Contacts	
School Information	Central High School 3535 North Cornelia Fresno, California 93722 559-276-0280
School Contact	Robert Perez Principal rperez@centralusd.k12.ca.us 559-276-0280 Ext 50142
Course Contacts:	

A. COVER PAGE - Course ID ?????	
1. Course Title	Computer Science 1
2. Transcript Title/Abbreviation	
3. Transcript Course Code/Number	
4. Seeking Honors Distinction	
5. Subject Area	Computer Science
6. Grade Levels	10-12
7. Unit Value	
8. Previously Approved s this course previously approved by UC?	
9. Is this course, or any separate section of this course, taught in an online learning environment?	
10. Is this course classified as a Career Technical Education course?	
Yes	

Brief Course Description:
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.

Course Purpose:
The course takes a wide lens on computer science by covering topics such as programming, physical computing, HTML/CSS, and data. The course inspires students as they build their own websites, apps, games, and physical computing devices.

Curriculum Topics
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.

Course Outline
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.

Key Assignments
Unit Activities/Projects:
Web Page Design
App Design
Game Design

Instructional Methods and/or Strategies

CS Discoveries Curriculum Guide

Direct Instruction

Concept Attainment
Cooperative Learning

Inquiry

Assessments Including Methods and/or Tools

Curriculum Guide

Computer Science Discoveries

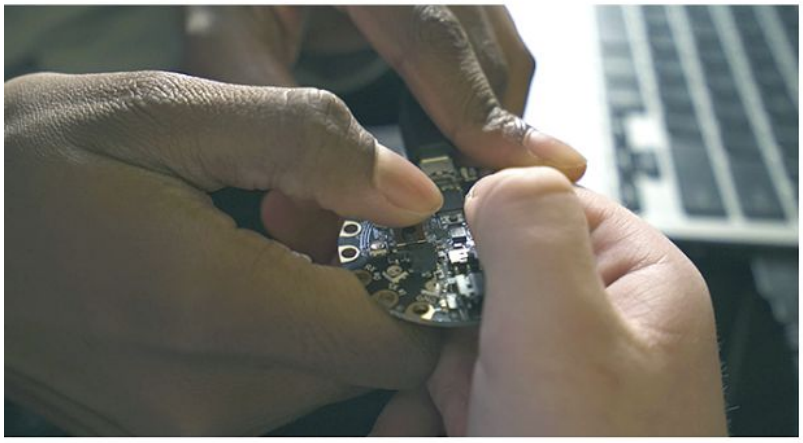


Table of Contents

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 supports than others, and so those with the greatest needs should be prioritized. All students can succeed in computer science when given the right supports 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.

Classroom Practices

The classroom practices for CS Discoveries are different strategies that are used repeatedly in many different lessons and units. These six classroom practices are at the core of the ways the curriculum is designed as we believe these are best practices that lead to positive classroom culture and ultimately student learning.

The 6 Main Classroom Practices of CS Discoveries:

- Lead Learner
- Pair Programming
- Think-Pair-Share
- Authentic Choice
- Unplugged Activities
- Peer Feedback

Lead Learner

What is it?

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

How does it connect to the curriculum?

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

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

How do I use it?

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

Pair Programming

What is it?

Pair programming is a technique in which two programmers work together at one computer. 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 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: teacherblog.code.org/post/147349807334/try-pair-programmingtrack-the-progress-of

Videos (Created for CS Fundamentals, but still applicable:

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

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

Think-Pair-Share

What is it?

Think-Pair-Share is a three part activity where 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 where 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 will 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. Keep your lead learner hat on and direct the conversation without giving away answers or cutting off the conversation too early.

Authentic Choice

What is it?

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

How does it connect to the curriculum?

In the curriculum, we give students many opportunities to work on projects that we hope will feel personally relevant. There is often a focus for the project, but the way it could be implemented leaves room for students to put their personal stamp on the creation. In addition, we encourage teachers to help students adapt projects to their interests when possible.

How do I use it?

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

Unplugged Activities

What is it?

In Code.org curriculum, we refer to activities where students are not on the computer as unplugged lessons. Ones where they are on the computer are called plugged lessons.

How does it connect to the curriculum?

Unplugged activities are more than just an alternative for the days when the computer lab is full. They are intentionally-placed kinesthetic opportunities that help students digest complicated concepts in ways that relate to their own lives. When we write the curriculum, we plan the progression of unplugged and plugged lessons to build on each other. Often something that is done in an unplugged environment is setting the stage for or reviewing a concept done in a plugged environment. Both are vital pieces of the curriculum as they build student knowledge and understanding in different ways. In addition, unplugged lessons can help build and maintain a collaborative environment in your classroom. An unplugged environment also can be a good way to check for student understanding.

How do I use it?

- Don't skip these lessons!
- Try to give the lessons in the order they were written in as best as possible
- Help students make sense of how the activity connects to the concepts they are learning in earlier and later lessons
- Call back to relevant unplugged activities during plugged lessons

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 are 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 who feedback
- 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.

Student Practices

Students in CS Discoveries work in a wide array of contexts, but these experiences are tied together by a core set of practices they develop throughout the course. These student practices provide coherence and serve as helpful reminders of the high-level skills and dispositions they should be continually developing. Throughout 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

- Value and expect mistakes as a natural and productive part of problem solving
- Continue working towards solutions in spite of setbacks
- Iterate and continue to improve partial solutions

Creativity

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

Collaboration

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

Communication

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

Problem Solving Processes

Structured problem solving, through our Problem Solving Process, is a tool for student growth and development which has been woven throughout the entire course. While students learn the generalized Problem Solving Process in Unit 1, we have also developed versions of the process that more clearly articulate what each step looks like in the context of Programming, Design, and Data. The detailed problem solving processes for Programming, Design, and Data can be found in Appendix B.

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6
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

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 like

Prepare

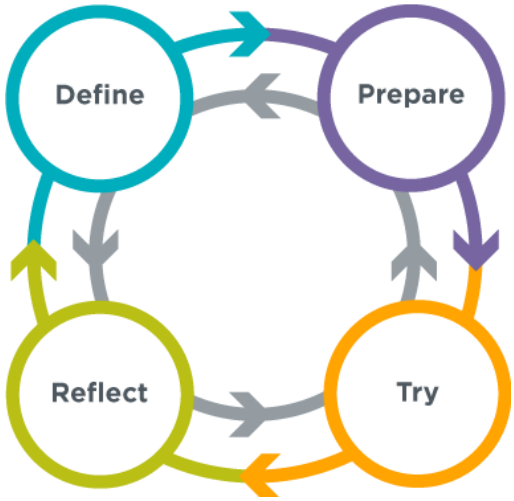
- Brainstorm / research possible solutions
- Compare pros and cons
- Make a plan

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



Course Overview

What is Computer Science Discoveries?

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.

Designed with equity in mind

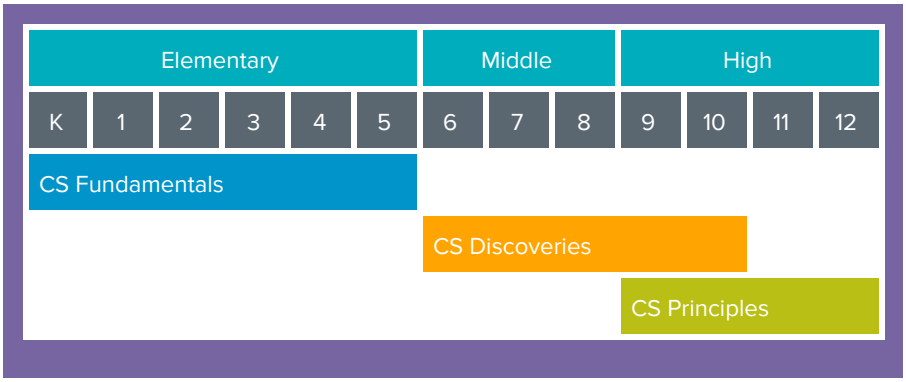
CS Discoveries is designed from the ground up to be an accessible and engaging course for all students, regardless of background or prior experience. It provides students opportunities to engage with culturally and personally relevant topics in a wide variety of contexts and aims to show all students that CS is for them.

Focus on creation

We know that giving students agency in their learning is a powerful tool for creating fun, engaging, and lasting learning experiences. CS Discoveries focuses on the skills that enable students to create and express themselves in a variety of contexts and media. Whether they are developing their own website, designing an app, building a game, or creating a physical computing device, students are empowered to bring their ideas to life.

Completing a K-12 pathway

CS Discoveries is designed to fit naturally between our CS Fundamentals courses and our CS Principles course. While each of these courses is designed to be an age-appropriate entry point to computer science, students with previous experience will find many new topics to explore, and they will revisit familiar topics in novel and more challenging contexts.



Supports for new-to-CS teachers

CS Discoveries is specifically designed to support new-to-CS teachers. The curriculum includes detailed lesson plans and frequent teaching tips. The accompanying forum is an active community that teachers can use to discuss their practice and find additional resources. Teachers can also apply for our professional learning program for further support.

Semester 1 Exploration and Expression	Unit 1 Problem Solving	Unit 2 Web Development	Unit 3 Animations and Games
Semester 2 Innovation and Impact	Unit 4 The Design Process	Unit 5 Data and Society	Unit 6 Physical Computing

Flexible implementation

CS Discoveries is appropriate for 6th to 10th grade classrooms and can be used in either middle or high school. The two semesters build on each other, allowing the course to be taught as a single semester, two sequential semesters, a full-year course, or even integrated into existing technology classes.

Professional Learning Program

Teachers implementing CS Discoveries as a semester or full-year course can apply to participate in a one-year professional learning program. Participation is not required to use any course materials. The professional learning program includes:

- **Summer Workshop:** A 5-day, in-person, workshop held during the summer, designed to introduce core teaching practices and CS concepts from the curriculum (Travel may be required)
- **School Year Workshops:** Follow-up one-day workshops typically held on the weekends, where teachers receive support on upcoming units and have a chance to collaborate with other teachers
- **Online Modules:** Unit-specific teaching supports to use while preparing to teach the lessons
- **Forum:** An active online community where teachers can ask questions, find resources, and collaborate

Powerful tools with a high ceiling

CS Discoveries introduces students to tools and programming languages that are accessible for beginners while offering more advanced students opportunities to create sophisticated projects. Using Code.org's existing App Lab programming environment and our new programming environment, Game Lab, students will be able to transition from blocks to typed code at their own pace while learning JavaScript. Students will also develop maker skills through the study of physical computing at a very accessible cost. 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	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6
	Web Lab	Game Lab	App Lab		Maker Toolkit with Circuit Playground

- **Web Lab** — A browser-based tool for creating and publishing HTML and CSS web sites
- **Game Lab** — A browser-based JavaScript programming environment designed to create sprite-based drawings, animations and games, with the ability to freely switch between programming in blocks or text
- **App Lab** — A browser-based JavaScript programming environment for creating interactive apps, with the ability to freely switch between programming in blocks or text
- **Maker Toolkit** — A collection of commands that extends App Lab's capabilities to allow students to easily program the Circuit Playground and many other physical computing devices directly from App Lab
- **Circuit Playground** — Adafruit's new low-cost Arduino-based microcontroller featuring multiple integrated sensors and output devices

Semester Overviews

Semester 1 - Exploration and Expression

Overview

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	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. Students develop a personal website throughout the unit.
Unit 3: Animations and Games	Students learn many 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

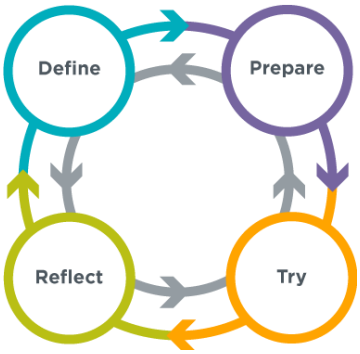
Overview

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 bare 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 prototype one of their own.

Unit 1

Overview



Unit 1 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. The unit concludes with students designing an application that helps solve a problem of their choosing.

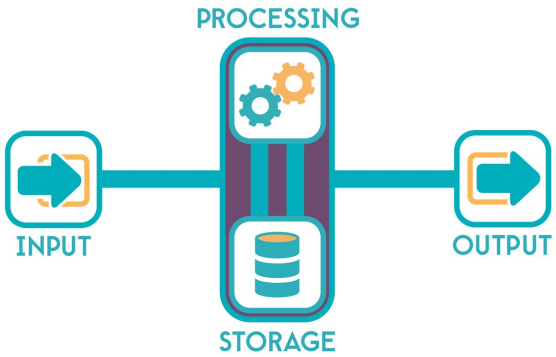
Core Frameworks

- Problem Solving Process
- What is a Computer? - Input, Output, Storage and Processing

Attitudinal Goals

Students will...

- Build a collaborative classroom environment
- Develop positive attitudes toward computer science (relevant, collaborative, fun and empowering)



Big Questions

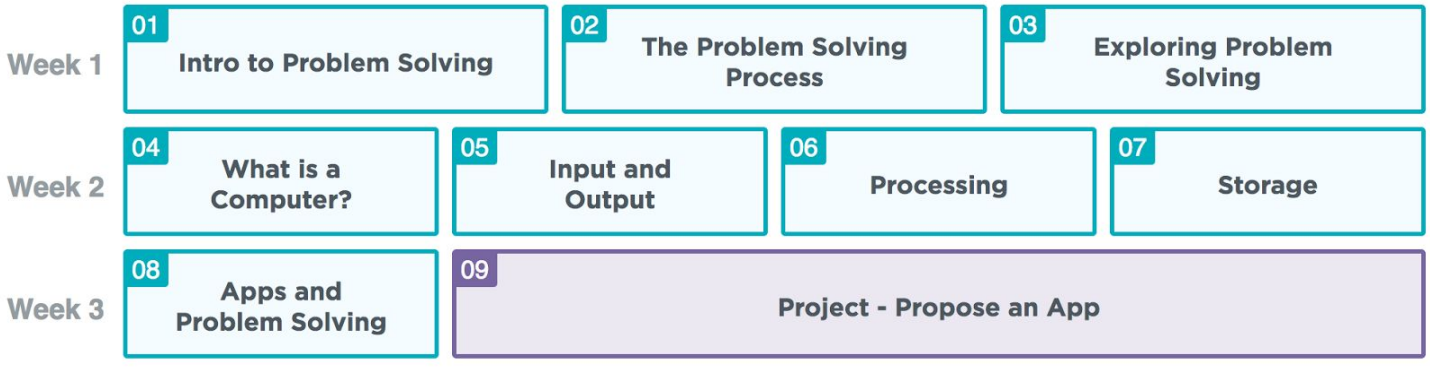
Chapter 1 - The Problem Solving Process

- 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?

Lessons and Pacing



Unit 2

Overview

In Unit 2, students are empowered to create and share the 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 think critically about the impact of sharing information online and how to be more critical content consumers. They are also introduced to problem solving as it relates to programming while they learn valuable skills such as debugging, commenting, and structure of language. At the conclusion of the unit, students will have created a personal website they can publish and share.

Core Frameworks

- Problem Solving Process for Programming
- How to Set Up a Web Page? - Content, Structure, Style
- HTML and CSS
- Digital Citizenship

Attitudinal Goals

Students will...

- View computer science as a form of expression
- See themselves as programmers
- Develop as thoughtful creators and critical online consumers

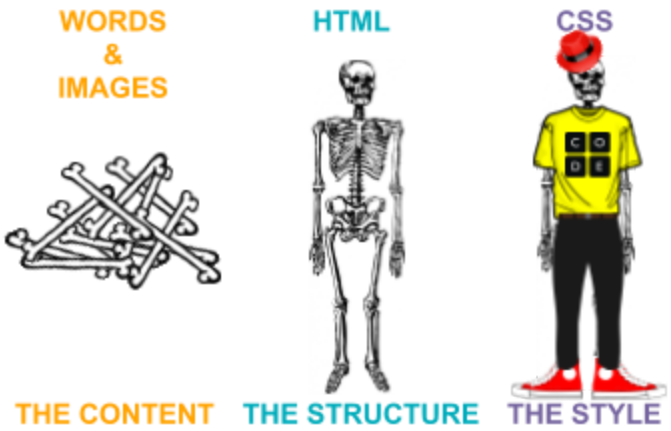
Big Questions

Chapter 1 - Web Content and HTML

- How can text communicate content and structure on a web page?
- Why do people create websites?
- How can I incorporate content I find online into my own webpage?
- What strategies can I use when coding to find and fix issues?

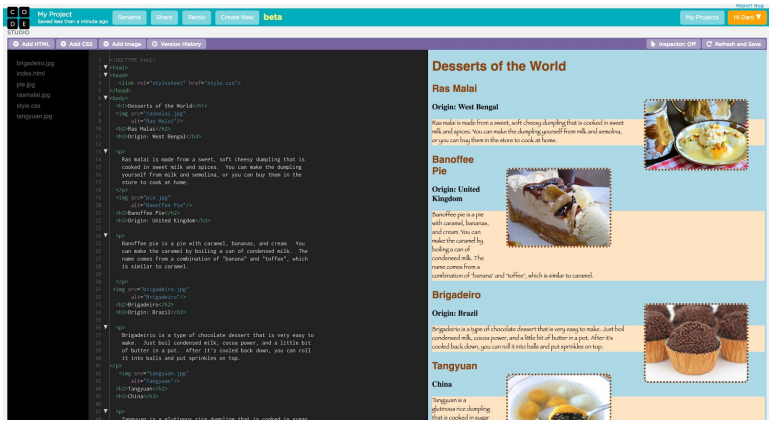
Chapter 2 - Styling and CSS

- How do I modify the appearance and style of my web pages?
- How do I safely and appropriately make use of the content published on the Internet?



Tool - Web Lab

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, as well as 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 studio.code.org/p/weblab.



Lessons and Pacing

Week 1	01 Exploring Websites	02 Websites for Expression	03 Intro to HTML	
Week 2	04 Headers	05 Digital Footprint		
Week 3	06 Lists	07 Intellectual Property	08 Clean Code and Debugging	
Week 4	09 Project - Multi-Page Websites			
Week 5	10 Styling Text with CSS	11 Styling Elements with CSS	12 Sources and Search Engines	13 RGB Colors and Classes
Week 6	14 Project - Personal Portfolio Website			

Unit 3

Overview

In Unit 3, 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.

Core Frameworks

- Problem Solving Process for Programming
- Sequencing and program flow
- Abstraction in programming
- Common programming structures: variables, conditionals, functions
- JavaScript

Attitudinal Goals

Students will...

- View programming as fun, creative, and expressive
- Understand that complex programs are constructed from simple concepts
- Feel that programs can be read and understood

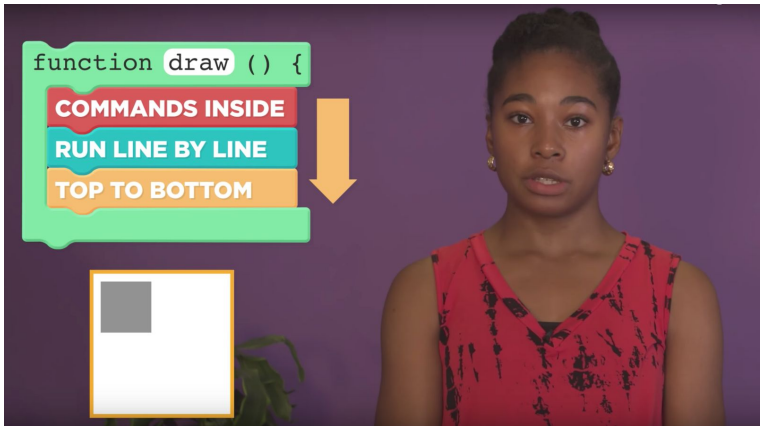
Big Questions

Chapter 1 - Images and Animations

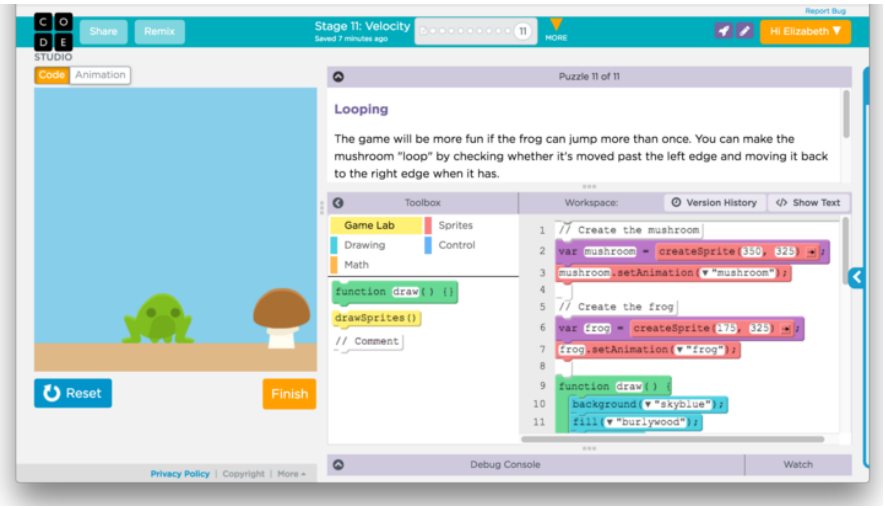
- What is a computer program?
- What are the core features of most programming languages?
- How does programming enable creativity and expression?
- Which practices and strategies will help me as I write programs?

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?



Tool - Game Lab



Description: Game Lab is a programming environment for developing animations and games using JavaScript. Students create 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 allows students to customize the look of their sprites. Using Game Lab, students learn fundamental programming constructs while being given the freedom to create their own virtual worlds. Students can program in either blocks or text and instantly switch between either mode. Game Lab allows curriculum to use a scoped toolbox of commands that focuses attention on the specific blocks or concepts being introduced in that lesson. In addition embedded support tools to help students track down errors in their code. Try the tool at code.org/gamelab.

Lessons and Pacing

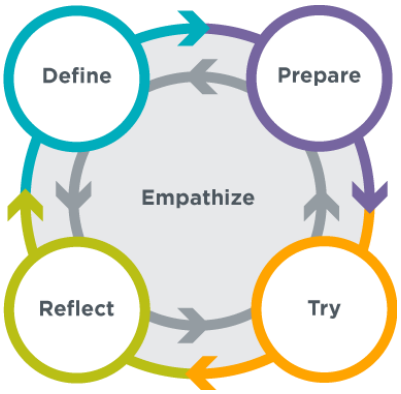
Week 1	01 Programming for Entertainment	02 Plotting Shapes	03 Drawing in Game Lab	04 Parameters and Randomization
Week 2	05 Variables	06 Sprites	07 The Draw Loop	08 Counter Pattern Unplugged
Week 3	09 Sprite Movement	10 Booleans Unplugged	11 Booleans and Conditionals	12 Conditionals and User Input
Week 4	13 Complex Conditionals	14 Project - Interactive Card		
Week 5	15 Velocity	16 Collision Detection	17 Complex Sprite Movement	18 Collisions
Week 6	19 Functions	20 The Game Design Process	21 Using the Game Design Process	
Week 7	22 Project - Design a Game			

Unit 4

Overview

In Unit 4, students transition away from thinking about computer science as a tool to solve their own problems and begin to examine 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.

Core Frameworks



- The Problem Solving Process for Design
- Low Fidelity Prototyping
- User Testing

Attitudinal Goals

Students will...

- View computer science as a tool for social impact
- Value empathy and understanding of a user
- Appreciate the many roles in software development

Big Questions

Chapter 1: User Centered Design

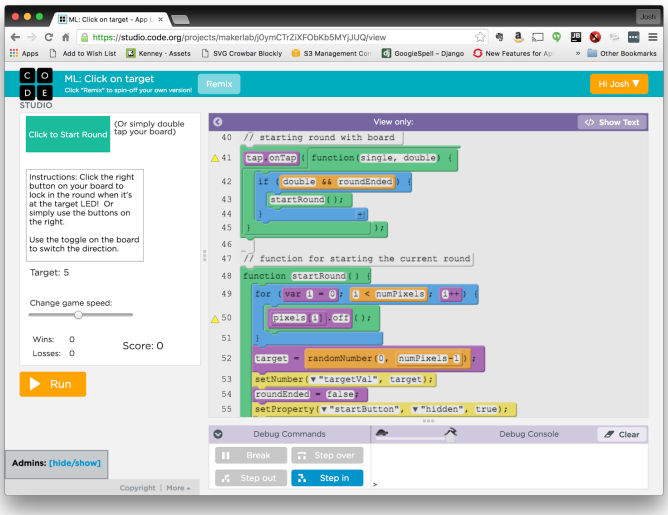
- How do computer scientists identify the needs of their users?
- How can we ensure that a user's needs are met by our designs?
- What processes will best allow us to efficiently create, test, and iterate upon our designs?

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?

Tool - App Lab

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 programmatically define the behavior of these page elements. 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 simple apps.Try the tool at code.org/applab.



Lessons and Pacing

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		
Week 4	10 Paper Prototypes	11 Prototype Testing		
Week 5	12 Digital Design	13 Event Driven Programming	14 Basic App Functionality	
Week 6	15 Testing the App	16 Improving and Iterating		
Week 7	17 Project - App Presentation			

Unit 5

Overview

This unit encourages students to think about the importance of data in solving problems and 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. The unit concludes by asking students to consider how the data problem solving process can be applied to an area of their choosing.

Core Frameworks

- The Problem Solving Process for Data
- Information Representation
- Binary Systems

Attitudinal Goals

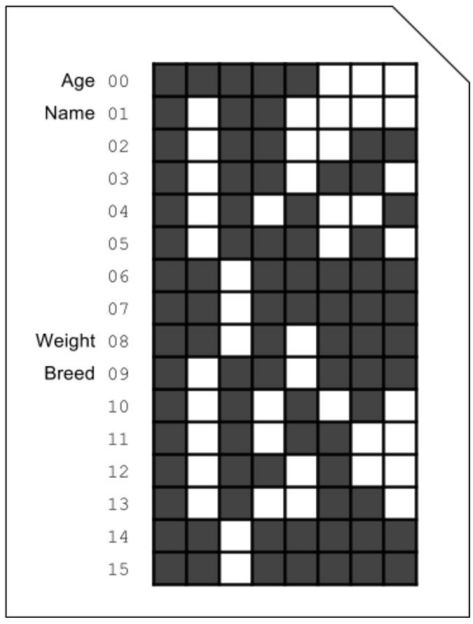
Students will...

- View data as an important tool for solving problems
- Understand the importance of representation systems in computing
- Appreciate the advantages and disadvantages of automated decision making

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 useable binary representation systems?
- How can we combine systems together to get more complex information?



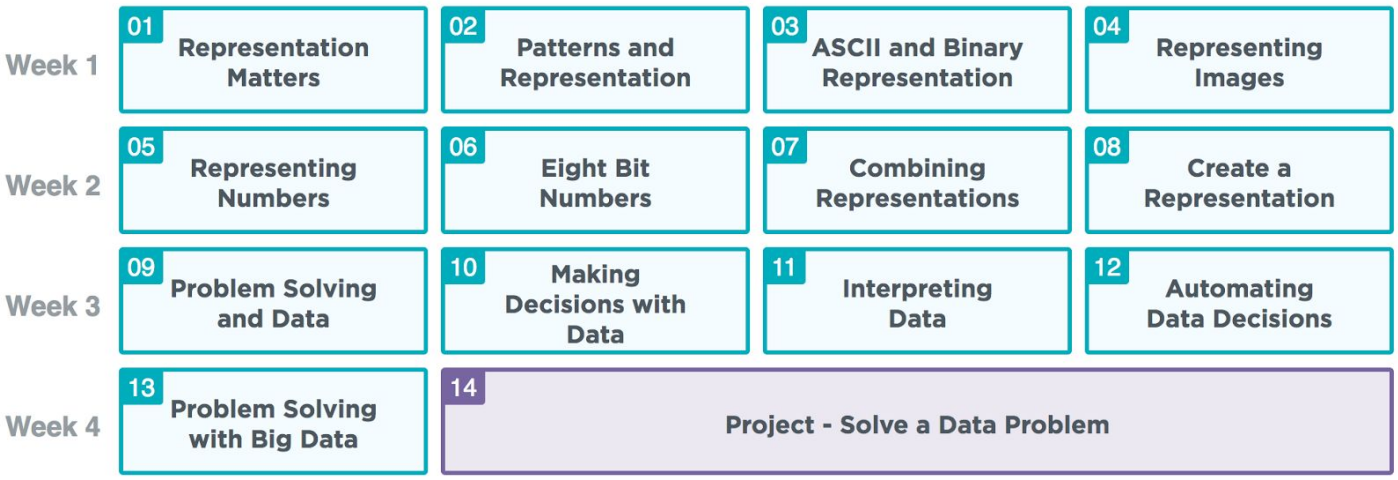
Chapter 2: Solving Data Problems

- 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 problems do computers use data to solve in the real world?

Tool

The majority of this unit features unplugged lessons, but in a few instances students use online tools such as a spreadsheet tool of the teacher's choice, a survey tool of the teacher's choice, and the Code.org Pixelation Widget.

Lessons and Pacing

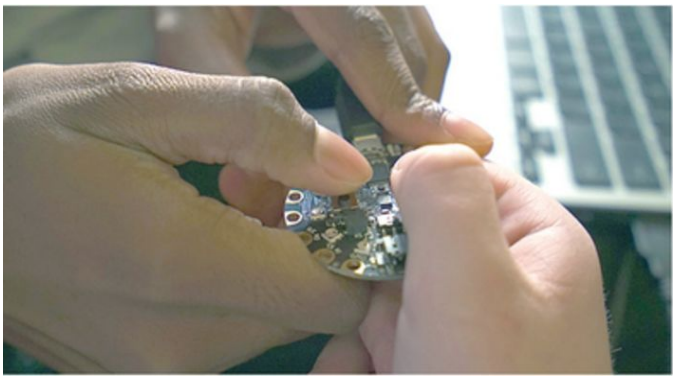


Unit 6

Overview

In Unit 6, students further develop their programming skills while exploring the role of hardware platforms in computing. Harkening back to the Input and Output elements of the Input/Storage/Processing/Output model for computing, students examine current and "smart" devices to understand the ways in which different sensors can provide more effective input and output than the traditional keyboard, mouse, and monitor. Using App Lab and Adafruit's Circuit Playground, students develop programs that utilize the same hardware inputs and outputs that they saw in the smart devices they explored earlier, 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.

Core Frameworks



- Problem Solving Process for Programming
- Event-driven programming
- Hardware input and output
- Physical prototyping
- JavaScript

Attitudinal Goals

Students will...

- View hardware as an approachable and fun topic in computing
- Believe that anyone can contribute to innovation
- Feel equipped to use physical computing to solve problems

Big Questions

Chapter 1: Hardware Output

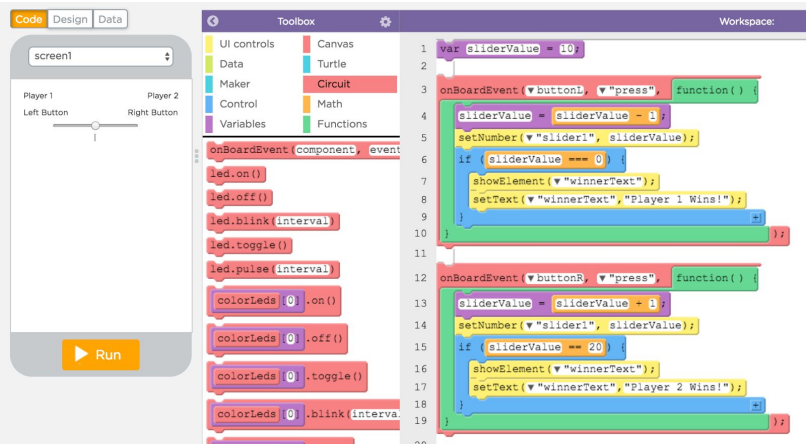
- How does software interact with hardware?
- How can computers communicate information with simple hardware outputs?
- How can programs be made to repeat tasks?

Chapter 2: Hardware Input

- How can computers sense and respond to their environment?
- How can complex real-world information be represented in code?
- How can simple hardware be used to develop innovative new products?

Tool - App Lab with Maker Toolkit

Description: The Maker Toolkit is a set of additional commands available in App Lab that allow you to communicate with a Circuit Playground plugged into USB. 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 makes it simple to quickly prototype apps that combine hardware and software without the overhead of installing additional software or worrying about wiring and electronics. Try the tool at code.org/applab.



Lessons and Pacing

Week 1	01 Innovations in Computing	02 Input Unplugged	03 Event Types
Week 2	04 Getters and Setters	05 The Circuit Playground	06 Lists
Week 3	07 Color LEDs	08 For Loops	09 Lists and For Loops
Week 4	10 Timed Loops	11 Project - Board Output	
Week 5	12 Physical Input	13 Analog Input	14 Sensor Applications
Week 6	15 Project - Prototype an Innovation		

Required Materials and 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 with print in mind and we highly recommend their use.

Beyond printing, some lessons call for typical classroom supplies and manipulatives such as:

- Student journal (See below for more details)
- Poster paper
- Markers, colored pencils, etc.
- 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)

The following items are called for in certain units:

Unit	Materials	Alternatives
Unit 1	Aluminum foil Container for water Pennies	The pennies can be replaced with some other kind of weight of the same size.
Unit 1	Playing cards (1 deck per 8 students)	None
Unit 5	Access to a spreadsheeting tool	None but there are free options
Unit 6	Classroom set of Circuit Playgrounds (1 board and micro USB cable for every 2 students) Check out code.org/circuitplayground for more details.	None - this is the only board currently supported by the curriculum.

Journaling:

The curriculum frequently provides opportunities for student journalling. When students are asked to journal, it is done with the assumption that they will have access their journal writings throughout the course to use 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.

Journaling prompts are frequently used as a reflective wrap up and to record concepts for later reference. You may also choose to use your journals for exit tickets, to support think-pair-share discussions, or in lieu of some activity guides.

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. Because 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.

Planning for Your Year

CS Discoveries is designed to be taught as a single semester, two semesters over multiple years, or a full year course. In order to accommodate the inevitable scheduling constraints you may encounter, we've also broken each unit into two chapters, allowing a clean break point roughly halfway through. The following pacing guide gives a rough recommendation for unit length, assuming that your class meets five days a week for at least 45 minutes per session.

Unit 1 (3-4 weeks)		**Unit 2 (6-7 weeks)**		**Unit 3 (7-8 weeks)**	
Chapter 1 **1 week**	Chapter 2 **2 weeks**	Chapter 1 **4 weeks**	Chapter 2 **2 weeks**	Chapter 1 **4 weeks**	Chapter 2 **3 weeks**
Unit 4 (7-8 weeks)		**Unit 5 (4-5 weeks)**		**Unit 6 (6-7 weeks)**	
Chapter 1 **2 weeks**	Chapter 2 **5 weeks**	Chapter 1 **2 weeks**	Chapter 2 **2 weeks**	Chapter 1 **4 weeks**	Chapter 2 **2 weeks**

Option A: Full Year

If teaching the course as a full year, we recommend teaching all six units in the order presented.

Option B: Separate Semesters

If teaching the course as two separate semesters, and/or spread out over multiple grade levels, we recommend that you ensure that students complete the first semester *before* the second.

Option C: Single Semester

If you only intend to teach a single semester fo CS Discoveries, we recommend that you teach units 1, 2, and 3.

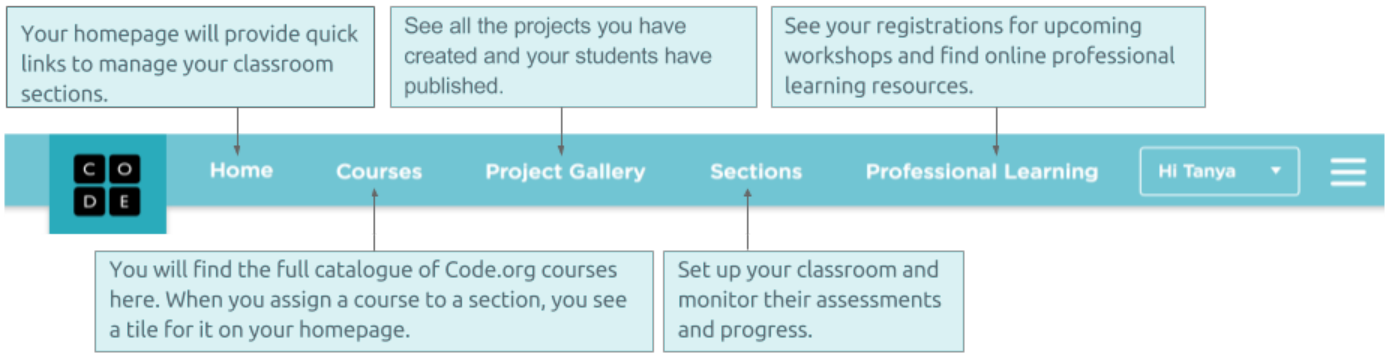
Making Adjustments

The most important thing to consider when making adjustments to the course is that the course was designed to go in order. There are dependencies between units, which may not be obvious to someone who has not taught the course before. Skipping over units is not recommended. In addition, something to consider is that each chapter is designed to culminate in an assessable project. If running up against a deadline, consider planning to stop at a chapter break. You may want to stop at the chapter break in the middle of a unit or to spend some extra time on an end of chapter project instead of just scratching the surface of the following unit.

Navigating the Code.org Website

Code.org Site Navigation

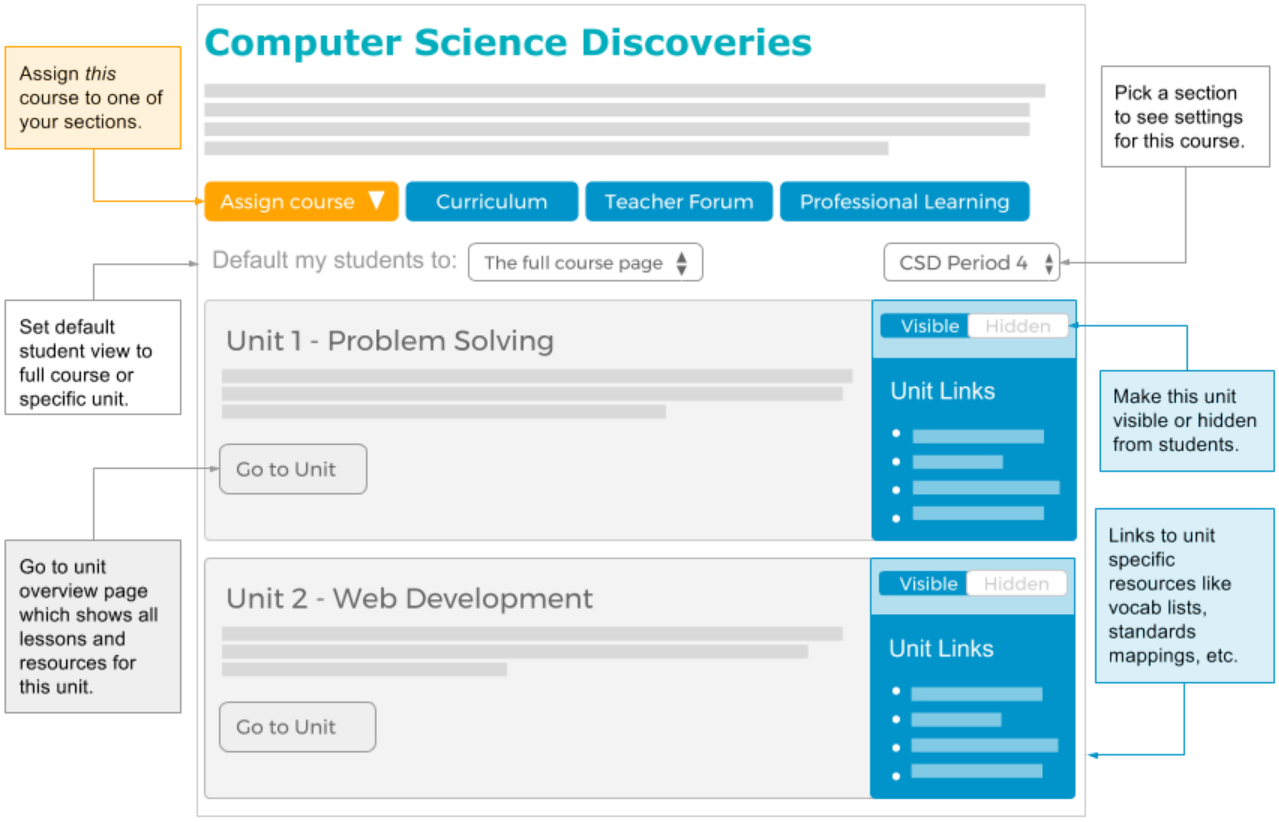
Log into Code.org with your teacher account. The website header will help you navigate the site:



Once you've assigned CS Discoveries to your section, click the tile on your homepage to reach the course overview page. This is your starting point for lesson planning, professional learning, and all the resources you need to teach the course.

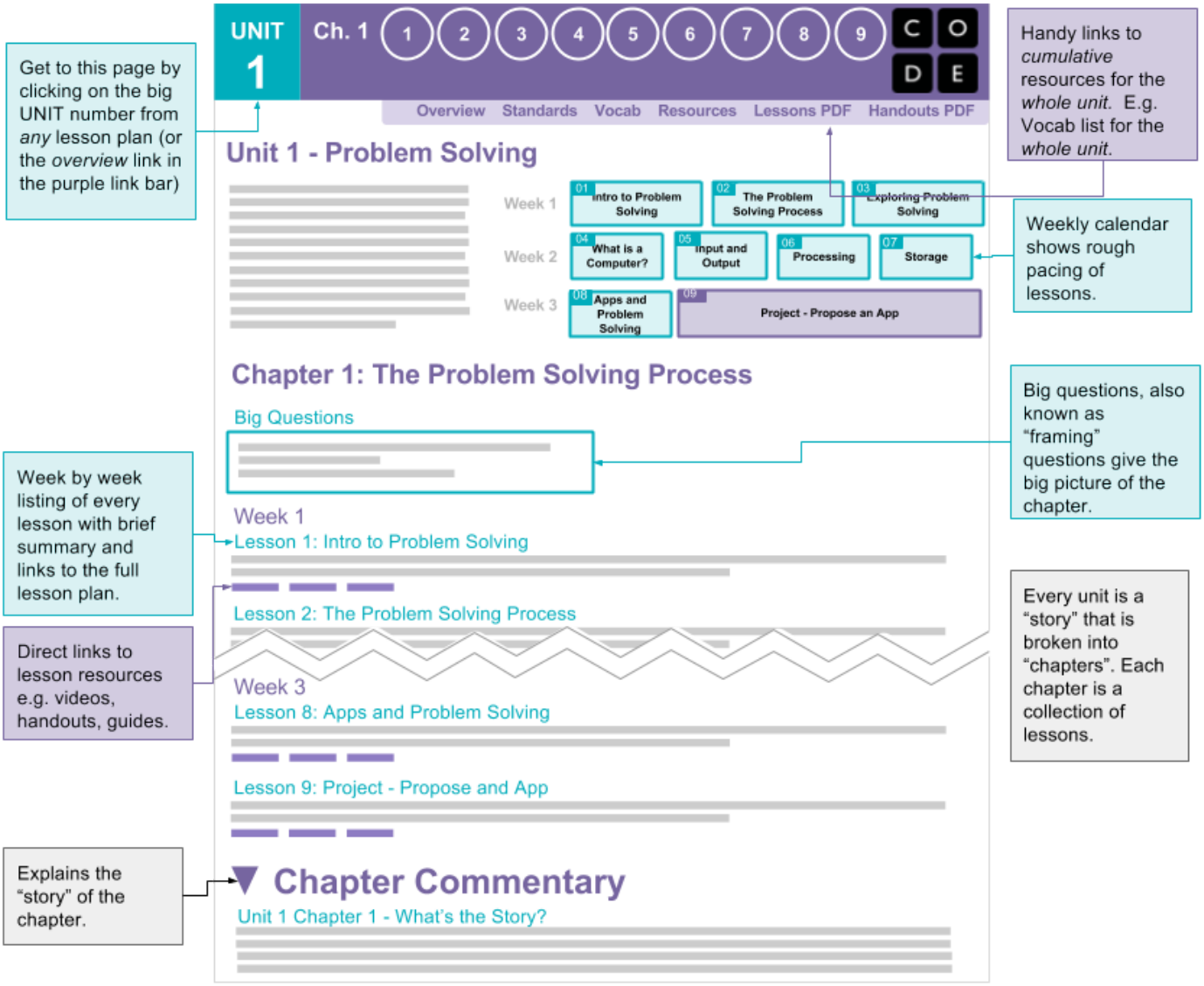
The Course Overview Page

The course overview page will provide quick links to go to the following:



The Curriculum Unit Overview Page

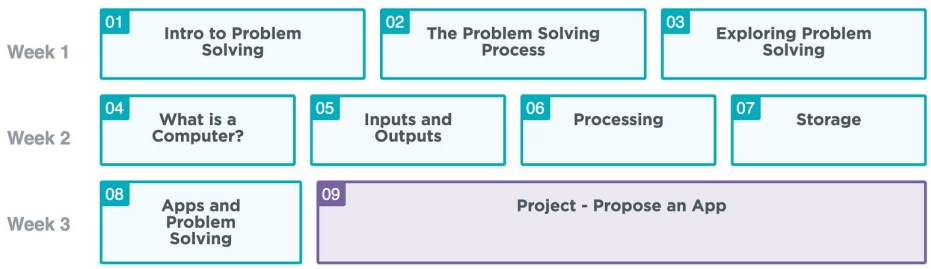
High-level planning should start by looking the unit overview page on curriculum.code.org.



Pacing Calendar

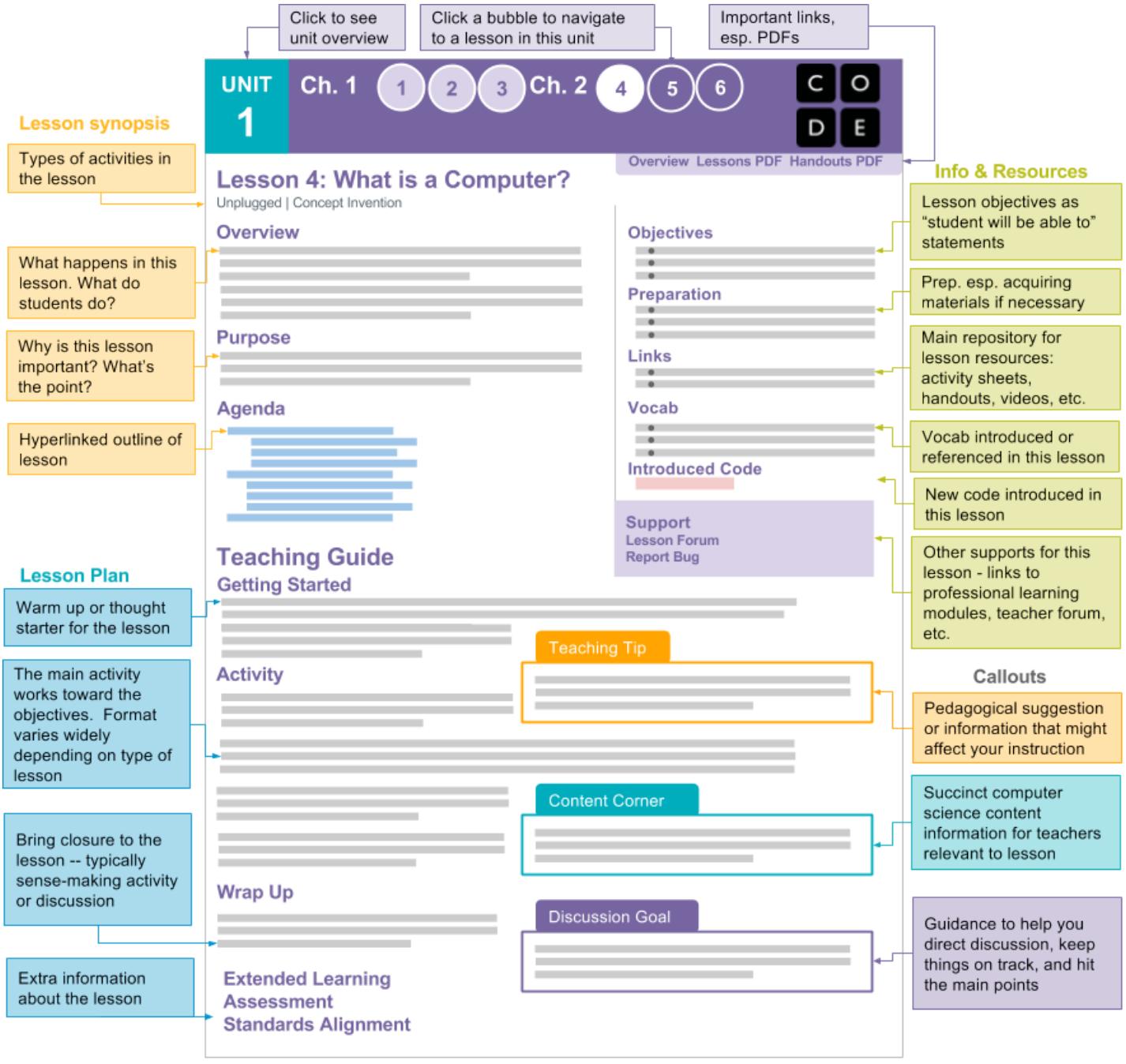
The calendar on the unit overview page shows the relative "size" of each lesson and suggests what you might be able to get through in a week. Larger projects are marked in purple.

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



The Code.org Lesson Plan Structure

Every lesson plan has a common structure that should make it easy to find what you need. Planning for a lesson starts by looking at the overview, then reviewing the core activity to get a deeper sense of what it is and how long it might take.



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 vary based on the age of your students, their background with the material, and their interests.

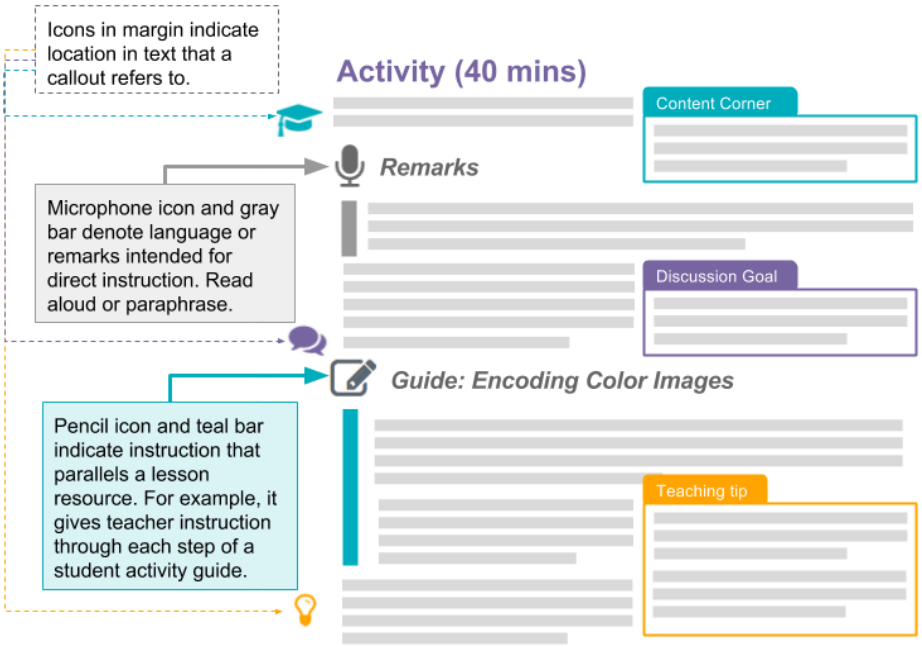
More Lesson Plan Features

Iconography

Within lesson plans you'll notice a number of icons and other kinds of callouts.

These are intended to give context about what "mode" you should be operating in for each part of the lesson.

Sometimes you speak directly to the students, and other times you need to understand the goal of a discussion or give guidance during an activity.

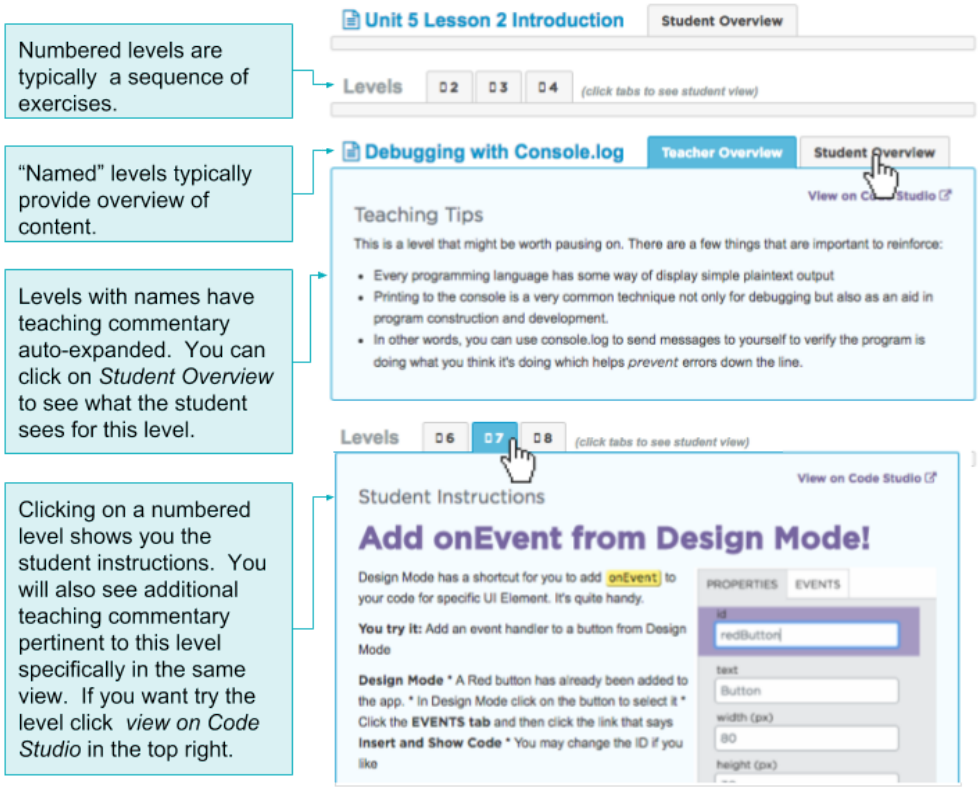


Interactive Code Studio View

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

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

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



Level Types

Informational Levels

Resource Levels
These levels contain instructions, text, or images to help 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.

Video Levels
Video levels contain a video to be used in the curriculum. Videos typically are hosted in multiple formats, including a downloadable file, to increase the likelihood of being compatible in a variety of classrooms.

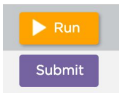
Map Levels
These levels explain a single concept in detail using text, diagrams, images, and external links. Like a map, students can refer to these levels for guidance as they make their way through a lesson. After the fact they can be a handy place to go for a quick review.

Active Levels

Programming Levels
These levels include a Code.org programming environment like Web Lab, Game Lab, or App Lab. An instructions panel explains any new content introduced in the level and provides a checklist of tasks to complete. Starter code may be provided. Teachers can review their students' code from the Teacher Panel.

Quick Check Levels
These levels feature a single multiple choice, free response, or matching question. Teachers can view student responses from the Teacher Panel.

Project Levels
These 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 (shown below) informs students when they are working in a project level.



Submittable Levels
These levels include a "Submit" button that allows students to submit their work to the teacher. Submission creates a timestamp and locks edits. Teachers can view student work and submission times from the Teacher Panel or Teacher Dashboard. Teachers may return a project to students for further edits. Students may also re-submit a project, generating a new timestamp.

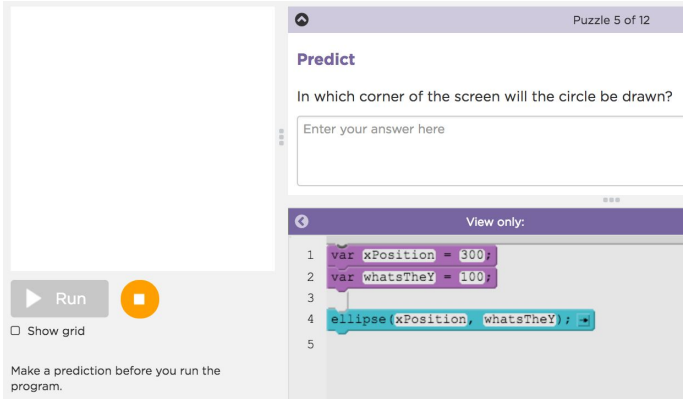
Prediction Levels
These levels ask students to make a multiple choice or free-response prediction about the output of a program. Students are prevented from running code (indicated by a gray "Run" button) until they lock in a prediction. Teachers can view student predictions from the Teacher Panel.

Assessment and Feedback

Frequent assessment and feedback are critical to ensure that students are actively involved in their own learning and teachers have evidence that the class is making appropriate progress. Whether it's for adjusting the pace of the class or measuring student growth and progress, CS Discoveries has incorporated opportunities for assessment and feedback throughout the course. However, since schools have diverse grading systems, it is left up to the teacher to decide how to use the assessment resources for grading purposes.

Checks for Understanding

Throughout the lessons, teachers have many ways to to quickly gauge whether students have grasped a particular concept. Almost all activities that students engage in allow for a teacher to check for understanding.

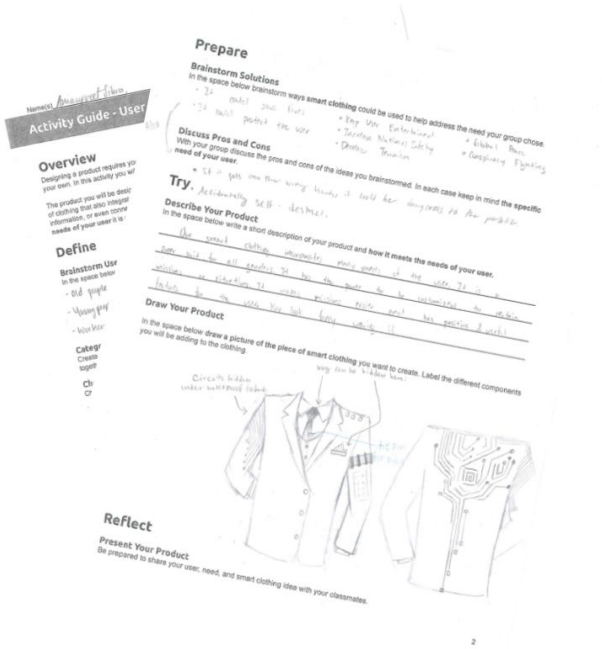


Prediction levels ask students to look at a piece of code and predict what it will do. They are often given before content is presented in order to scope students' exploration during the learning process. Teachers can monitor students' predictions and compare to their later work in the lesson.

Skill-building levels challenge students to complete a small programming task. Teachers have access to all student work in these levels and can read and run the code that students have produced. Puzzles often include exemplar solutions for teachers to reference.

Class discussions provide an opportunity for group sense making. These discussions may begin with students writing down their individual thoughts before sharing with a partner or group.

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.



Opportunities for Feedback

At the end of each lesson, teachers have at least one opportunity to assess and give feedback on what students have learned. Students may submit this work online through Code Studio or on paper.

Submittable levels are Code Studio levels that can be submitted to a teacher for feedback. Most of these levels come at the end of a lesson and involve a creation task. Teachers can use these tasks to assess whether students have mastered the objectives of the lesson and give students feedback on their progress.

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.

Journal Questions allow students to reflect on what they have learned, and what they hope to learn more about.

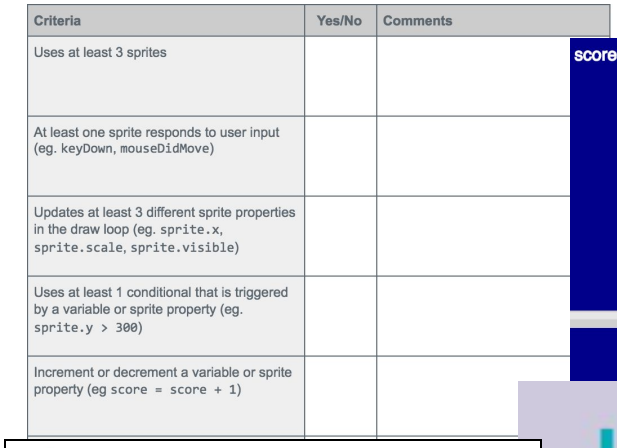
End of Chapter Projects

Each chapter includes a project that incorporates the skills and understandings students have developed. These projects are designed to assess unit-specific skills and the five student practices that thread through the entire course.

Student-facing rubrics give guidance on the skills they must demonstrate, while allowing for plenty of choice in how to demonstrate what they have learned.

Flexible implementation allows teachers to incorporate different modes of presentation, such as posters and oral presentation.

Peer feedback forms prompt students to reflect explicitly on the goals of the project, as well as practice effective communication skills and iterative problem solving.



Criteria	Yes/No	Comments
Uses at least 3 sprites		
At least one sprite responds to user input (eg. keyDown, mouseDidMove)		
Updates at least 3 different sprite properties in the draw loop (eg. `sprite.x`, `sprite.scale`, `sprite.visible`)		
Uses at least 1 conditional that is triggered by a variable or sprite property (eg. `sprite.y > 300`)		
Increment or decrement a variable or sprite property (eg `score = score + 1`)		

Standards Mapping

CS Discoveries was written using both the K-12 Framework for Computer Science and the draft CSTA standards as guidance. Because the revised CSTA standards have not been finalized as of May 2017, we are waiting to publish formal standards mapping documents for CS Discoveries to ensure that we have an opportunity to address any changes that may appear in the final draft. Once the CSTA standards have been finalized and published, we will complete a full pass to articulate, on a unit and lesson level, our mapping to not only the the CSTA standards, but also to select ISTE, Common Core Math, Common Core ELA, and NGSS standards. Once this mapping has been completed, it will be available in the lesson plans and at code.org/csd/standards.

Appendix A: Planning Handouts

Building Your Recruitment Plan

Use this document to map out a recruitment plan for your school, looking at the ways different members of your school team can help recruit and support students.

Course Description. *In your own words, how would you describe this course to a student?*
Strategies for Recruiting. *How do you plan to recruit students to your course?*
Getting Help Recruiting. *How can the other members of your school team help to recruit for this course?*

School Counselors	School Administrators

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

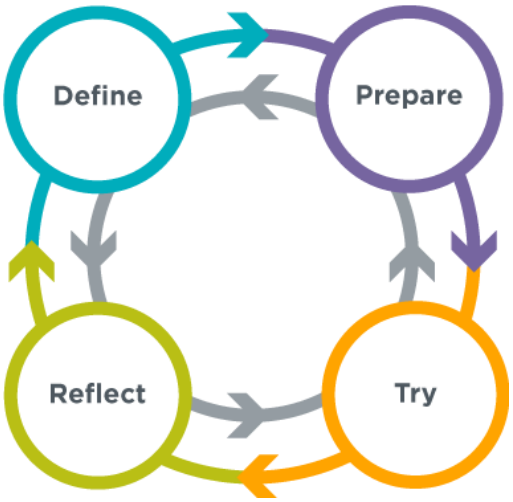
Use the space below to document your pacing plan for moving through each of the instructional units and performance tasks.

What	*Duration*	When do you plan to start?	When do you plan to finish?	Notes or special considerations
Unit 1	*3 weeks*			
Unit 2	*6 weeks*			
Unit 3	*7 weeks*			
Unit 4	*7 weeks*			
Unit 5	*4 weeks*			
Unit 6	*6 weeks*			

Appendix B: The Problem Solving Processes

The Problem Solving Process

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 are you trying to solve
- Identify your constraints
- Describe what success will look like

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

The Problem Solving Process for Programming

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 English 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 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

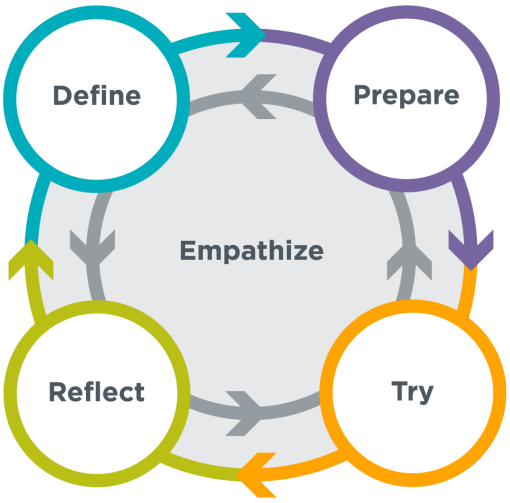
Reflect

- 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 that your documentation is clear and accurate
- Try to "break" your program to find types of interactions or input that you could handle better
- Identify a few incremental changes that you could make in the next iteration

The Problem Solving Process for Design

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 need 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

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 it needs
- 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 change 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 your data using the plan you created
- Clean your data by removing errors, unexpected values, and inconsistencies
- Visualize the data by creating tables, graphs, or charts that help you see broad trends in your data
- Interpret the trends and patterns in your 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'll need to go back to one of the previous steps