal with data problems. The class is tasked with deciding what a city most needs to spend resources on. They must find and use data from the internet to support their decision.
	Lesson 10: Problem Solving with Big Data This lesson covers how data is collected and used by organizations to solve problems in the real world. Students look at three scenarios that could be solved using data and brainstorm the types of data they would want to use to solve each problem, as well as strategies they could use to collect the data. Each scenario also includes a video about a real-world service that has solved a similar problem with data.
	Lesson 11: Structuring Data This lesson goes further into the interpretation of data, including how to clean and visualize raw data sets. The class first looks at how presenting data in different ways can help people to understand it better. After seeing how cleaning and visualization can help people make better decisions, students look at which parts of this process can be automated, and which parts need a human.
	Lesson 12: Making Decisions with Data This lesson gives students a chance to practice the data problem solving process introduced in the last lesson. Not all questions have right answers, and in some cases the class can and should decide that they need to collect more data. The lesson concludes with a discussion about how different people could draw different conclusions from the same data, and how collecting different data might have affected the decisions they made.
	Lesson 13: Interpreting Data Students begin the lesson by looking at a cake preference survey where respondents specified both a cake and an icing flavor. They discuss how knowing the relationship between cake and icing preference helps them better decide which combination to recommend. Students are then introduced to cross tabulation, which allows them to graph relationships to different preferences. They use this technique to find relationships in a preference survey, then brainstorm the different types of problems that this process could help solve.
	Lesson 14: Automating Data Decisions In this lesson, the class looks at a simple example of how a computer could be used to complete the decision making step of the data problem solving process. Students are given the task of creating an algorithm that suggests a vacation spot. They then create rules, or an algorithm, that a computer could use to make this decision automatically. Students share their rules and what choices their rules would make with the class data. Next, they use data from their classmates to test whether their rules would make the same decision that a person would. The lesson concludes with a discussion about the benefits and drawbacks of using computers to automate the data problem solving process.
	Lesson 15: Project - Make a Recommendation To conclude this unit, the class designs ways to use data to make recommendations or predictions to help solve a problem. In the first several steps, students brainstorm problems, perform simple research, and define a problem of their choosing. They then decide what kind of data they want to collect, how it could be collected, and how it could be used, before exchanging feedback and giving a final presentation.

Unit 6 - Physical Computing

Overview and Timeline

In the Physical Computing unit, students further develop their programming skills, while exploring more deeply the role of hardware platforms in computing. Harkening back to the Input/Storage/Processing/Output model for a computer, students look towards modern "smart" devices to understand the ways in which non-traditional computing platforms take input and provide output in ways that couldn't be done with the traditional keyboard, mouse, and monitor.



Week 1	01 Innovations in Computing	02 Designing Screens with Code	03 The Circuit Playground	
Week 2	04 Input Unplugged	05 Board Events	06 Getting Properties	07 Analog Input
Week 3	08 The Program Design Process	09 Project: Make a Game		
Week 4	10 Arrays and Color LEDs	11 Making Music	12 Arrays and For Loops	13 Accelerometer
Week 5	14 Functions with Parameters	15 Circuits and Physical Prototypes		
Week 6	16 Project: Prototype an Innovation			

Using App Lab and Adafruit's Circuit Playground, students develop programs that utilize the same hardware inputs and outputs that we see in many modern smart devices, and they get to see how a simple rough prototype can lead to a finished product. The unit concludes with a design challenge that asks students to use the Circuit Playground as the basis for an innovation of their own design.

Big Questions

Chapter 1: Programming with Hardware

- How does software interact with hardware?
- How can computers sense and respond to their environment?
- What kind of information can be communicated with hardware outputs?

Chapter 2: Building Physical Prototypes

- How do programmers work with larger amounts of similar values?
- How can complex real-world information be represented in code?
- How can simple hardware be used to develop innovative new products?

Unit Goals

By the end of the unit, students should be able to design and build a physical computing device that integrates hardware inputs and outputs with software. This unit builds on the skills and understandings from the Interactive Animations and Games unit with more sophisticated programming constructs, such as arrays, for-loops, and parameters, as well as deepens students' understanding of the types of input and output that can be used in computing. Students should leave the unit feeling equipped to use physical computing to solve problems in fun and innovative ways.

Major Projects

- **Lesson 9: Project - Make a Game**
- **Lesson 16: Project - Prototype an Innovation**

There are two major projects in this unit, which are at the end of each chapter. Each project is cumulative for the chapter completed, and each takes a different approach to hardware — the first focusing on building a game that uses the inputs and outputs of Circuit Playground and the second focusing on developing and testing an physical prototype of an innovative computing device. To learn more about these projects, check out the descriptions in this unit's lesson progression.

Tools

This unit uses **Maker Toolkit in App Lab.** For more detailed information, see the **Learning Tools** section of this curriculum guide.

Lesson Progression: Unit 6 - Physical Computing

Chapter 1 Programming with Hardware	**Lesson 1: Innovations in Computing** In this lesson, students explore a wide variety of new and innovative computing platforms while expanding their understanding of what a computer can be.
	Lesson 2: Designing Screens with Code By reading and changing the content on the screen of an app, the class starts to build apps that only need a single screen. Even with just one screen, students can begin to see that these techniques allow for lots of user interaction and functionality.
	Lesson 3: The Circuit Playground In this lesson, students get to know the Circuit Playground, the circuit board that will be used throughout the rest of this unit. Using App Lab, they develop programs that use the Circuit Playground for output.
	Lesson 4: Input Unplugged Students experience two different ways that an app can collect input from a user, while learning more about the event-driven programming model used in App Lab.
	Lesson 5: Board Events Using the hardware buttons and switch, students develop programs that use the Circuit Playground as an input.
	Lesson 6: Getting Properties This lesson introduces students to the getProperty block, which allows them to access the properties of different elements with code. They first practice using the block to determine what the user has input in various user interface elements. Students later use getProperty and setProperty together with the counter pattern to make elements move across the screen. A new screen element, the slider, and a new event trigger, onChange, are also introduced.
	Lesson 7: Analog Input Students get to explore the analog inputs on the Circuit Playground, writing programs that respond to the environment through sensors.
	Lesson 8: The Program Design Process This lesson introduces students to the process they will use to design programs of their own throughout this unit. This process is centered around a project guide that asks students to sketch out their screens, identify elements of the Circuit Playground to be used, define variables, and describe events before they begin programming (a process very similar to the Game Design Process that students used in Unit 3). Students begin by playing a tug o' war style game where the code is hidden. They discuss what they think the board components, events, and variables would need to be to make the program. Then, they are then given a completed project guide that shows one way to implement the project, and are walked through this process in a series of levels. At the end of the lesson, students have an opportunity to make improvements to the program to make it their own.
	Lesson 9: Project: Make a Game For this project, students design and create a game that leverages the new inputs and outputs that are available to them. This project is purposefully left very open-ended to empower students to think broadly about how physical output might be useful in an app, while still giving them a chance to review the program development process and try out the new features available through the Circuit Playground.

Chapter 2 Building Physical Prototypes

Lesson 10: Arrays and Color LEDs
In this lesson, students are introduced to the ring of color LEDs, which are exposed as an array called colorLeds. Students learn how to access and control each LED in an array individually, preparing them to access multiple LEDs through iteration later in the chapter.

Lesson 11: Making Music
In this lesson, students will use the Circuit Playground's buzzer feature to its full extent by producing sounds, notes, and songs. Students start with a short review of the buzzer's frequency and duration parameters, then move on to the concept of musical notes. Notes allow students to constrain themselves to frequencies that are used in Western music and provide a layer of abstraction that helps them to understand which frequencies might sound good together. Once students are able to play notes on the buzzer, they use arrays to hold and play sequences of notes, and compose simple songs.

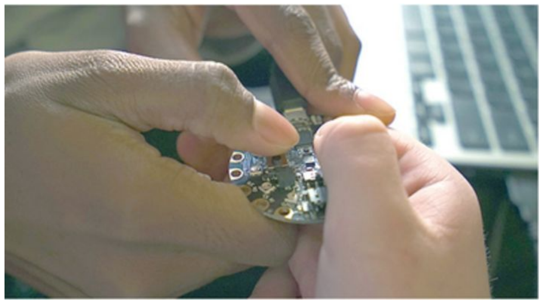
Lesson 12: Arrays and For Loops
Students learn to combine lists and for-loops in this lesson, which allows them to write code that impacts every element of a list, regardless of how long it is. The class uses this structure to write programs that process all of the elements in lists, including the list of color LEDs.

Lesson 13: Accelerometer
In this lesson, students explore the Circuit Playground's accelerometer feature and its capabilities. They become familiar with the accelerometer's events and properties as they create multiple programs with the feature, similar to those they've likely come across in real world applications.

Lesson 14: Functions with Parameters
This lesson starts with a quick review of parameters in the context of the App Lab blocks that students have seen recently. Students then look at examples of parameters within user-created functions in App Lab, and create and call functions with parameters to control multiple elements on a screen. Afterward, they use for loops to iterate over an array, passing each element into a function. Last, students use what they have learned to create a star catching game.

Lesson 15: Circuits and Physical Prototypes
In this lesson, students wire simple circuits to create a physical prototype using low-cost and easily-found materials.

Lesson 16: Project: Prototype an Innovation
This final project challenges students to develop and test a prototype for an innovative computing device that interacts with the physical world through various types of input and output, whichallow for interesting and unique user interactions. This project is an opportunity for students to showcase their technical skills, but they will also need to demonstrate collaboration, constructive peer feedback, and iterative problem solving as they encounter obstacles along the way. This project should be student-directed whenever possible, and provide an empowering and memorable conclusion to the final unit of CS Discoveries.

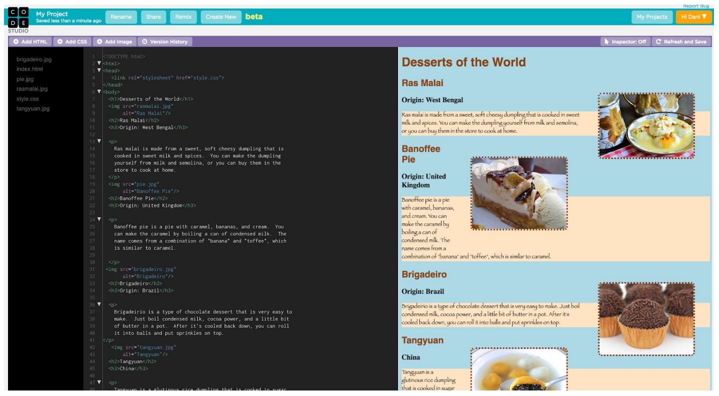

Students using the Circuit Playground

Learning Tools

Web Lab

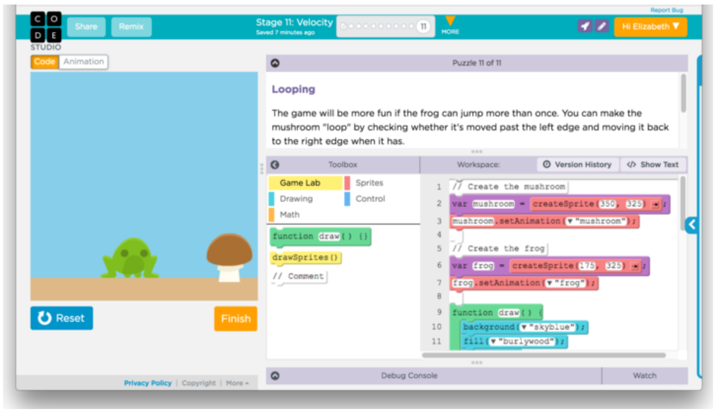
Where in the curriculum: Unit 2 - Web Development

Description
Web Lab is a browser-based text editor for building web pages in HTML and CSS. It features a text editor with many helpful tools for creating and debugging HTML and CSS code, including a live preview of the web page that updates in real time and the ability to publish the completed web page to its own unique URL. Try the tool at code.org/weblab.



Game Lab

Where in the curriculum: Unit 3 - Interactive Animations and Games
Description
Game Lab is a programming environment for developing animations and games using JavaScript. Students start by creating sprites - characters whose appearance, movement, and interactions can be controlled through code. A preloaded library of images and sounds and a full-featured pixel editor allow students to customize the look of their sprites. Then, using Game Lab, students learn fundamental programming constructs while being given the freedom to create their own virtual worlds. Students can program using either blocks or text and instantly switch between either mode. Game Lab allows for activities with a scoped toolbox of commands that focuses attention on the specific blocks and concepts being introduced in that lesson. In addition, embedded support tools help students track down errors in their code. Try the tool at code.org/gamelab.


Stage 11: Velocity

Puzzle 11 of 11

Looping

The game will be more fun if the frog can jump more than once. You can make the mushroom "loop" by checking whether it's moved past the left edge and moving it back to the right edge when it has.

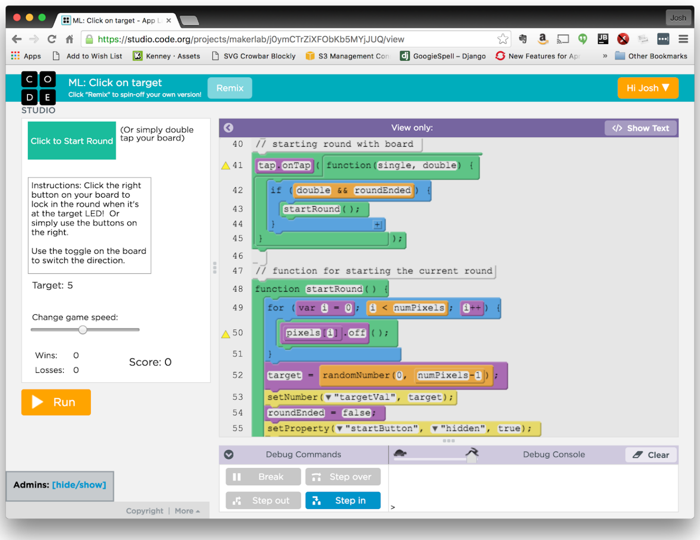
```
// Create the mushroom
var mushroom = createSprite(350, 325);
mushroom.setAnimation("mushroom");

// Create the frog
var frog = createSprite(175, 325);
frog.setAnimation("frog");

function draw() {
  background("skyblue");
  fill("burlywood");
```

App Lab

Where in the curriculum: Unit 4 - The Design Process
Description
App Lab is a programming environment for developing applications in JavaScript. A drag-and-drop editor allows students to add and edit page elements without having to write the associated HTML and CSS. Working in either blocks or text, students define the behavior of these page elements using code. The scoped toolbox of commands focuses attention on the specific blocks or concepts being introduced in that lesson. The embedded support tools help students track down errors in their code. Thanks to these features, App Lab is particularly well-suited for quickly prototyping apps. Try the tool at code.org/applab.


ML: Click on target

Instructions: Click the right button on your board to lock in the round when it's at the target LED! Or simply use the buttons on the right.

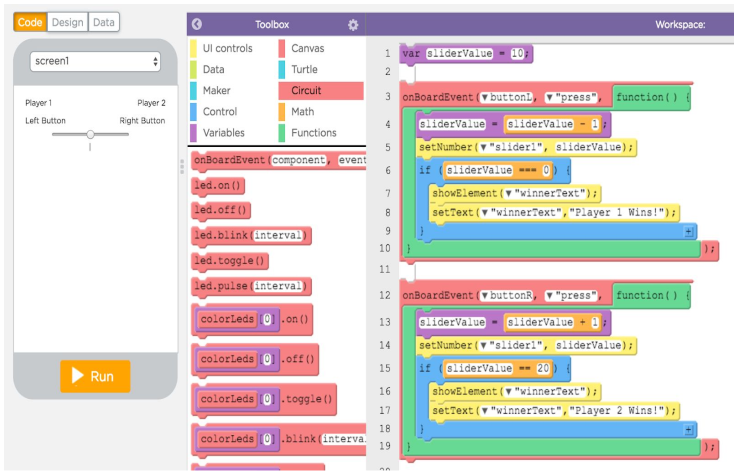
Use the toggle on the board to switch the direction.

```
// starting round with board
tap.onTap(function(single, double) {
  if (double && roundEnded) {
    startRound();
  }
});

// function for starting the current round
function startRound() {
  for (var i = 0; i < numPixels; i++) {
    pixels[i].off();
  }
  target = randomNumber(0, numPixels-1);
  setNumber("targetVal", target);
  roundEnded = false;
  setProperty("startButton", "hidden", true);
```

Maker Toolkit

Where in the curriculum: Unit 6 - Physical Computing



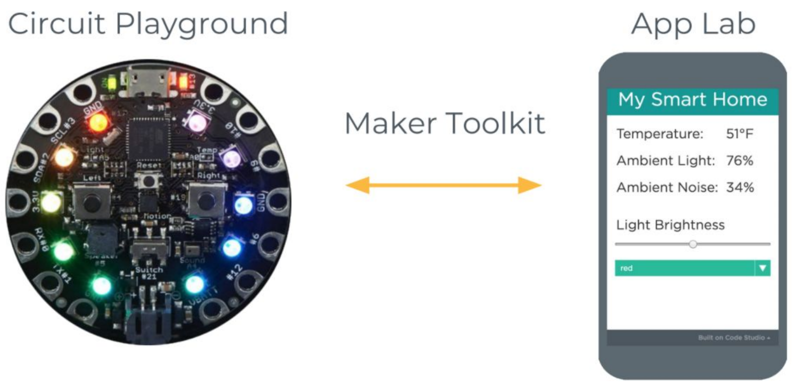
```
var sliderValue = 10;

onBoardEvent("buttonL", "press", function( ) {
  sliderValue = sliderValue - 1;
  setNumber("slider1", sliderValue);
  if (sliderValue === 0) {
    showElement("winnerText");
    setText("winnerText", "Player 1 Wins!");
  }
});

onBoardEvent("buttonR", "press", function( ) {
  sliderValue = sliderValue + 1;
  setNumber("slider1", sliderValue);
  if (sliderValue == 20) {
    showElement("winnerText");
    setText("winnerText", "Player 2 Wins!");
  }
});
```

Description

The Maker Toolkit is an environment that allows you to communicate with the Circuit Playground using a set of additional commands available in App Lab. Using the same drag-and-drop editor that students have become comfortable with in App Lab and Game Lab, students can turn on LEDs, read sensors, and write programs that use physical hardware for user input and output. By integrating these commands, App Lab allows you to quickly prototype apps that combine hardware and software without the additional challenge of transitioning to a new language or tool or worrying about wiring and electronics. Try the tool at studio.code.org/maker/setup.



Unplugged and Plugged Activities

Unplugged Activities

What are they?

We refer to activities where students are not working on a computer as "unplugged." Students will often be working with pencil and paper, or physical manipulatives.

How are they used?

Unplugged activities are more than just an alternative for the days when the computer lab is full. They are intentionally placed, often kinesthetic, opportunities for students to digest concepts in approachable ways. Unplugged lessons are particularly good for building and maintaining a collaborative classroom environment, and they are useful touchstone experiences you can refer to when introducing more abstract concepts.

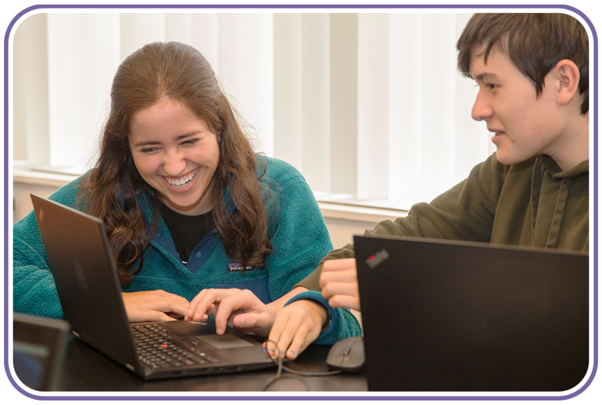
Tips for Effectively Teaching Unplugged Activities

- Don't skip these activities!
- Teach units in the order they are written. The sequence is designed to scaffold student understanding.
- Help students identify the computer science concepts underlying these approachable activities.
- Refer back to unplugged activities to reinforce concepts in subsequent plugged lessons.

Plugged Activities

What are they?

We refer to activities where students are working on a computer as "plugged." Students may be conducting research, completing a programming assignment, or using an interactive "widget".



How are they used?

Plugged activities are designed to allow students to get hands-on with tools and concepts. That said, plugged lessons typically have many of the same features of their unplugged counterparts. Lessons will begin and end with discussions or activities that help motivate and synthesize learning. Students are encouraged and often even required to work with one another. Key moments for you to check in with your students are noted in lesson plans. Students will be using a computer, but the ways students interact with each other and your role as the teacher should remain largely unchanged.

Tips for Effectively Teaching Plugged Activities

- 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 more easily established during unplugged activities.

"Plugged" doesn't mean the computer is the students' teacher! If anything, 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.

Teaching and Learning Strategies

The teaching and learning strategies listed below are generally useful across different lessons and units. We believe these strategies lead to positive classroom culture and, ultimately, student learning.

Pair Programming

What is it?

Pair programming is a technique in which two programmers work together at one computer. One, the driver, writes code while the other, the navigator, directs the driver on 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 the “real world” working environment

How does it connect to the curriculum?

In CS Discoveries, there are many lessons on the computer (plugged lessons) during which students develop programming skills through online progressions. Pair programming can help to foster a sense of camaraderie and collaboration in your classroom during sets of plugged lessons. It has been shown to increase the enrollment, retention, and performance of students in computer science classes. It promotes diversity in the classroom by reducing the so-called “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. Assign 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 plugged 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: bit.ly/pair-programming-feature

Videos :

- For Teachers: bit.ly/pair-programming- teacher (Created for CS Fundamentals, but still applicable)
- For Students: bit.ly/pair-programming- student

The National Center for Women & Information Technology (NCWIT) has a great resource about the benefits of pair programming. Check it out at: http://bit.ly/pair-programming-ncwit

Think-Pair-Share

What is it?
Think-Pair-Share is a three part activity in which students are presented with a problem or task to work on.

Think: First, students work individually. Working individually gives students the opportunity to collect their thoughts before communicating them with others. They should write down their thoughts in a journal for later sharing.

Pair: Once students have had time to work individually, they then enter the “Pair” stage in which they work with a small group. These groups can consist of two or three students. The group discusses the thoughts each member collected during the “Think” stage. The goal is for students to engage in a low-risk discussion where they get a chance to share their ideas with others. This activity is especially useful in the early stages of developing collaborative skills such as attentive listening to a partner.

Share: Finally, the groups share out some of the ideas they discussed to the whole class, and the discussion will continue as needed in the whole group setting. This allows major ideas to bubble up to the whole group, where everyone can hear and benefit from them.

How does it connect to the curriculum?
Almost every lesson in the CS Discoveries curriculum involves some kind of discussion that uses a version of Think-Pair-Share. It is one of the most common practices used for warm ups and wrap ups. Think-Pair-Share is used for these discussions as it gives students time to think on their own and engage with the content before talking to someone else. When students talk to their partner, it should be a low risk environment to try out an idea. It also allows everyone to play a part in the discussion, even if they don’t like talking in the whole class environment.

How do I use it?
- Whenever you are given a prompt, consider giving students time to work individually and then with a partner before bringing the discussion or creation to the whole class.
- View Think-Pair-Share as a way for the class to learn from each other as much as possible. As the lead learner, you should direct the conversation without giving away answers or cutting off the conversation too early.

Peer Feedback

What is it?
Peer feedback is the practice where students give each other feedback on work they have done. The feedback is meant to provide opportunities for students to learn from each other, both by seeing ways others approached the same problem and by incorporating feedback to improve their own work.

How does it connect to the curriculum?
Throughout the CS Discoveries curriculum, there are many activities that have structured moments for students to give each other peer feedback. We support these activities with structured guides for the peer feedback process. Many of the guides follow a similar format where students are first given the opportunity to express what they would like feedback on. Then the peer reviewer gives feedback on some standard questions, often related to the goals of the work, as well as leaving some free response feedback using the sentence starters "I like," "I wish," and "What if." Finally, students are encouraged to reflect on the feedback they received and think about ways to incorporate it in the future.

How do I use it?
- Create a structured peer feedback process.
- Decide who is giving feedback to whom.
- Allow students to share some areas that they would like feedback on.
- Give students time to provide feedback.
- Give students time to respond and incorporate feedback.
- Provide examples of constructive feedback.
- Have students use sentence starters for their feedback such as: I like, I wish, What if
- Treat this as a skill that students develop throughout the course and which they will need to be taught.

Debugging

What is it?

Debugging is the act of finding and fixing problems in code. It's a major part of programming and a critical skill for students to develop. Your role as the teacher is to avoid directly debugging for your students, but to help guide them in the development of their own debugging skills.

How does it connect to the curriculum?

Occasionally students will encounter a debugging level where they are explicitly asked to identify and fix bugs in provided code. While these are great opportunities to highlight specific kinds of errors and misconceptions, it's important to build a culture of constant debugging, as this isn't an activity that is done in isolated moments.

As with most things, people get better at debugging by doing it! That said, reflective strategies can help students learn more from debugging. Encourage students to talk about their bugs and how they were able to address them. Students may create bug logs in their journals or a bug poster for the classroom. If students are too specific with the bugs that they have found, consider reframing what they say into something more generally useful (e.g. "The 'r' and the 'c' were switched." could become "My keyword was not spelled correctly.")

How do I use it?

- Ask questions about the code (and what changes were made when the bug was introduced), making sure that the students can clearly explain how the code is intended to work.
- Have students read their code aloud, line by line, explaining the purpose of each command.
- Encourage students to ask aloud the same questions that you have been asking them.
- Avoid finding the bug for students or being too specific with your questioning.
- Celebrate discovering (and fixing) new types of bugs to normalize the debugging process.

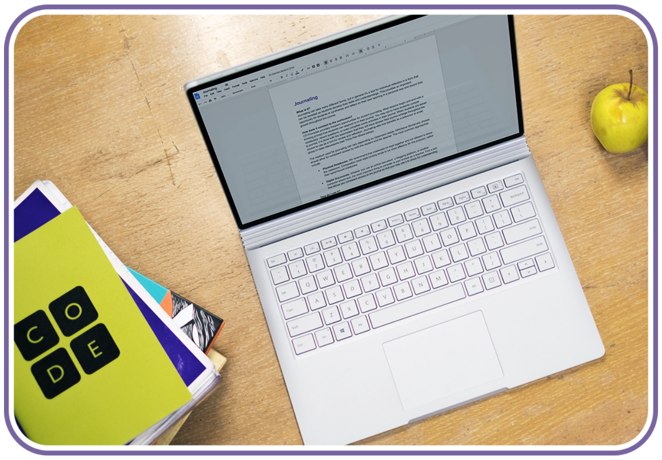
Resources

See Appendix C for a more detailed teacher facing guide to debugging along with a student facing debugging resource.

Journaling

What is it?

Journaling can take many different forms, but in general it is a tool for individual reflection in a form that can be revisited as students develop their skills and understandings. This provides an important opportunity for students to reflect on their own learning in a personal way and record their growth throughout the course.



How does it connect to the curriculum?

CS Discoveries frequently provides opportunities for student journaling, often as a tool for individual sensemaking after having completed an activity as a class. When students are asked to journal, it is done with the assumption that they will have access to their journal writings throughout the course as a tool for review and reflection. Occasionally students are also asked to revisit specific journal prompts. The medium used for journaling can vary, depending on classroom needs. Whichever format you choose should allow for consistent access by both the student and the teacher. The most common approaches include:

- **Physical Notebooks:** We recommend that notebooks be kept together and not allowed to leave the classroom. Composition book style binding tends to be more effective for this purpose, rather than spiral-bound notebooks.
- **Digital Documents:** Whether you use Google Docs, a blogging platform, or another computer-based tool, the most important thing to consider is your access as a teacher. Find a tool that allows you consistent access to the journal so that you may use it to check for understanding.

How do I use it?

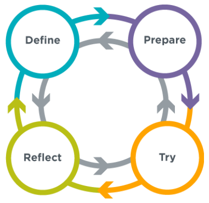
- Provide students a journal at the beginning of the school year.
- Prompt students to journal about specific challenges or bugs they encounter.
- Give students time to revisit previous journal entries and reflect on their growth.

Student Practices

The work students do in CS Discoveries is connected by a core set of practices developed over time. These student practices provide coherence and represent the high-level skills and dispositions students develop throughout the course. In the curriculum you will find reminders of moments when students can reflect on this development, and all major projects include an opportunity for student reflection on their growth in each practice.

Problem Solving	Use a structured problem solving process to help address new problems View challenges as solvable problems Break down larger problems into smaller components
Persistence	Expect and value 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
Creativity	Incorporate personal interests and ideas into activities and projects 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 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

Problem Solving Process



The Problem Solving Process is a tool for structured problem solving that is woven throughout the entire course to promote student growth and development. Having a strategy for approaching problems can help you develop new insights and come up with new and better solutions. This is an iterative process that is broadly useful for solving all kinds of problems:

Define
- Determine the problem you are trying to solve
- Identify your constraints
- Describe what success will look

Prepare
- Brainstorm / research possible solutions
- Compare pros and cons
- Make a plan

Try
- Put your plan into action

Reflect
- Compare your results to the goals you set while defining the problem
- Decide what you can learn from this or do better next time
- Identify any new problems you have discovered

Each unit leverages the problem solving process in a different way.

Unit 1 Problem Solving and Computing	Unit 2 Web Development	Unit 3 Animations and Games	Unit 4 The Design Process	Unit 5 Data and Society	Unit 6 Physical Computing
The Problem Solving Process	The Problem Solving Process for Programming	The Problem Solving Process for Programming	The Problem Solving Process for Design	The Problem Solving Process for Data	The Problem Solving Process for Programming

The Problem Solving Process for Programming

Used in: Unit 2 - Web Development, Unit 3 - Interactive Animations and Games, and Unit 6 - Physical Computing

Define

- Read the instructions carefully to ensure you understand the goals
- Rephrase the problem in your own words
- Identify any new skills you are being asked to apply
- Look for other problems you've solved that are similar to this one
- If there is starter code, read it to understand what it does

Prepare

- Write out an idea in plain language or pseudocode
- Sketch out your idea on paper
- List what you already know how to do and what you don't yet know
- Describe your idea to a classmate
- Review similar programs that you've written in the past

Try

- Write one small piece of code at a time
- Test your program often
- Use comments to document what your code does
- Apply appropriate debugging strategies
- Go back to previous steps if you get stuck or don't know whether you've solved the problem

Reflect

- Compare your finished program to the defined problem to make sure you've solved all aspects of the problem
- Ask a classmate to try your program and note places where they struggle or exhibit confusion
- Ask a classmate to read your code to make sure your documentation is clear and accurate
- Try to "break" your program to find types of interactions or input the program could handle better
- Identify a few incremental changes you could make in the next iteration

The Problem Solving Process for Design

Used in: Unit 4 - The Design Process

Define

- Identify potential users
- Interview users
- Read user profiles
- Identify needs and wants

Prepare

- Connect needs and wants to specific problems
- Research how others have addressed these issues
- Brainstorm potential solutions
- Discuss pros and cons
- Identify the minimum work needed to test your assumptions

Try

- Draw your product on paper
- Develop a low fidelity prototype to communicate your design
- Share prototypes with potential end users for feedback

Reflect

- Present to stakeholders
- Review user feedback

The Problem Solving Process for Data

Used in: Unit 5 - Data and Society

Define

- Decide what problem you are trying to solve or what question you are trying to answer
- Make sure you understand your target audience (it could be you!) and what specifically they need
- Identify the parts of your problem you could address with data, and how more information could help

Prepare

- Decide what kinds of data you will collect
- Decide how you will collect the data and in which format you will collect it
- Anticipate possible challenges in data collection and modify your plan to account for them
- Develop a plan for how you will analyze your data and make sure your data will be useful for that kind of analysis

Try

- Collect the data using the plan you created
- Clean the data by removing errors, unexpected values, and inconsistencies
- Visualize the data by creating tables, graphs, or charts that help you see broad trends
- Interpret the trends and patterns in the visualizations based on your knowledge of the problem

Reflect

- Review what you've learned about your question or problem
- Decide if what you've learned has solved your problem and allows you to make a decision, or if you need to go back to one of the previous steps

Course Resources

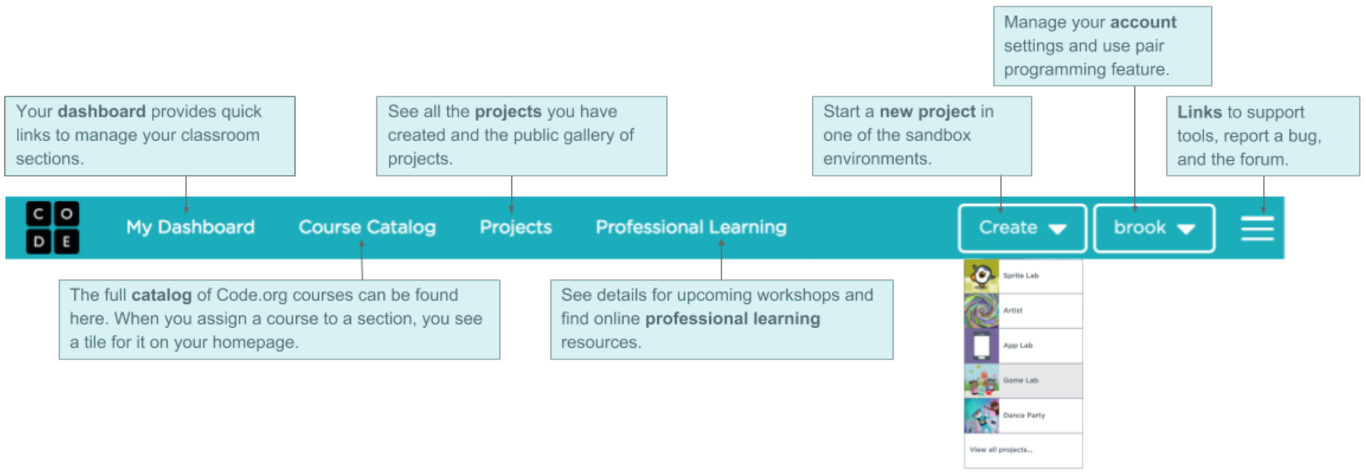
The CS Discoveries curriculum is made up of student-facing and teacher-facing components. Teachers will access curriculum materials in two different places on the Code.org website: our Code Studio platform and in the teacher-facing curriculum. The table below outlines what you can do in each of these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



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.

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

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.

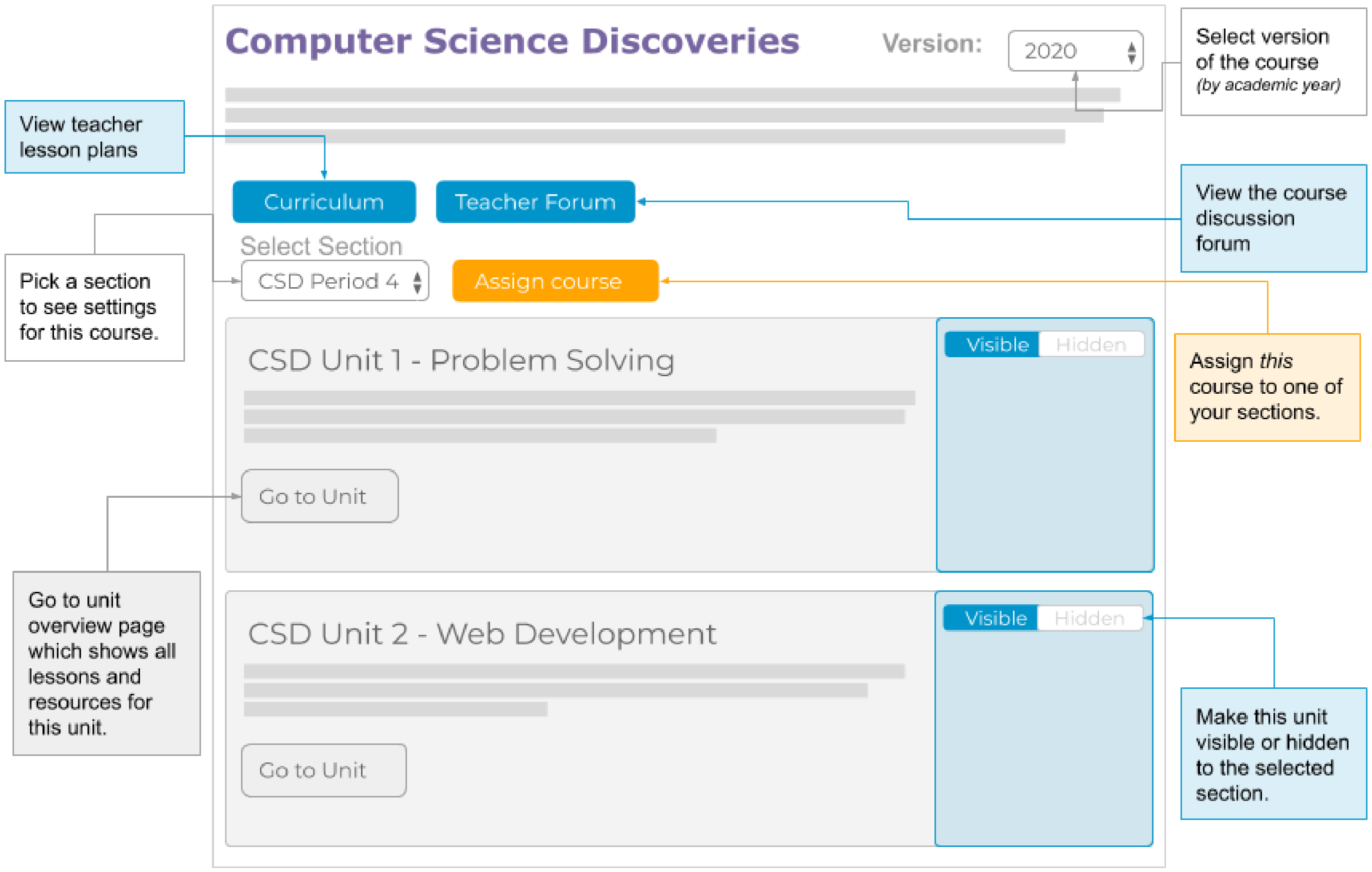
The Code Studio home page is the starting point for everything in the course. To get started with your students, you will need to create a section. For details on how to create a section, visit the **getting started** support articles at support.code.org.

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

Course Overview Pages

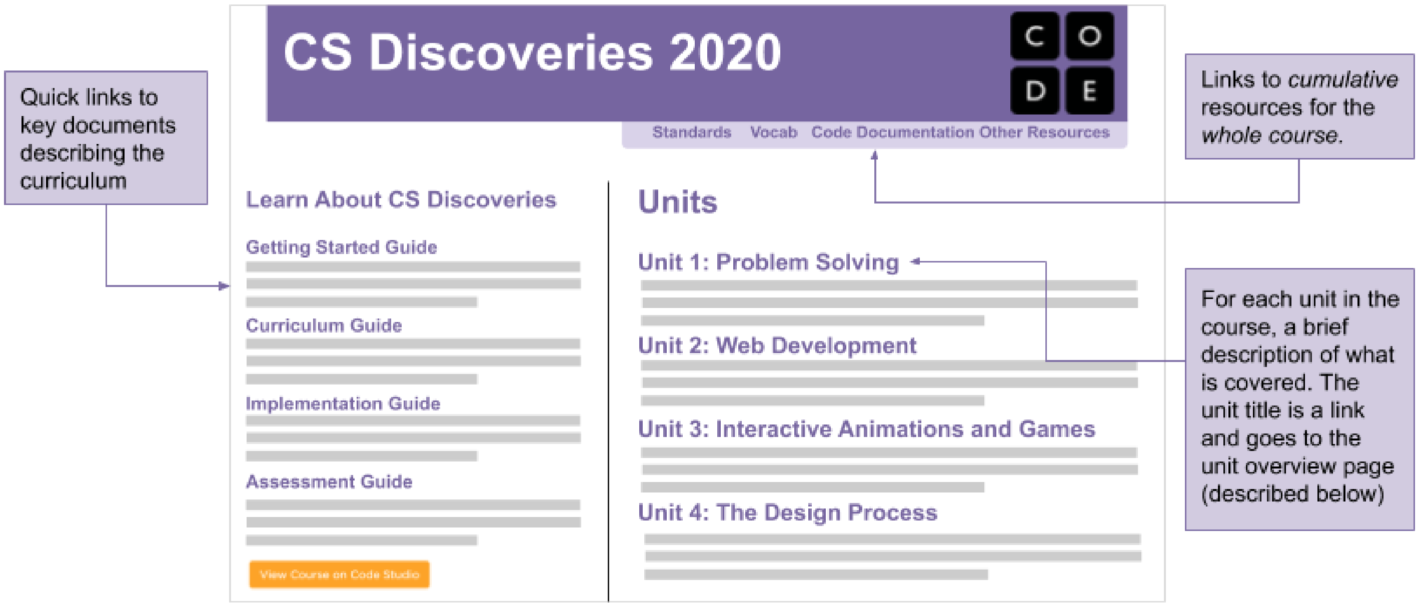
Code Studio — Course Overview

The course overview page on Code Studio [studio.code.org/course/csd] is a hub for managing your course, and includes the following:


Computer Science Discoveries
Version: 2020
Select version of the course (by academic year)
View teacher lesson plans
Curriculum
Teacher Forum
View the course discussion forum
Select Section
CSD Period 4
Assign course
Pick a section to see settings for this course.
Assign this course to one of your sections.
CSD Unit 1 - Problem Solving
Visible Hidden
Go to Unit
Go to unit overview page which shows all lessons and resources for this unit.
CSD Unit 2 - Web Development
Visible Hidden
Go to Unit
Make this unit visible or hidden to the selected section.

Teacher-facing Curriculum — Course Overview

This page [curriculum.code.org/csd] provides an overview of the teacher-facing curriculum, and includes the following:

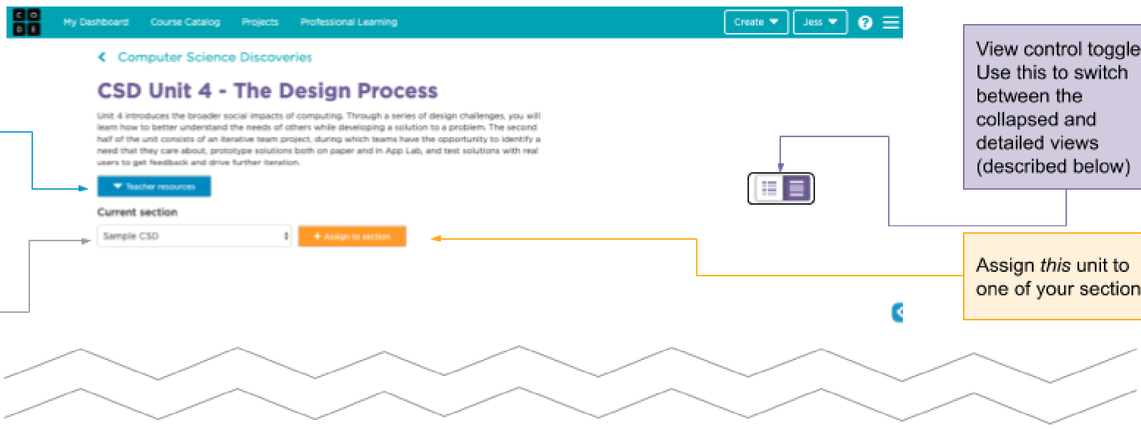

CS Discoveries 2020
Standards Vocab Code Documentation Other Resources
Links to cumulative resources for the whole course.
Quick links to key documents describing the curriculum
Learn About CS Discoveries
Getting Started Guide
Curriculum Guide
Implementation Guide
Assessment Guide
View Course on Code Studio
Units
Unit 1: Problem Solving
Unit 2: Web Development
Unit 3: Interactive Animations and Games
Unit 4: The Design Process
For each unit in the course, a brief description of what is covered. The unit title is a link and goes to the unit overview page (described below)

Unit Overview Pages

Code Studio — Unit Overview

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

Pick a section to see settings for this unit.



Computer Science Discoveries

CSD Unit 4 - The Design Process

Unit 4 introduces the broader social impacts of computing. Through a series of design challenges, you will learn how to better understand the needs of others while developing a solution to a problem. The second half of the unit consists of an iterative team project, during which teams have the opportunity to identify a need that they care about, prototype solutions both on paper and in App Lab, and test solutions with real users to get feedback and drive further iteration.

Teacher resources

Current section: Sample CSD

Assign to section

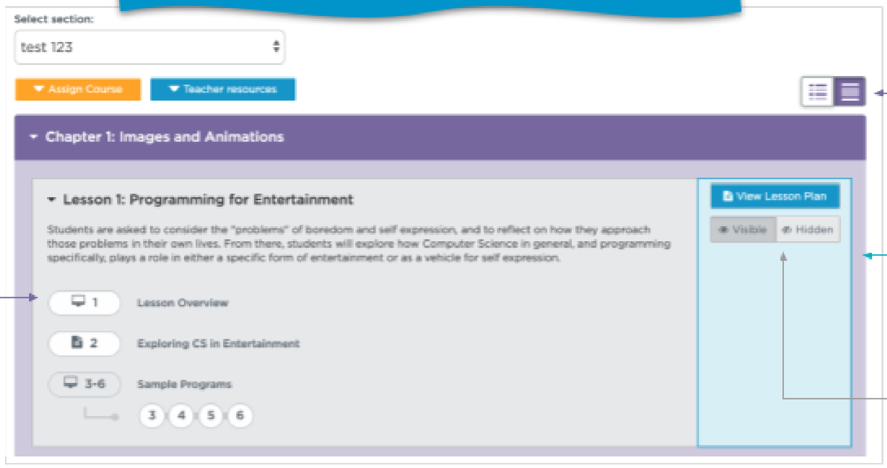
View control toggle. Use this to switch between the collapsed and detailed views (described below)

Assign *this* unit to one of your sections.

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

Detailed view:

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



Select section: test 123

Chapter 1: Images and Animations

Lesson 1: Programming for Entertainment

Students are asked to consider the "problems" of boredom and self expression, and to reflect on how they approach those problems in their own lives. From there, students will explore how Computer Science in general, and programming specifically, plays a role in either a specific form of entertainment or as a vehicle for self expression.

1 Lesson Overview

2 Exploring CS in Entertainment

3-6 Sample Programs

3 4 5 6

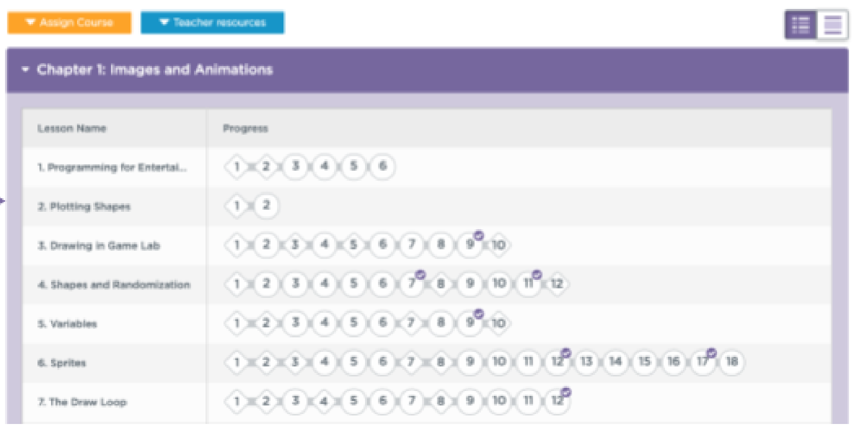
View control toggle in detailed view

Link to the teacher lesson plan

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

Collapsed view:

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



Chapter 1: Images and Animations

Lesson Name	Progress
1. Programming for Entertai...	1 2 3 4 5 6
2. Plotting Shapes	1 2
3. Drawing in Game Lab	1 2 3 4 5 6 7 8 9 10
4. Shapes and Randomization	1 2 3 4 5 6 7 8 9 10 11 12
5. Variables	1 2 3 4 5 6 7 8 9 10
6. Sprites	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
7. The Draw Loop	1 2 3 4 5 6 7 8 9 10 11 12

View control toggle in collapsed view

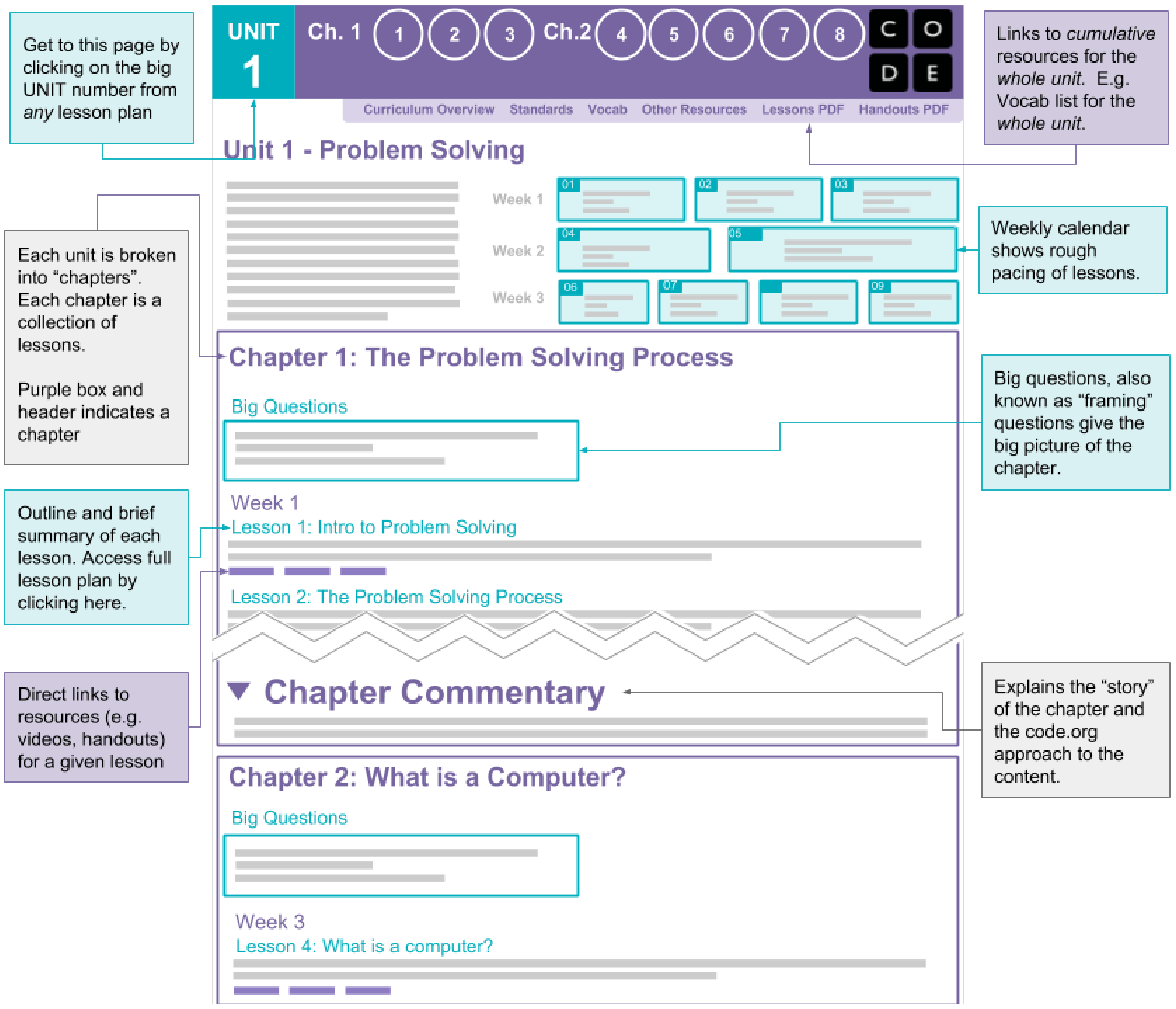
Code Studio - Iconography on Unit Overview Page

When looking at the detailed view of the unit overview page (e.g., studio.code.org/s/csd1) 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.

✓ Assessment	This small check mark symbol superimposed over the level is an indication for you, the teacher, that the level is a good candidate for assessing what your students have learned. This symbol does not guarantee that the level has been automatically checked for correctness. Instead, it's there to help guide you in selecting levels to focus on for assessment. Assessment check marks can be found on "question" and "online" level types (see below for details).
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.
Choice	Choice levels contain multiple activities for students to choose from. Some choice levels are practice levels intended to provide students with multiple opportunities to practice skills learned in the previous levels. Some choice levels are challenge levels intended to allow students an opportunity to explore new concepts not covered in the lesson objectives.
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, and 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 found in the post-project test, found at the end of each unit. In these cases, the items are meant to be *summative* assessment items.
Online	These levels use a Code.org tool, widget, or programming environment like App Lab. An instructions panel appears on these levels to help explain any new content introduced in the level, provide a checklist of tasks to complete, and may include starter code. Teachers can review their students' code from the Teacher Panel. Using these levels, you can 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 — Unit Overview

High-level planning should start by looking at the unit overview page on curriculum.code.org/csd.



Get to this page by clicking on the big UNIT number from *any* lesson plan

Each unit is broken into "chapters". Each chapter is a collection of lessons.

Purple box and header indicates a chapter

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

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

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

Weekly calendar shows rough pacing of lessons.

Big questions, also known as "framing" questions give the big picture of the chapter.

Explains the "story" of the chapter and the code.org approach to the content.

UNIT 1 Ch. 1 1 2 3 Ch.2 4 5 6 7 8

Curriculum Overview Standards Vocab Other Resources Lessons PDF Handouts PDF

Unit 1 - Problem Solving

Week 1 Week 2 Week 3

Chapter 1: The Problem Solving Process

Big Questions

Week 1

Lesson 1: Intro to Problem Solving

Lesson 2: The Problem Solving Process

Chapter Commentary

Chapter 2: What is a Computer?

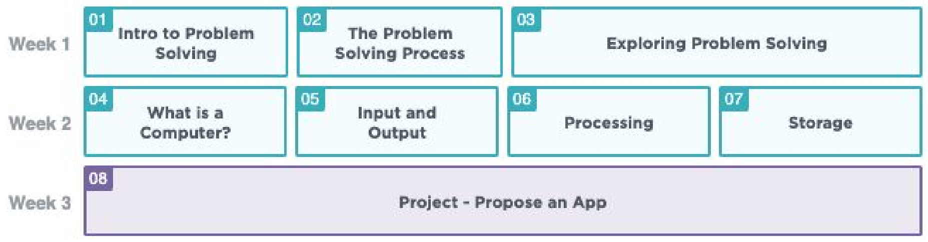
Big Questions

Week 3

Lesson 4: What is a computer?

Pacing Calendar

The calendar on the unit overview page shows the relative "size," or length of each lesson and suggests what you might be able to get through in a week. Larger projects are marked in purple.



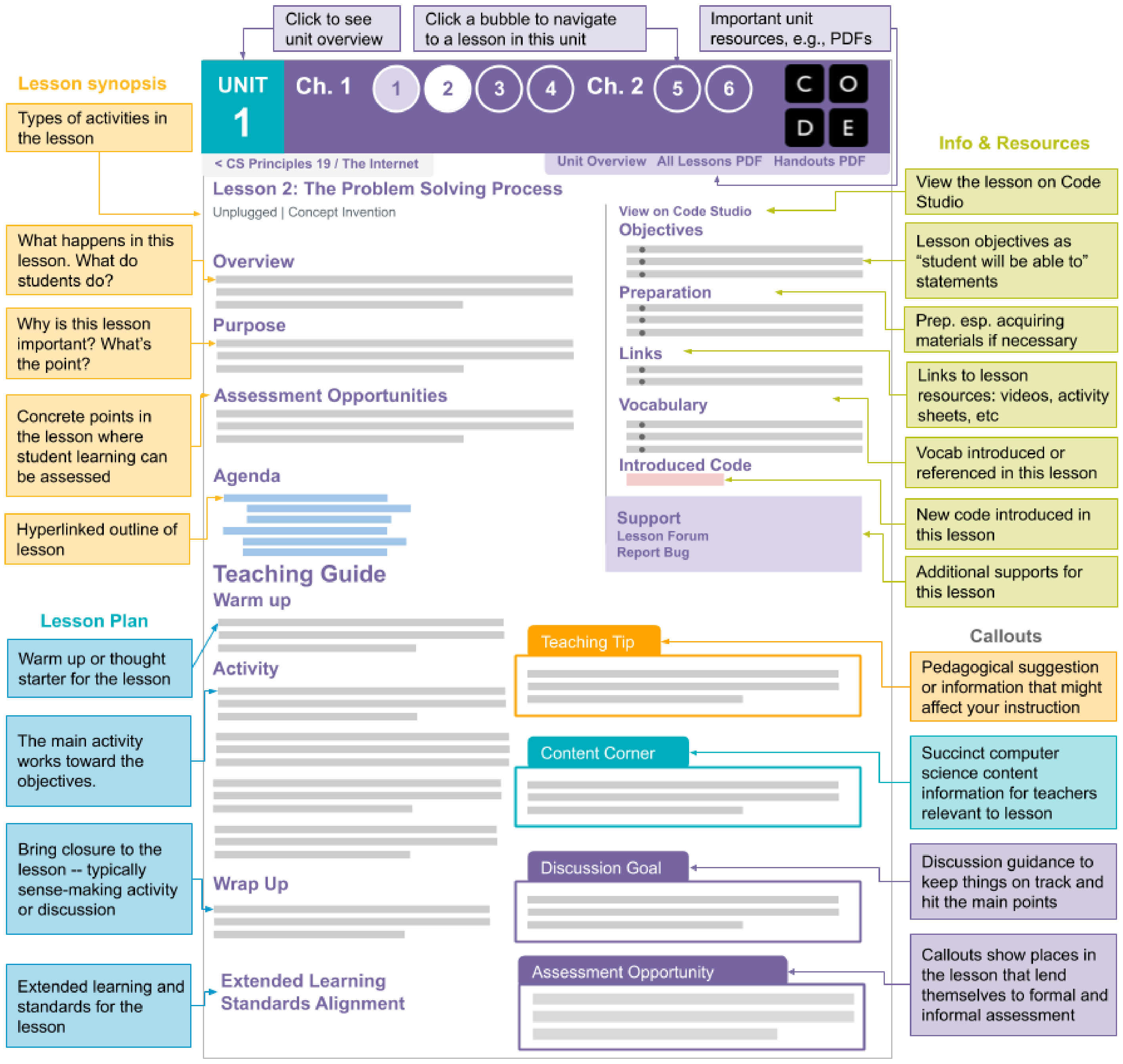
Week 1	01 Intro to Problem Solving	02 The Problem Solving Process	03 Exploring Problem Solving	
Week 2	04 What is a Computer?	05 Input and Output	06 Processing	07 Storage
Week 3	08 Project - Propose an App			

If you finish the set of lessons for a week ahead of schedule, you can absorb the time if the previous week went long, or start the next week early, time permitting.

Lesson Structure and Iconography

Teacher-facing Curriculum — Lesson Plans

Every lesson plan has a common structure designed to make it easy for you 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.

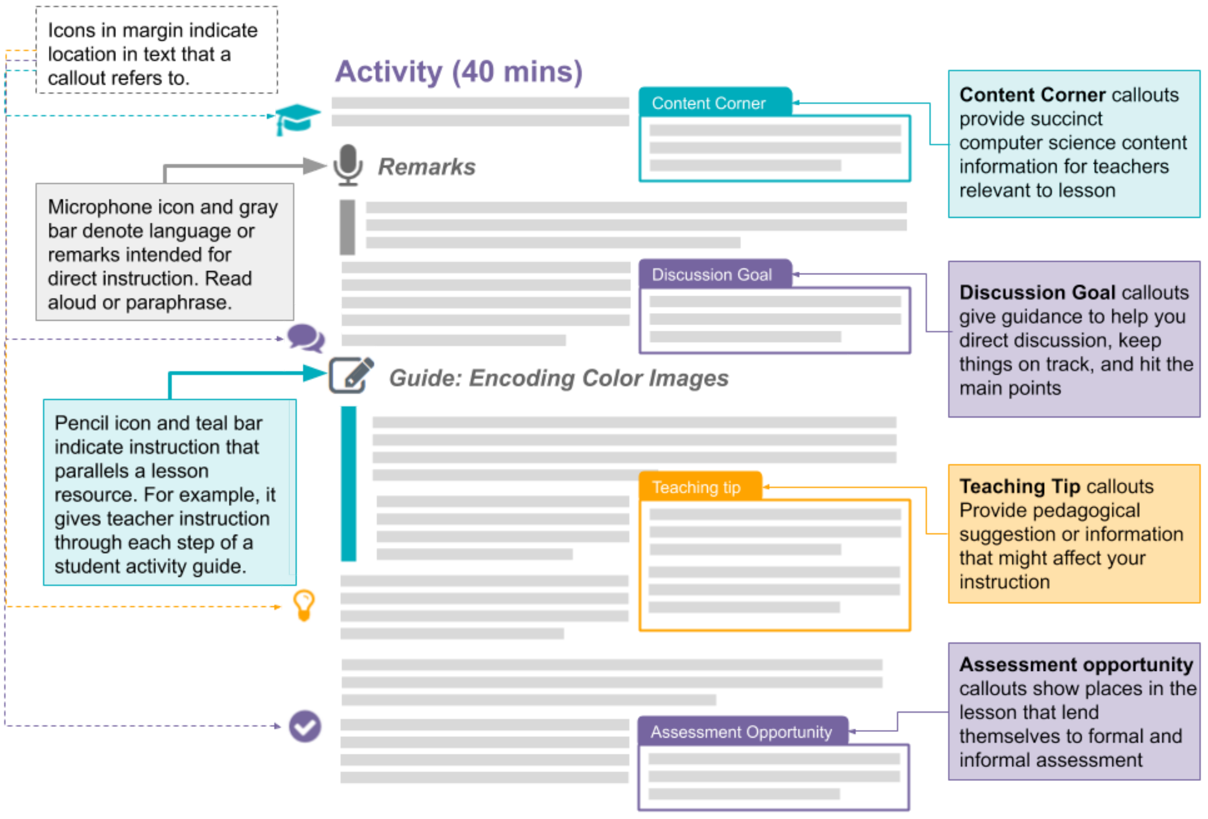

Click to see unit overview
Click a bubble to navigate to a lesson in this unit
Important unit resources, e.g., PDFs
UNIT 1
Ch. 1
1
2
3
4
Ch. 2
5
6
C O D E
< CS Principles 19 / The Internet
Unit Overview
All Lessons PDF
Handouts PDF
Lesson synopsis
Types of activities in the lesson
Lesson 2: The Problem Solving Process
Unplugged | Concept Invention
What happens in this lesson. What do students do?
Overview
Why is this lesson important? What's the point?
Purpose
Concrete points in the lesson where student learning can be assessed
Assessment Opportunities
Hyperlinked outline of lesson
Agenda
Teaching Guide
Warm up
Activity
Wrap Up
Extended Learning
Standards Alignment
Lesson Plan
Warm up or thought starter for the lesson
The main activity works toward the objectives.
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
View on Code Studio
Objectives
Lesson objectives as "student will be able to" statements
Preparation
Prep. esp. acquiring materials if necessary
Links
Links to lesson resources: videos, activity sheets, etc
Vocabulary
Vocab introduced or referenced in this lesson
Introduced Code
New code introduced in this lesson
Support
Lesson Forum
Report Bug
Additional supports for this lesson
Callouts
Teaching Tip
Pedagogical suggestion or information that might affect your instruction
Content Corner
Succinct computer science content information for teachers relevant to lesson
Discussion Goal
Discussion guidance to keep things on track and hit the main points
Assessment Opportunity
Callouts show places in the lesson that lend themselves to formal and informal assessment

Lesson Length

Lessons in CS Discoveries are written for a wide variety of classrooms. We generally try to make one lesson equal to one 45 to 60 minute class period. However, some lessons take multiple days, such as projects or concepts that do not easily break down into separate lesson plans. Many lessons include time estimates, but these may vary based on the age of your students, their background with the material, or their interests.

Lesson Plan Iconography

Within lesson plans, you'll notice a number of icons and other kinds of callouts, which 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 that a callout refers to.

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

Assessment opportunity callouts show places in the lesson that lend themselves to formal and informal assessment

Interactive Code Studio View (inside lesson plans)

Lesson plans give you an interactive view into all of the text content and instructions that 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 in Code Studio.

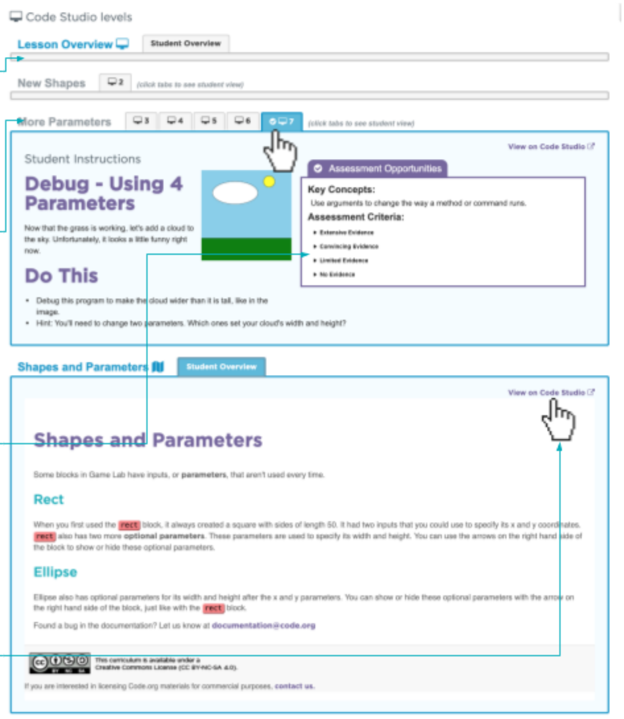
This should greatly speed up your preparations for class or serve as a quick way to remind yourself what's in each lesson.

Programming levels from Code Studio are pulled directly into the lesson plan.

Clicking on a level shows the instructions or content displayed on that level in Code Studio.

Assessment opportunities are marked with a check mark, just like the one you see in Code Studio. For levels that include a mini-rubric, the assessment criteria show in a purple call out box

Clicking "view on Code Studio" takes you directly to the corresponding level in Code Studio.



Student Instructions

Debug - Using 4 Parameters

Now that the grass is working, let's add a cloud to the sky. Unfortunately, it looks a little funny right now.

Do This

- Debug this program to make the cloud wider than it is tall, like in the image.
- Hint: You'll need to change two parameters. Which ones set your cloud's width and height?

Assessment Opportunities

Key Concepts:
Use arguments to change the way a method or command runs.

Assessment Criteria:
- Extensive Evidence
- Convincing Evidence
- Limited Evidence
- No Evidence

Shapes and Parameters

Some blocks in Game Lab have inputs, or **parameters**, that aren't used every time.

Rect

When you first used the rect block, it always created a square with sides of length 50. It had two inputs that you could use to specify its x and y coordinates. rect also has two more **optional parameters**. These parameters are used to specify its width and height. You can use the arrows on the right hand side of the block to show or hide these optional parameters.

Ellipse

Ellipse also has optional parameters for its width and height after the x and y parameters. You can show or hide these optional parameters with the arrow on the right hand side of the block, just like with the rect block.

Found a bug in the documentation? Let us know at documentation@code.org

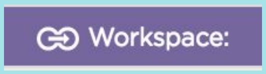
This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, contact us.

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.

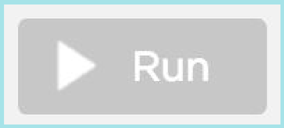
Project Levels



These programming levels share code with one another but have different instructions. Project levels allow students to build up projects across a lesson or unit. An alert box informs students when they are working in a project level.

Make sure students understand that these levels share code across them, so any work they do in one level will affect all of the related levels. If you want to see student progress over time, you can ask them to "remix" their code at certain points and share it with you as a standalone project.

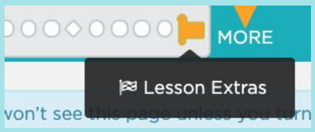
Prediction Levels



These levels ask students to make a multiple choice or free-response prediction about the output of a program. Students are prevented from running code (indicated by a grayed-out "Run" button) until they lock in a prediction. Teachers can view student predictions from the Teacher Panel.

These levels can be done as a whole group or in pairs. Give students a chance to explore and discuss working code before hitting the "Run" button. Make sure students don't feel pressure to be 'correct,' and use predictions as a starting point for discussions of how the code works

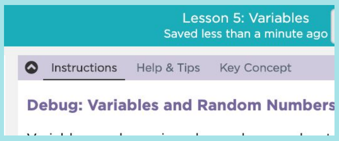
Lesson Extras



These are extra activities found at the end of the lesson. Students can find challenge tasks or Free Play levels that allow them to use what they have learned in an open ended project.

You can assign particular tasks to some students, or allow them to pick and choose what they would like to do on their own. Free Play levels provide a scoped toolbox, but without any particular instructions or starter code. You can use these for classroom demonstrations or to assign your own programming task to students.

Tabs in the Instruction panel on programming levels


Lesson 5: Variables
Saved less than a minute ago
Instructions
Help & Tips
Key Concept
Debug: Variables and Random Numbers

For programming lessons, the instruction panel at the top of the environment also includes sections with important information: Instructions, Help & Tips, and Key Concept, as well as Feedback, should you choose to provide it for your students.

Instructions: a description of the task that students are trying to accomplish in the level.

Help and tips: links to any map levels and videos that are relevant to the task.

Feedback: Teachers can see the feedback tab on any programming level, when looking at a student's work. Students see it once the teacher leaves feedback.

Key Concepts: Programming levels that have been identified as assessment opportunities (as indicated by the check icon) include a key concepts tab, which outlines the programming concept that students should be demonstrating in that level. These levels provide space for you to evaluate students via a mini-rubric and leave written feedback on their work.

Assessment and Feedback

Frequent assessment and feedback are critical to ensuring that students are actively involved in their own learning and that teachers have evidence that their class is making progress. We have incorporated opportunities for both formal and informal evaluation throughout the CS Discoveries curriculum to support you in measuring student growth and informing the pace of your instruction. However, since schools have diverse grading systems, it is up to you to decide how to use the assessment resources for grading purposes.

Each unit includes a learning framework document that outlines the expected student outcomes assessed throughout the unit and within that unit's major projects and post-project test. These outcomes are organized into concept clusters to help students and teachers understand the broad goals of each unit, and, when taken together, address the Big Questions included in the unit's description. Learning objectives and assessment opportunities listed in individual lesson plans connect back to the unit's overall framework.

Specific student outcomes within these learning frameworks may be mapped to external standards such as the CSTA K-12 Computer Science Standards or other similar state-level standards for computer science.

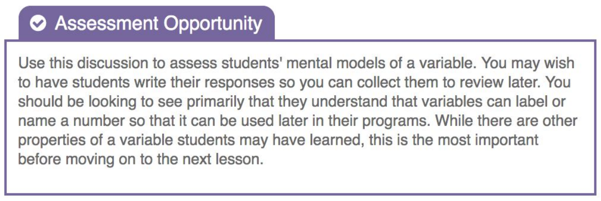
To allow students and teachers flexibility in reaching these learning goals, the course does not assess completion of specific activities.

Opportunities for assessment and feedback

We believe that teachers in classrooms are in the best position to assess and give feedback on student performance. In most cases, we expect that teachers evaluate student work using the criteria and rubrics provided in the lesson plans.

Lesson-level Assessment

Opportunities for assessment within each lesson are listed out in the lesson plans themselves. These opportunities may include sections of the activity guides, discussion and reflection questions, or online programming and quick check levels. Guidance for how to use these opportunities to assess student progress is included in each lesson plan.


Assessment Opportunity

Use this discussion to assess students' mental models of a variable. You may wish to have students write their responses so you can collect them to review later. You should be looking to see primarily that they understand that variables can label or name a number so that it can be used later in their programs. While there are other properties of a variable students may have learned, this is the most important before moving on to the next lesson.

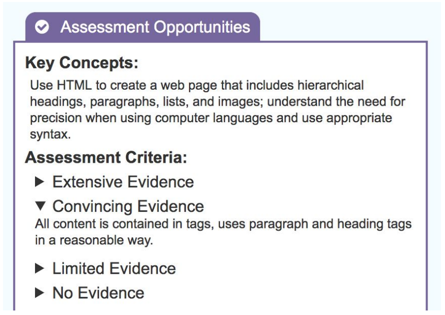
Teachers have access to student activity throughout the lesson, including input on prediction levels, the code for all programming levels, and reflective activities students may engage in. While teachers might choose to assess and give feedback on student performance in any of these situations, some places have specifically been marked as useful for assessing student progress. Focusing on these areas can free teachers from feeling the need to look at every level a student has completed.

Class discussions provide an opportunity for group sensemaking and for teachers to informally assess student understanding. These discussions may begin with students writing down their individual thoughts before sharing with a partner or group. The goals of each discussion and how teachers might use them to evaluate learning are included in the lesson plan as callout instructions.

Journal questions allow students to reflect on what they have learned, and what they hope to learn more about. Journal prompts often accompany discussion questions, and guidance for assessment is also included in the lesson plan.

Quick-check levels include multiple choice or short answer questions. These are usually given after students have had a chance to explore a concept. They check for common misunderstandings before students move on to the next lesson or task. Students are able to get feedback from the system immediately, and revise their answers before moving on to the next task. Each quick-check level includes teacher notes detailing the learning objective being assessed.

Programming levels challenge students to complete a small programming task. Teachers have access to all student work in these levels, can read and run the code that students have produced, and can leave feedback for students about their work. These levels also include exemplar solutions for teachers to reference and a mini-rubric that provides assessment criteria and the learning objective being assessed.


Assessment Opportunities

Key Concepts:
Use HTML to create a web page that includes hierarchical headings, paragraphs, lists, and images; understand the need for precision when using computer languages and use appropriate syntax.

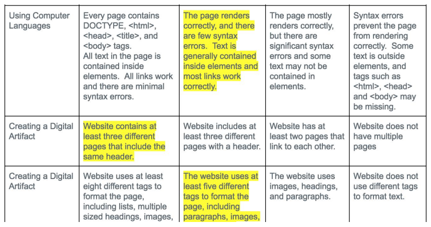
Assessment Criteria:
▶ Extensive Evidence
▼ Convincing Evidence
All content is contained in tags, uses paragraph and heading tags in a reasonable way.
▶ Limited Evidence
▶ No Evidence

Activity Guides accompany unplugged lessons in the curriculum. They include prompts and questions that teachers can use to follow students' progress through the lesson and reflection questions that can give insight into what students have learned from the activity. Guidance for assessment is included in the lesson plan and any related exemplars.

Chapter- and Unit-level Assessment

End of chapter projects incorporate the skills and understandings students have developed while progressing through that chapter's lessons. These projects are designed to assess unit-specific skills and the five student practices that thread through the entire course. There is broad guidance for the activity, but project implementation leaves room for students to put their personal stamp on the creation.

Student-facing rubrics give guidance on the skills they must demonstrate, while allowing for plenty of choice in how to show what they have learned. These rubrics are directly related to the expected student outcomes listed in the learning framework and give criteria for evaluation of the various objectives of the unit.



Using Computer Languages	Every page contains DOCTYPE, <html>, <head>, <title>, and <body> tags. All text in the page is contained inside elements. All links work and there are minimal syntax errors.	The page renders correctly, and there are few syntax errors. Text is generally contained inside elements and most links work correctly.	The page mostly renders correctly, but there are significant syntax errors and some text may not be contained in elements.	Syntax errors prevent the page from rendering correctly. Some text is outside elements, and tags such as <html>, <head> and <body> may be missing.
Creating a Digital Artifact	Website contains at least three different pages that include the same header.	Website includes at least three different pages with a header.	Website has at least two pages that link to each other.	Website does not have multiple pages
Creating a Digital Artifact	Website uses at least eight different tags to format the page, including lists, multiple sized headings, images,	The website uses at least five different tags to format the page, including paragraphs, images,	The website uses images, headings, and paragraphs.	Website does not use different tags to format text.

Teacher-facing sample projects and associated marked rubrics provide guidance in how student work can be assessed. The sample projects demonstrate varied levels of mastery of the different skills assessed by the project, to allow teachers to compare their own student work to the marked exemplars.

Practice reflections are more open ended, with space for students and teachers to reflect on how the five student practices played into the student experience with the project. We've intentionally made these reflections the same throughout all projects in the course, so that students and teachers can track progress over time.

Post-project tests are included at the end of every unit. These include several multiple choice and matching questions as well as open ended reflections on the final project of the unit. These tests are aligned to the learning framework of each unit and are designed to assess parts of the framework that may not have been covered by the project rubrics.

Individual assessment of group activities

Many CS Discoveries activities are designed to be completed in pairs or small groups. While group work is essential to help students develop the collaborative skills that are a key focus of the course, teachers may find it difficult to assess each individual student's learning. Consider using reflection questions that dig into both the student's role in the project and in the student's understandings and skills for more insight into individual learning outcomes.

Planning for the Year

Pacing

CS Discoveries is designed to be taught as a single semester, two semesters over multiple years, or as one full year course. The following pacing guide gives a rough recommendation of unit length, assuming that your class meets five days a week for at least 45 minutes per session. Check out Appendix B for alternative implementation options.

Semester 1	**Unit 1: Problem Solving Computing** *3 weeks*		**Unit 2: Web Development** *7 weeks*		**Unit 3: Animations & Games** *9 weeks*	
	Chapter 1 *1 week*	Chapter 2 *2 weeks*	Chapter 1 *4 weeks*	Chapter 2 *3 weeks*	Chapter 1 *5 weeks*	Chapter 2 *4 weeks*
Semester 2	**Unit 4: The Design Process** *6 weeks*		**Unit 5: Data & Society** *5 weeks*		**Unit 6: Physical Computing** *6 weeks*	
	Chapter 1 *2 weeks*	Chapter 2 *4 weeks*	Chapter 1 *2 weeks*	Chapter 2 *3 weeks*	Chapter 1 *3 weeks*	Chapter 2 *3 weeks*

Tech Requirements and Required Materials

Technical Requirements

The course requires and assumes a 1:1 computer lab or setup such that each student has access to an internet-connected computer every day in class. Each computer must have a modern web browser installed. All of the course tools and resources (lesson plans, teacher dashboard, videos, student tools, programming environment, etc.) are online and accessible through a web browser. For more information about tech setup go to: code.org/educate/it

While the course features many "unplugged" activities away from the computer, daily access to a computer is essential for every student. However, it is not required that students have access to internet-connected computers at home. Since almost all of the materials are online, internet access at home is certainly an advantage. PDFs of handouts, worksheets and readings are available on the course website.

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, post-it notes (or similar sized scrap paper), rulers or a straight edge of some kind, index cards (or similar sized scrap paper)

In addition to those general course materials, the following items are called for in specific units:

- Unit 1 - Problem Solving and Computing
  - Aluminum foil, container for water, pennies (note that pennies can be replaced with some other kind of weight of the same size.) - Alternate activities are available if you do not have access to these supplies.
- Unit 6 - Physical Computing
  - Classroom set of Circuit Playgrounds (1 board and micro USB cable for every 2 students). Check out code.org/circuitplayground for more details.

Appendix A: Professional Learning Handouts

NOTE: This appendix was created to support teachers who are participating in the Code.org Professional Learning Program.

Build your Action Plan: Getting to Fall

Use the space below to make a plan for preparation you want to complete between now and the start of the school year.

Your open questions		
What *questions do you have?*	***Where*** *can you find answers to those questions?*	***When*** *do you plan to get answers to those questions?*

Things to explore further		
Which *topics in the curriculum do you want to further explore before you start teaching the course?*	***Who*** *can you work with in exploring these topics?*	***When*** *do you plan to do this exploration?*

Pacing and Planning: Instructional Units

Use the space below to document your pacing plan for moving through each of the instructional units and performance tasks.

What	*Duration*	When do you plan to start?	When do you plan to finish?	Notes or special considerations
Unit 1 ***Problem Solving and Computing***	*3 weeks*			
Unit 2 ***Web Development***	*7 weeks*			
Unit 3 **Interactive Animations and Games**	*9 weeks*			
Unit 4 **The Design Process**	*6 weeks*			
Unit 5 **Data and Society**	*5 weeks*			
Unit 6 **Physical Computing**	*6 weeks*			

Appendix B: CS Discoveries Getting Started and Implementation Options

NOTE: The following two documents (Getting Started and Intended Implementation Options) were created to support teachers who are not participating in the Code.org Professional Learning Program but are shared here because they are generally useful documents for any CS Discoveries teacher.

Getting Started

Basic Course Information

Computer Science Discoveries is an introductory, classroom-based course appropriate for 6 - 10th grade students. It is designed with the new-to-CS student and teacher in mind and can be taught as a semester or year-long course (3-5 hours per week of instruction for 9+ weeks).

The course takes a wide lens on computer science by covering topics such as programming, physical computing, web development, design, and data. The course inspires students as they build their own websites, apps, games, and physical computing devices. Our curriculum is available at no cost for anyone, anywhere in the world.

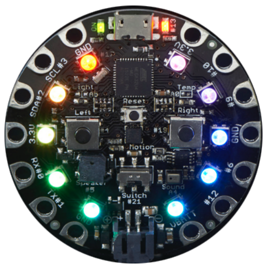
Options for Implementation

CS Discoveries consists of two semesters that build on each other. Schools can choose to teach a single semester, two sequential semesters, or a single, year-long course. For schools with less than a semester, we always suggest starting with the Problem Solving and Computing unit. Afterward, the class can move on to Web Development or Interactive Animations and Games. The second chapter of each unit can be skipped if pressed for time. Read more about our implementation options in the section below.

What You'll Need

The course requires that students have access to computers with a modern web browser. At this time, our courses are not optimized for tablets or mobile devices. For more details, check out Code.org's technology requirements at code.org/educate/it.

In addition to computer access, you'll need typical classroom supplies, such as pencils, paper, scissors and glue, as well as an ability to print lesson handouts for students. A few activities may require some specific supplies such as a deck of cards or aluminum foil. You can see a full list of resources needed for each lesson at https://curriculum.code.org/csd/resources/.



Adafruit's Circuit Playground Boards and Micro USB cables are required for our physical computing unit. The curriculum is designed for a ratio of 2 students to 1 board & 1 usb cable. For more details, check out our Circuit Playground page at code.org/circuitplayground.

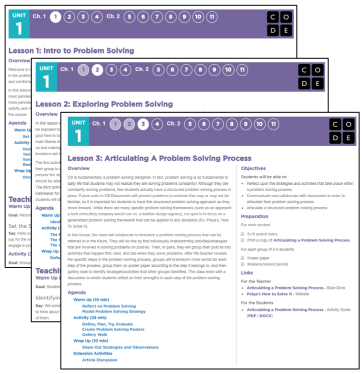
Learning More

For more in-depth information about the CS Discoveries curriculum, go to http://curriculum.code.org/csd/. This site includes unit overviews, detailed lesson plans, student handouts, standards alignment, and links to resources such as online code documentation. It also includes the CS Discoveries Curriculum Guide, which guides teachers though everything they need to know about the course, from our course content and pedagogical approach to specific instructions for using Code.org tools.

Getting Started

To get started with the curriculum, you'll need to create and verify a teacher account. Go to https://studio.code.org/users/sign_up to set up your Code.org teacher account. This will give you access to lesson plans, activity guides, and all online activities that your students will see. To access answer keys and other restricted materials, go to https://code.org/verified. The verification process may take up to seven business days.

Running a Code.org Lesson


Lesson 1: Intro to Problem Solving
Lesson 2: Exploring Problem Solving
Lesson 3: Articulating A Problem Solving Process

CS Discoveries is written with the new-to-CS teacher in mind, meaning that you do not need to be an expert in computer science to run the lesson. Teachers do, however, play an active role as the lead learner in the course, establishing a collaborative and safe learning culture, facilitating sensemaking discussions, and helping students to overcome challenges in the activities through modelling effective learning strategies.

Each lesson plan includes teaching tips, discussion goals, and extra information on content to help you fulfill your role in the classroom. The most valuable preparation for a new-to-CS teacher, though, may be going through the course activities yourself to better understand and empathize with the challenges your students face when engaging with the new material.

To avoid technical problems, make sure that you can play the course videos and access published Web Lab, Game Lab, and App Lab projects on student devices. You may need to check with your IT department to allow these sites though your school's firewall. For a full list of websites to unblock, visit the Code.org IT Requirements at code.org/educate/it.

Where to go for help

The Code.org forum is an active online community where teachers can ask questions, find resources, and collaborate. Verified teachers have access to teacher-only boards where they can discuss answers and share restricted resources. The forum is a great place to find supplementary materials that teachers have created to suit their particular classrooms. https://forum.code.org

To read help articles or get support directly from Code.org staff, go to support.code.org. You can search for answers to frequently asked questions, report bugs, and give other feedback on the curriculum and tools.

Every 21st century student should have the opportunity to learn computer science. The basics of computer science help nurture creativity and problem-solving skills, preparing students for a future in any field or career.

Intended Implementation Options

CS Discoveries is designed to be taught as a single semester or full-year course. The following pacing guide gives a rough recommendation for unit length, assuming the class meets five days a week for at least 45 minutes per session.

If teaching the course as a semester or full year, we recommend teaching the units in the order they appear in the curriculum, which is also presented below. Some schools may choose to split the course over multiple years, in which case students should complete the first semester before the second.

Semester 1	**Problem Solving & Computing** *3 weeks*		**Web Development** *7 weeks*		**Interactive Animations & Games** *9 weeks*	
	Chapter 1 *1 week*	Chapter 2 *2 weeks*	Chapter 1 *4 weeks*	Chapter 2 *3 weeks*	Chapter 1 *5 weeks*	Chapter 2 *4 weeks*
Semester 2	**The Design Process** *6 weeks*		**Data & Society** *5 weeks*		**Physical Computing** *6 weeks*	
	Chapter 1 *2 weeks*	Chapter 2 *4 weeks*	Chapter 1 *2 weeks*	Chapter 2 *3 weeks*	Chapter 1 *3 weeks*	Chapter 2 *3 weeks*

Guidelines for Further Adjustments

- **Always start with the Problem Solving and Computing unit, which introduces core frameworks and classroom norms.**
- **Afterward, the class can move to Web Development or Interactive Animations and Games.**
- **The second chapter can be skipped if pressed for time.**

A few sample pacing guides that follow these guidelines are provided below, but you may choose to create your own.

5-Weeks: Web Development

Problem Solving & Computing	**Web Development**
Chapter 1 *1 week*	Chapter 1 *4 weeks*

8-Weeks: Interactive Animations

Problem Solving & Computing		**Interactive Animations and Games**
Chapter 1 *1 week*	Chapter 2 *2 weeks*	Chapter 1 *5 weeks*

10-Weeks: Web Development

Problem Solving & Computing		**Web Development**	
Chapter 1 *1 week*	Chapter 2 *2 weeks*	Chapter 1 *4 weeks*	Chapter 2 *3 weeks*

12-Weeks: Condensed Semester

Problem Solving & Computing		**Web Development**	**Interactive Animations & Games**
Chapter 1 *1 week*	Chapter 2 *2 weeks*	Chapter 1 *4 weeks*	Chapter 1 *5 weeks*

Appendix C: Additional Guides and Resources

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?

Guide to Debugging - Teacher Facing

Introduction to Debugging

Debugging is the process of finding and fixing problems in code. For most programs, the time spent debugging far outweighs the time spent writing new code. Whether students or professional engineers, all programmers get bugs, and debugging is a normal part of the programming process.

Although students may see bugs as inconveniences to be eliminated as soon as possible, bugs in student programs should be seen as opportunities to reinforce positive attitudes toward debugging and persistence, identify and address student misconceptions, and further develop good debugging skills. Your role as the teacher is to avoid directly debugging for your students, but to guide students in better taking advantage of these opportunities.

Reinforcing positive attitudes

Finding bugs is the first step toward fixing them. Programmers deliberately test their code in order to uncover any possible bugs. Celebrate the discovery of new bugs as students report them in the classroom, framing finding a bug as the first step to fixing it. Model enjoying the interesting or funny behaviors a bug can cause, such as making a sprite move in unexpected ways, or distorting an image on a web page. Remind students that if programs all worked exactly as they wanted the first time, programming wouldn't be as interesting or fun. Encourage students as they get frustrated, reinforcing the debugging strategies and students' self-efficacy as they improve their debugging skills, and talk about the bugs that you get in your own programs, reminding them that everyone gets bugs, even professional software developers.

Identify and address misconceptions

Often a bug occurs because students have misconceptions around how the computer interprets their code. As part of the debugging process, have students explain their code, line by line, reminding them that the program is really only doing exactly as it was told. If the program displays an error message, ask the student to connect the message to what is happening in the code itself. Prompt them with relevant questions about how the associated programming structures (such as conditionals or loops) work, and refer them back to previous lessons, worked examples, or documentation when needed. After students have found the bug, ask them to reflect on what they learned about the associated programming structures and how they could explain it to a friend with a similar bug.

Develop debugging skills

As students get more experience debugging their programs, they will build the skills they need to debug more independently. Help students to generalize the strategies that they use by asking them to reflect on what processes were effective and reframing those processes as a general strategy. They should also learn ways to simplify the debugging process by making their code more readable with good naming conventions, clear formatting, and relevant comments; organizing code into functions or other logical chunks where appropriate; and testing small pieces of code as they go. Call out these practices as facilitating debugging as you help students with their code.

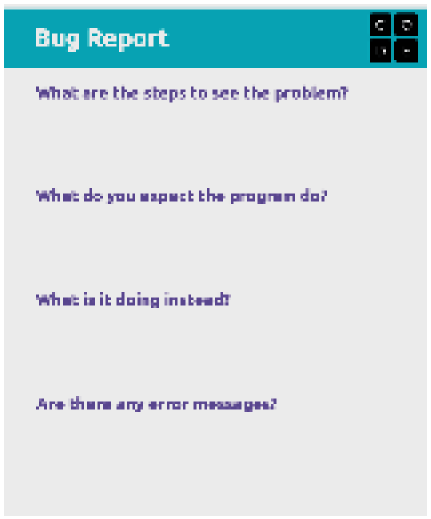
Debugging as problem solving

In computer science, debugging is framed as a form of problem solving, and students can use a version of the four step Problem Solving Process as a framework for debugging their programs. Just as in other forms of problem solving, many students may jump to the "Try" part of the framework and begin making changes to their code before understanding the nature of the bug itself. Remind them that all parts of the process are important, and that ignoring the other three steps will actually make debugging more time consuming and difficult in the long run.

Define - Describe the bug

In the context of debugging, defining the problem is describing the bug. This step can be done by anyone using the program, not just the person who will eventually be debugging it. Students will need to know the following information before they move on to the next step:

- When does it happen?
- What did you expect the program to do?
- What did it do instead?
- Are there any error messages?


Bug Report

What are the steps to see the problem?

What do you expect the program do?

What is it doing instead?

Are there any error messages?

Some bugs will keep the code from running at all, while others will run, but not correctly, or will run for a while, then stop or suddenly do something unexpected. All of those things are clues that will help the student find the bug in the next step.

You can encourage students to clearly describe the bugs they find by having them write up bug reports, such as in this worksheet. As you foster a positive culture around debugging, encourage students to write bug reports for their classmates' code as well as their own.

Prepare - Hunt for the bug

In most cases, hunting for the bug (the "prepare" step in the debugging process) will take up most of a programmer's time. Remind students that it's natural to take a long time to find a bug, but that there are things that will make their search easier.

1. **Why is the bug happening?**
   Often students focus on what they want the program to do and why they believe their code is correct, rather than investigating what could be causing the bug. Encourage students to start with the error messages and what is actually happening when the program is run, and try to connect that with the code that they have written, rather than explain why their code "should" be working. They can also "trace" their code, by reading it line by line, not necessarily from top to bottom, but in the order that the computer would interpret it while the program is running. Using debugging tools such as watchers, breakpoints, or the inspector tool may help them to identify why the code is running as it is.
2. **What changed right before the bug appeared?**
   If students have been testing their code along the way (as they should!), they can focus on code that they have recently changed. As they investigate that code, they should follow the logic of their program in the same order that the code runs, rather than reading the code line by line from top to bottom.
3. **How does this code compare to "correct" solutions?**
   Students should also make use of the various resources available to them, such as working projects, examples in previous lessons, and code documentation. Have them compare the patterns that they find in the documentation and exemplars to their own code, differentiating between the programming patterns that should be the same (loops, counter pattern, HTML syntax) and specifics of their program that will be different (variable names, image URLs, coordinates).

Try - Change the code

If students believe that they have found the bug, they can go ahead and try to fix it, but in many cases they may want to make changes to the code to narrow down their search. Some common strategies include:

1. **Commenting out code**
   By commenting out sections of the program, students can narrow down the part of the program that is causing the bug. After commenting out large sections of code, function calls, or html elements, test whether the bug is eliminated. This method is especially helpful when students have used good modular programming techniques.
2. **Print to the console**
   Using the console log command can help students to understand whether a conditional has been triggered, or keep track of a quickly changing variable over a period of time.
3. **Change the starting values of variables**
   Changing the starting values of variables can make it easier to reproduce a certain bug. For example, students may want to change the starting score to 99 to test a bug that only occurs when the score reaches 100. They may want to set their number of lives to a very high number to allow them to test for longer before losing the game.
4. **Amplify small effects**
   Sometimes bugs have such tiny effects that it's difficult to investigate them. Amplifying small effects, such as making elements move further on the screen or giving web page elements a background color to make them more visible, can make it easier to understand what is happening in a program.

In many cases, students will introduce new bugs during the debugging process, especially if they are randomly changing code as part of a "guess and check" method. Prompt them to explain why they are changing the code, and encourage them to make small changes and test them often, making it easier to go back if their solution didn't work.

Reflect - Document what happened

As students make changes and see the effects, they should reflect and document on their experiences. This will help them to build a better model of how the program constructs work and what debugging strategies are most effective. Students should consider the following after they have eliminated a bug from their program:

1. **What caused the bug?**
   The computer had a reason for doing what it did, and students should understand why their code caused the bug itself. There may be a rule that they can share with others in the class ("There is no closing tag for images") or a misconception ("Sprites all get drawn to the screen when `drawSprites` is called, not when they are updated.") that they can clear up.
2. **How did you find the bug?**
   Have students describe the debugging process that they used, paying special attention to how the type of bug and any error messages lent themselves to particular debugging strategies. Help them to generalize their strategies so that they can use them in a variety of situations (e.g. "I had to capitalize the 'r' in my variable" might become "You double checked the capitalization and spelling of the variable the program didn't recognize.").
3. **What in your code made it easier or harder to debug?**
   Debugging is a great time to reinforce "clean code" practices, such as good naming conventions, use of functions or other ways of "chunking" code into logical sections, commenting, and clear formatting. Point out when comments or well named functions and variables make it easier to trace code and find an error. Debugging is also easier when students have separated out code into logical chunks and functions, which can be commented out individually.

Writing debuggable code

Various practices will make it easier for students to debug their code. Encourage students to neatly format their code, as well as make good use of comments and whitespace. Organizing code into logical chunks, using functions, and having reasonable names for classes, variables, and functions will help them to read and interpret their code as they debug. Point out times when students' good programming practices have made it easier for them to debug their own code.

Guide to Differentiation - Teacher Facing

CS Discoveries Approach to Differentiation

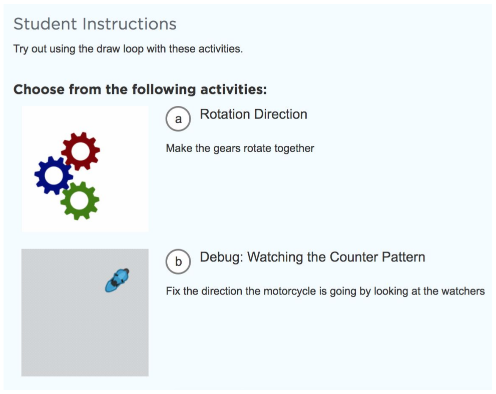
In order to meet the needs of a wide variety of learners, CS Discoveries is designed with a flexibility that allows teachers to differentiate their instruction at the class and student level. Students are expected to take an active part in driving these learning experiences, seeking out appropriate levels of challenge and support with the guidance of a teacher. The course includes multiple features that provide opportunities for students to make choices around how they reach the common learning objectives of the course, while ensuring that the class stays together and maintains a sense of community.

Student Level Differentiation

CS Discoveries lessons and projects are designed to meet these various needs of students within a classroom cohort while keeping them moving at the same pace throughout the course. Within each lesson and project, there are multiple places for students to engage in more practice, get extra support, and challenge themselves with further material.

Practice Levels

Practice levels are found in many programming lessons, after students have been introduced to the target skills, but before the assessment level. Practice levels provide multiple ways to engage with the lesson content, including debugging activities and modifying existing code.



Student Instructions

Try out using the draw loop with these activities.

Choose from the following activities:

a Rotation Direction

Make the gears rotate together

b Debug: Watching the Counter Pattern

Fix the direction the motorcycle is going by looking at the watchers

With the guidance of a teacher, students may choose to complete all the practice options, or just as many as they need to be ready for the assessment. None of the practice options introduce new skills or concepts, so students who skip ahead to the assessment level will not have missed any key material. Previews and short descriptions help students to find an activity that appeals to them.

Challenge Levels

Challenge levels are found after the assessment levels in most programming lessons. These levels include new code and challenges that go beyond the learning objectives of the lesson. Most also include a “Free Play” option that allows students to use the new skills they have learned in whatever way they choose.

Although these levels include new content, students will be able to move on to later lessons without needing anything introduced here, and none of it is necessary to achieve the goals of the course. These levels provide students who have already mastered the lesson content new challenges and extension code, giving them more options to express themselves creatively in the programming environment. As those students take on these more difficult activities, teachers can provide targeted support to students who are still working toward mastery of the lesson objectives.

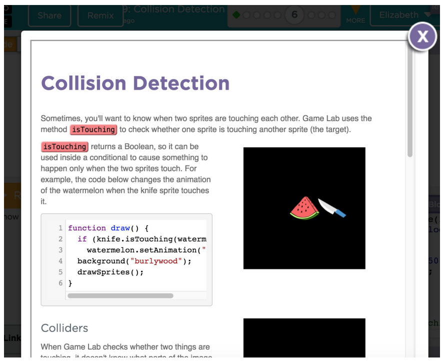
Projects

Open-ended projects are found throughout the programming units, with a major project at the end of each chapter. While each project includes a rubric that assesses key learning objectives, students may choose the content of their projects, and teachers may choose to add or alter requirements for a class or individual students. These requirements could include extra challenges for those who should go further, or they could provide scaffolding for students who need more guidance.

Within each project lesson, students will be asked to plan out their project before starting to code. Teachers may want to look over these plans and ensure that students are choosing a level of difficulty that is appropriate for them. As students iterate on their projects, teachers may want to provide extra challenges or help them to modify their plans to make them more manageable.

Resources

Learning to use resources is a key goal of the course, and given resources provide an opportunity for students to self-differentiate in how they interact with key course content. Programming levels include a “Help and Tips” tab that includes concept explanations, code documentation, and videos to support students in completing their activities.



Collision Detection

Sometimes, you'll want to know when two sprites are touching each other. Game Lab uses the method isTouching to check whether one sprite is touching another sprite (the target).

isTouching returns a Boolean, so it can be used inside a conditional to cause something to happen only when the two sprites touch. For example, the code below changes the animation of the watermelon when the knife sprite touches it.

```
function draw() {
  if (knife.isTouching(waterm
    watermelon.setAnimation("
  background("burlywood");
  drawSprites();
}
```

Colliders

When Game Lab checks whether two things are

As students engage in the programming activities, encourage them to find ways of using the resources that work for them. For example, some students may prefer to look over the documentation before beginning the activities, while others may choose to use it only as a reference when needed.

Class Level Differentiation

CS Discoveries also provide opportunities to tailor the course to meet the needs of the class as a whole.

Implementation Options

Although the course comes with a standard pacing guide, class schedules and student needs will make modifications necessary for most classrooms. Teachers should feel free to spend more time where students need it, even if it means skipping other parts of the course. In particular, the first chapter of every unit ends in a project that will wrap the content up nicely if there is not time for a second chapter.

Alternate Lessons

The first chapter of the first unit, The Problem Solving Process, includes several alternate lessons that cover the same core material with different activities. Teachers may choose a lesson that will work best for their classroom. Multiple options also make it easy to run this introductory unit multiple times, for classrooms in which the course is taught in a modular way, and later modules may include both returning and new students.

Further Support

CS Discoveries Teacher Forum (**https://forum.code.org/c/csd**) is a great place to find advice from other teachers, as well as teacher-created resources to support differentiation. You can also share your own experiences for the benefit of others.

Lesson Plans (**http://curriculum.code.org/csd**) include teaching tips that give guidance on how to differentiate for a class or individual students. They also include a preview of each of the practice and challenge levels, as well as alternate lessons.

Support Articles (**http://support.code.org/**) give detailed information on how to use the various features of the course.

Guide to Resources

Define

What do I need to know?

What is the problem I am trying to solves?

What does success look like?

Prepare

Where can I find answers to this sort of problem?

What has helped me solve similar problems in the past?

What resources are available to me in this programming environment?

Try

Try different types of information:

- Example code
- Explanations
- Videos
- Documentation

Document

What have you learned?

What strategies did you use?

What questions do you have?

Guide to Resources - Teacher Facing

Introduction to Using Resources

Using resources is a core, authentic practice of computer science. Whether novice or expert, all programmers rely on code documentation and other resources when writing programs. In order to build their programming skills, students will need to effectively leverage the resources that they have available to them.

Although students may see their need for resources as a barrier to overcome, the use of resources should be presented as an opportunity to foster a growth mindset and build important skills in using resources. When students have questions, the teacher's role is to help them develop the ability to find out the answers, rather than to directly solve their problems. In modelling and scaffolding the use of resources in the classroom, the teacher demonstrates lifelong learning skills that students can use beyond their current tasks.

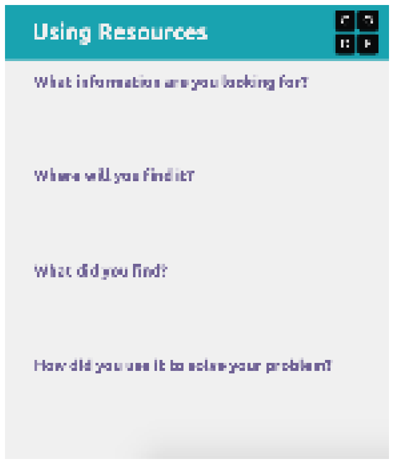
Resources in CS Discoveries

CS Discoveries includes several key resources to support students in the learning process. Video, code documentation, and the help and tips tab are all available to students as they engage in their programming activities. For the most part, these resources are set up to be used as needed, not read as a pre-reading or presented as a lecture. Students are not expected to already know all the needed information when they engage in programming activities. This structure reflects the "discovery" model of the course, but also supports students in the authentic use of resources and code documentation.

Promoting Growth Mindset

It is not reasonable to expect anyone, even an expert, to know everything about a programming language. As programming languages are updated and new languages are introduced, it's important that programmers have the willingness and ability to learn new skills. As resources are used in the classroom, explicitly mention using them as an expert practice, and frame the use of resources as a positive problem-solving strategy. Celebrate when students are able to use resources effectively, even when they feel they should "already know how" to solve the problem themselves. Be open and explicit about using resources when you are unsure or would like clarification of a concept.

Scaffolding the use of resources



Using Resources

What information are you looking for?

Where will you find it?

What did you find?

How did you use it to solve your problem?

Within Code Studio, the use of resources has been scaffolded to help students better access the information that they need. In Web Development, the instructions pane includes several questions that assist students in defining the type of information that they should use resources to access, as well as direct answers to those questions. In all programming units, students have access to a Help and Tips tab, which includes the resources relevant to their current activity.

To support students in using these resources, teachers can introduce a specific framework based on the problem solving process that students use throughout the course. As students have questions, refer them to the framework for using resources and help them to understand what part of the process they are in, and how they can progress. Avoid directly giving them the information, especially before they've engaged in the activity, but support them in understanding where they can find answers and how they can frame their own search.

Using resources as problem solving

In computer science, using resources is a key part of problem solving, and students can use a version of the four step Problem Solving Process as a framework for using resources. Just as in other forms of problem solving, many students may jump to the "Try" part of the framework and begin looking through documentation before really understanding what they are looking for. Remind them that all parts of the process are important, and that ignoring the other three steps will actually take more time in the long run.

Define - Describe what you need to know

Before looking for information, students should first know what they are looking for. Most likely, they are stuck in a programming problem, but they may not understand exactly what sort of information might help them. Students can use the following questions as prompts:

- What do I need to know?
- What is the problem I am trying to solve?
- What does success look like?

In some cases, the answer to this step might be very specific ("I need to know how to make my text green."), but in other cases it might be more general or hard to define. Encourage students to have a question that they want answered before moving to the next step. You may need to support them in articulating their question or understanding what makes a question specific enough to be useful.

Prepare - Decide where to look for the information

Some questions, such as those around how a particular Game Lab block or HTML tag is used, will easily lead to a particular piece of documentation. Others, such as more general questions on how to accomplish a particular task, might not be as obvious. Students can use the following questions to help them decide where they can look for help:

- Where can I find answers to this sort of problem (code documentation, help and tips pages)?
- What has helped me solve similar problems in the past?
- What resources are available for this programming environment?

Try - Read the resources

As students read over the resources, they should compare them to the question that they came up with, and try to relate the information that they are reading to their own problem. They should be actively searching for answers, rather than reading for general comprehension.

As students are reading the resource, they should also be working to use what they have learned. This means that they may read a bit, change their code and test it, then repeat the process depending on the results of their code.

Reflect - Think about what worked

After using the resources, students should think about what was helpful and what was not. Some questions they may ask include:

- Did that answer the question?
- Did the question I asked and information I looked for help me solve my problem?
- What works well for me? Videos? Examples? Step by step instructions?
- What will I do if I have a similar question in the future?

Encourage students to think about how their skills at using resources are improving over time. Do they know what information they should look for? Do they know where to find it? Do they know how to use it?