

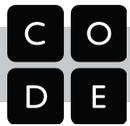
Curriculum Guide

Computer Science Principles



Table of Contents

Code.org Values and Philosophy	3
Pedagogical Approach to our Values	4
Teacher as Lead Learner	5
Code.org AP® Computer Science Principles Curriculum	6
Curriculum Overview and Goals	6
AP Endorsement	6
CS Principles Course At-A-Glance	7
Unit Overviews	8
Unit 1 - The Internet	8
Unit 2 - Digital Information	12
Unit 3 - Introduction to Programming	16
Unit 4 - Big Data and Privacy	18
Unit 5 - Building Apps	20
Performance Tasks	24
Materials List & Tech Requirements	25
AP® Endorsement and Alignment	26
The AP Course Audit Process	26
Planning for the AP Exam and Performance Tasks	26
Planning and Pacing	28
Navigating the Code.org Website	28
The Course Overview Page	28
How to use the Curriculum Unit Overview Page	29
Understanding the Code.org Lesson Plan Structure	30
Other Lesson Plan Features	31
Planning for the year	32
Assessments	36
Tools and How-To guides	38
Widgets	39
Pixelation Widget	40
Internet Simulator	42
App Lab	44
Appendix A: Planning Handouts	45
Building your Recruitment Plan	46
Build your Action Plan: Getting to the Fall	47
Pacing and Planning: Instructional Units	48
Pacing and Planning: AP Exam and Performance Tasks	49
Appendix B: 2-D View of the Framework	50
Appendix C: Course Syllabus	52



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 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 shape meaning from what they have experienced. We also expect teachers to act as the curator of materials. Finally, we include an abundance of materials and teaching strategies - 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, and focus on foundational concepts that allow them to stand the test of time. They are designed to support exploration and discovery by those without computer science knowledge, so that students can develop an understanding of these concepts through "play" and experimentation. From our coding environments to other 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 usually 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.

Code.org AP[®] Computer Science Principles Curriculum

Code.org's Computer Science Principles (CSP) curriculum is a full-year, rigorous, entry-level course that introduces high school students to the foundations of modern computing. The course covers a broad range of foundational topics such as programming, algorithms, the Internet, big data, digital privacy and security, and the societal impacts of computing.

Curriculum Overview and Goals

Computing affects almost all aspects of modern life and all students deserve an education that prepares them to pursue the wide array of intellectual and career opportunities that computing has made possible. This course seeks to provide foundational knowledge and skills to live and meaningfully participate in our increasingly digital society, economy, and culture.

Unit 1: The Internet	Learn how the multi-layered systems of the Internet function as you collaboratively solve problems and puzzles about encoding and transmitting data, both 'unplugged' and using Code.org's Internet Simulator.
Unit 2: Digital Information	Use a variety of digital tools to look at, generate, clean, and manipulate data to explore the relationship between information and data. Create and use visualizations to identify patterns and trends.
Unit 3: Algorithms and programming	Learn the JavaScript language with turtle programming in Code.org's App Lab coding environment. Learn general principles of algorithms and program design that are applicable to any programming language.
Unit 4: Big Data and Privacy	Research current events around the complex questions related to public policy, law, ethics, and societal impact. Learn the basics of how and why modern encryption works.
Unit 5: Building Apps	Continue learning how to program in the JavaScript language. Use Code.org's App Lab environment to create a series of applications that live on the web. Each app highlights a core concept of programming.
AP Performance Tasks	The AP Performance Tasks are projects that students submit to the College Board as part of the AP assessment. The <i>Explore</i> task is a small research project about a modern innovation. The <i>Create</i> task a programming project.

AP Endorsement

Code.org is recognized by the College Board as an endorsed provider of curriculum and professional development for AP Computer Science Principles. This endorsement affirms that all components of Code.org CSP's offerings are aligned to the AP Curriculum Framework standards and the AP CSP assessment. When you register your AP class using materials from an endorsed provider your course is pre-approved for the College Board's AP Course Audit. AP Endorsement also means that Code.org's Professional Development program satisfies the College Board's guidelines for teacher preparation for the course.



AP is a trademark registered and owned by the College Board.

CS Principles Course At-A-Glance

Unit 1 - The Internet

Ch. 1: Representing and Transmitting Info

wk 1 Personal Innovations
Sending Binary Messages
Sending Messages with the Simulator

2 Number Systems
Binary Numbers
Sending Numbers

3 Encoding and Sending Formatted Text
Unit 1 Chapter 1 Assessment

Ch. 2: Inventing The Internet

The Internet is for Everyone

4 The Need for Addressing
Routers and Redundancy
Packets and Making a Reliable Internet

5 The Need for DNS
HTTP and Abstraction
Practice PT - The Internet and Society
Unit 1 Chapter 2 Assessment

Unit 2 - Digital Information

Ch. 1: Encoding and Compressing Info

wk 1 Bytes and File Sizes
Text Compression
Encoding B&W Images

2 Encoding Color Images
Lossy Compression and File Formats

3 Encode an Experience
Unit 2 Chapter 1 Assessment

Ch. 2 - Manipulating and Visualizing Data

4 Intro to Data
Finding Trends with Visualizations
Check Your Assumptions
Good and Bad Data Visualizations

5 Making Data Visualizations
Discover a Data Story

6 Cleaning Data
Creating Summary Tables
Practice PT - Tell a Data Story
Unit 2 Chapter 2 Assessment

Unit 3 - Intro to Programming

Ch. 1 - Programming Languages & Algorithms

wk 1 The Need For Programming Languages
The Need for Algorithms
Creativity in Algorithms

2 Using Simple Commands
Creating Functions
Functions and Top-Down Design

3 APIs and Function Parameters
Creating Functions with Parameters
Looping and Random Numbers

Unit 3 -Intro to Programming (con'd)

wk 4 Practice PT - Design a Digital Scene
Unit 3 Chapter 1 Assessment

Unit 4 -Big Data and Privacy

Ch. 1: The World of Big Data and Encryption

wk 1 What is Big Data?
Rapid Research - Data Innovations
Identifying People with Data

2 The Cost of Free
Simple Encryption

3 Encryption with Keys and Passwords
Public Key Crypto
Rapid Research - Cybercrime

4 Practice PT - Big Data and Security Dilemmas
Unit 4 Chapter 1 Assessment

Unit 5 -Building Apps

Ch. 1: Event-Driven Programming

wk 1 Buttons and Events
Multi-screen Apps
Building an App: Multi-Screen App

2 Controlling Memory with Variables
Building an App: Clicker Game
Unit 5 Assessment 1
User Input and Strings

3 "If" Statements Unplugged
Boolean Expressions and "If" Statements

4 "if-else-if" and Conditional Logic
Building an App: Color Sleuth
Unit 5 Assessment 2

Ch. 2: Programming with Data Structures

5 While Loops
Loops and Simulations
Introduction to Arrays

6 Building an App: Image Scroller
Unit 5 Assessment 3
Processing Arrays
Functions with Return Values

7 Building an App: Canvas Painter
Unit 5 Assessment 4
Practice PT - Create
Unit 5 Assessment 5 - AP Pseudocode Practice

Performance Tasks

2 hrs AP Tech Setup (1-2 hrs)
Can be completed at any time

10 hrs Review the Explore Task & Make a Plan (2 hrs)
Explore PT (8 class hours)
Can be completed any time after Unit 4

14 hrs Review the Create Task & Make a Plan (2 hrs)
Create PT (12 class hours)
Can be completed any time after Unit 5 Chapter 1

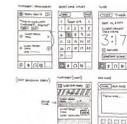
Unit 1 Chapter 1: Representing and Transmitting Information

Big Questions: Why do computers use binary to represent digital information? How does data physically get from one computer to another? Are the ways computers represent and transmit data laws of nature or laws of man?

week Chapter 1 Lessons

Lesson 1: Personal Innovations

Welcome to Computer Science Principles! Groups make a "rapid" prototype of an innovative idea and share it. Students watch a brief video about computing innovations.



Lesson 2: Sending Binary Messages

Students collaborate in an iterative design process to make a "bit sending device" using classroom supplies and everyday objects. They develop their own systems for encoding and sending simple binary messages over some physical distance.



Lesson 3: Sending Binary Messages with the Internet Simulator

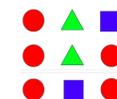
Students use the Internet Simulator for the first time in this lesson. It is configured to represent two parties connected on a single shared wire that only holds one of two possible states. Students invent a binary call-response protocol that can overcome the coordination, timing, and synchronization problems that arise when forced to use such a truly binary system.



Optional Lesson: Sending Bits in the Real World

Lesson 4: Number Systems

Students will explore the properties of number systems by inventing their own number system using only three shapes: a circle, triangle and a square. This lesson is a precursor to looking at several other number systems important to computing, especially binary and hexadecimal.



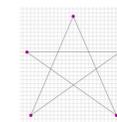
Lesson 5: Binary Numbers

Students become more familiar with the binary number system by making a "flippy do," using the binary odometer widget, and practicing binary-to-decimal number conversions.



Lesson 6: Sending Numbers

Students invent a binary protocol for sending a line drawing represented as list of grid coordinates (numbers). Students test and hone their protocols using a new version of the Internet Simulator which is now configured to automatically send and receive streams of bits.



Optional Lesson: Encoding Numbers in the Real World

Lesson 7: Encoding and Sending Formatted Text

Students invent a protocol that uses ASCII (the standard plain text encoding scheme) to encode formatted text such as fonts, colors, sizes, etc. Students test their protocol using the Internet Simulator, now configured to send and interpret ASCII text, to see if a partner can properly render (draw on paper) the formatting encoded in a message.

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F

Unit 1 Chapter 1 - What's the story?

This chapter is a story about bits - what they can represent and, importantly in the age of the Internet, *how* they physically get from one place to another. Through the lessons in this chapter we look at different types of primitive information (mostly numbers and text) and repeatedly ask a pair of questions: 1) how can I represent *that* with bits? And 2) how do I get those bits from *here* to *there*.

Ultimately we want to give students a sense of the layers of abstraction involved in representing and transmitting information. Sending a single bit becomes sending a stream of bits; a stream of bits can represent number; a number can represent a letter of the alphabet, and so on. In order to communicate using only bits, two parties must agree on **protocols** for encoding and decoding information.

A strong foundation for learning computer science starts with developing a certain level of comfort with abstraction. Computer science is unlike other scientific disciplines in that many of its rules and laws have been defined by people, sometimes arbitrarily, rather than nature. The closest thing to a natural law we can rely on are things related to binary representation, and the associated logic and mathematics that come along with it.

Our Approach to the Content

We want students to go on a bit of a journey in which they repeatedly solve problems related to representing information. A broad term for our approach in this chapter is “**concept invention**” in which we ask students to invent their own solution to a problem before revealing the conventional solution. For example, we ask students to use sequences of geometric shapes - a circle, a triangle, and a square - to invent a number system as precursor to learning about binary.

One advantage of concept invention is that it tends to foster a more **equitable** classroom environment. We try to structure the activities so that they do not advantage prior knowledge of the concept. The key pattern to this approach is to introduce vocabulary and definitions **after** students have had some shared experience with the activity and its associated concepts. A pilot teacher of the course coined the phrase: *Activity Before Concept, Concept Before Vocabulary* or “*ABC / CBV*”.

A seeming disadvantage of concept invention is that students may initially be resistant to learning this way if they are fixated on getting the “right” answer. In an exploratory or discovery-based activity, of course, there is no one right answer. Early on you, yourself, might worry about “chaos” or the fact that students are learning things “wrong.” It is worth being patient and sticking with it because much of computer science (especially programming) is about coming up with your own solutions to problems using the tools you have at hand. There is no one right way to write a program either. Furthermore, this more open approach will have a positive effect on your classroom environment. After a certain amount of trust has been established that you are not setting students up to fail, but rather encouraging them to create and invent, you’ll see a shift in attitudes and **overall student engagement**.

We make heavy use of the **Internet Simulator** in these lessons to enforce rules about binary representation or the activity designed to use it. The configuration of the Internet Simulator changes slightly for each lesson that uses it, but it is always about point-to-point communication: you sending a message to a partner using only bits. The simulator is intended to be a tool for experimentation and play. We encourage you to let students investigate any new features rather than explaining it to them at the outset. Similar to the approach for concept invention, our philosophy is that you can always explain how the tool works **after** students have had some experience (or even frustration) poking around. In fact, they’ll be much more receptive to the explanation after having a hands-on experience.

Unit 1 Chapter 2: Inventing the Internet

Big Questions: Who and what is “in charge” of the Internet and how it functions? How is information transmitted from one computer to the other when they are not directly connected? How can the Internet keep growing? How does that work?

week **Chapter 2 Lessons**

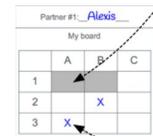
3

Lesson 8: The Internet is for Everyone

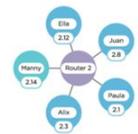
Students read portions of a memo written by Vint Cerf entitled “The Internet is for Everyone,” a call-to-arms that highlights the benefits of a free and open Internet, and identifies threats to this system.

**Lesson 9: The Need for Addressing**

Students invent a protocol (similar to the real Internet Protocol IP) to encode the necessary elements for playing a simplified, but multi-person, game of “Battleship”. Students first play the game ‘unplugged¹,’ then use a new version of the Internet Simulator configured to allow a user to broadcast messages to a group.

**Lesson 10: Routers and Redundancy**

Students use a new version of the Internet Simulator configured with simulated IP addresses and routers to explore the benefits (and potential security concerns) associated with routing traffic across the Internet. Students should see that messages go through many different routers, that messages may not always take the same path to reach a destination, and that all the traffic is publicly viewable!



4

Lesson 11: Packets and Making a Reliable Internet

Through an unplugged activity, students are introduced to packets and issues with packets being delayed and dropped. Students invent a protocol to reliably send a message over an unreliable network using the Internet Simulator, which is now configured to be “unreliable” by delaying and randomly dropping packets sent between routers.



Optional Lessons: Minimum Spanning Tree, Shortest Path, How Routers Learn

5

Lesson 12: The Need for DNS

Through an unplugged activity, students see the difficulties in trying to maintain a universal name-to-IP address mapping system. The Domain Name System (DNS) is introduced and students can experiment with a simplified version using the Internet Simulator.

**Lesson 13: HTTP and Abstraction on the Internet**

Students learn about and investigate HTTP by looking at HTTP traffic generated within their own browser. Students visit a variety of websites and use the browser’s built-in tools to view all the traffic.

**Lesson 14: Practice PT - The Internet and Society**

In this Practice Performance Task, students research and prepare a brief presentation about the controversial issues around either Net Neutrality or Internet Censorship. The presentation and artifacts produced should exhibit students’ knowledge of the Internet.



¹ “Unplugged” activities are done off the computer, and typically consist of physical, kinesthetic activities designed to address complicated concepts in a simplified way.

Unit 1 Chapter 2 - What's the story?

The story of this chapter is in many ways the story of the Internet itself: how it came into existence, and how its protocols were designed to build on each other and allow for its rapid expansion and scaling. Instead of inventing protocols to send information directly from one peer to another, we now consider the protocols necessary for any one computer to communicate with *any* other computer, assuming they are all connected via multiple interconnected networks - i.e. over the Internet.

The lessons in this chapter build on each other just like the real internet protocols do. For example, in a networked world, for any one computer to communicate with any other you need a protocol for computer addresses (**IP addresses**). Once you have addresses, you can consider how data should be **routed** over multiple networks to find the address it's supposed to. Once you have that, you can consider breaking large messages into pieces (**TCP packets**) for more reliable transfer. You also need a way to translate text addresses (like code.org) into IP addresses (**DNS**). Finally, you need a way to communicate high-level text information from one place to another (**HTTP**) that relies on all of these other pieces working as designed.

Again, we want to give students a sense of the **layers of abstraction** involved the creation of these protocols: each protocol solves one problem and solves it well, allowing a more complex problem to be solved on top of it. This philosophy the “open Internet” has allowed an enormously complex global system like the Internet to not only be built in the first place, but also allows everyone to participate in growing, scaling, and adapting to new problems over time.

Our Approach to the Content

Similar to the last chapter, our general approach to learning is through **concept invention**. Each lesson in the chapter poses one of the fundamental problems that had to be solved for Internet communication to work. Because the concepts are often abstract and hard to see, we typically have a robust **unplugged** activity that pairs with each Internet protocol. The activity is designed to illuminate the problem in a physical, experiential way, so that students have a **common experience** to draw on when working to develop a solution.

The solution to a problem is always the invention of a new protocol, and we usually enforce the rules of that protocol by using the **Internet Simulator**, which is configured differently for each lesson. Each configuration incorporates the solution to the last major problem so students don't have to solve it again, while at the same time it exhibits some new behavior or problem that must be solved. For example, after students have invented an addressing protocol, all subsequent versions of the simulator have a simple IP address built in that is handled automatically by the system, but perhaps it will impose a limit on the size of any message, thereby forcing students to invent a protocol to handle “packeting” of information.

Even though designing protocols and testing them on the Internet Simulator is not programming, per se, it has many of the same skills. It requires students to be precise, to manage a growing complexity of systems, to creatively and iteratively solve problems with imposed constraints, and to “debug” those solutions using the tools at hand. Also there is no one right answer most of the time.

We believe that many of the lessons help to improve students' collaborative problem solving behaviors, and bring out other characteristics of computer science work that we want to encourage. We believe these non-programming, computational problems are fun, engaging, and novel. It is unlikely that many students, even those with some background knowledge in computer science, will have considered problems about devising communication protocols. Our theory, again, is that this helps to foster an **equitable** classroom environment that does not unduly advantage students' prior knowledge. We want every student to have an equal stake in the proceedings, to discuss problems, and work together to test out solutions.

Unit 2 Chapter 1: Encoding and Compressing Complex Information

Big Questions: Are the ways in which digital information is encoded more laws of nature or are they man-made? What kinds of limitations does the binary encoding of information impose on what can be represented inside a computer? How accurately can human experience and perception be captured or reflected in digital information?

week Unit 2 Chapter 1 - Lessons

Lesson 1: Bytes and File Sizes

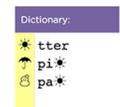
Students are introduced to the standard units for measuring the sizes of digital files: bytes, kilobytes, megabytes, gigabytes, etc., and research the sizes of files they make use of every day.



Lesson 2: Text Compression

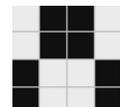
1

Students learn that at some point we reach a physical limit of how fast we can send bits. If we want to send a large amount of information faster, we have to find a way to represent the same information with fewer bits - we must compress the data.



Lesson 3: Encoding B&W Images

Students explore methods for encoding digital images in binary, which requires representing metadata, such as width and height, as well as pixel data. Students use the the Pixelation Widget to encode simple black and white raster images.



Lesson 4: Encoding Color Images

Students learn about the RGB color encoding scheme and use an updated version of the Pixelation Widget to encode color images. Hexadecimal notation is useful for representing larger groupings of binary digits.



2

Lesson 5: Lossy Compression and File Formats

Students research real compression schemes used for images, text, or sound, and determine what kind of compression it uses - lossy or lossless - explaining the theory behind it.



Lesson 6: Practice PT - Encode an Experience

3

Students break down an ambiguous type of information, such as personal experience (attending a party, playing a game, etc.), and invent a way to encode its sub-parts. The project includes a written reflection questions similar to those students will see on the AP Performance Tasks.



Unit 2 Chapter 1 - What's the story?

The story here is about *representing* increasingly complex data and information as an entree to *manipulating* data and information in the next chapter. The lessons are essentially a tour through some of the more interesting forms of digital information representation - specifically, images and text. Encoding images in binary can quickly explode into a number of bits that's hard to keep in one's head all at once. It requires structuring data that includes **metadata**. Compression is the art and science of how to represent the same data with fewer bits, and there are two forms: **lossless** compression, which allows you to reconstruct the exact original bits from the compressed version; and **lossy** compression, common in images and sounds, which throws out information that is likely invisible or inaudible.

The small project that concludes the chapter, *Encode an Experience*, is about the **intersection of abstraction and data**. In a nutshell, students have to think: how can I represent everything here as a series of numbers? The top-down design approach we advocate for is a useful **thinking and problem-solving strategy** for progressively working at finer and finer levels of detail. This approach is about understanding the spectrum of choices that are made when deciding how to **represent information as data**. Since so many different choices can be made, it explains the existence of so many different data formats for similar information that you encounter on a daily basis. For images you see .jpg, .gif, png. For text: .txt, .docx, .pdf, and so on. What are the differences between these things and, more importantly, *why* are there differences? Why can't we just settle on a standard image format or protocol? We explore these reasons through learning experiences that allow students to try their hand at it.

The *Encode an Experience* project has a few underlying purposes: 1) it shows how quickly human decision-making comes into play when figuring out how to represent information; 2) the structure students come up with will look like a tree of relationships between different components of information that make up the whole - this is similar to the layers of data abstraction in database designs, and a lot of publicly-available data is often broken up this way; and 3) the "top down" approach for breaking down information is a precursor to ideas about top-down program design we address in Unit 3 - Algorithms and Programming.

Our Approach to the Content

These lessons will, in many ways, feel a lot like the information representation problems encountered in Unit 1 Chapter 1, and the approach you take should be similar - the only difference is that these lessons are strictly about information representation, rather than being about the Internet. Ultimately the choices made about how to represent information affect how you are able to process or compute with it. We encourage students to "peek" out into the real world as you go through lessons in this chapter to relate the way we encode images and compress to text to the way it's done in the "real world".

This chapter leans heavily on two major widgets that allow students to play with concepts. The **Pixelation Widget** lets students enter binary information and the widget renders an image according to the embedded image format. The black and white version simply encodes images with 1 bit per pixel - 0 is black, 1 is white - while the color version requires students to understand how the **RGB** color scheme works and why hexadecimal representation is so useful for looking at long strings of binary values. In the widget students must also include **metadata** about the image (width, height, amount of color information), which mimics the "real world" uncompressed image encoding scheme known as **bitmap (bmp)**.

The **Text Compression Widget** lets students play with a text encoding/compression scheme that mimics what's known as LZW or ZIP compression. It works by identifying repeated patterns in the original text and storing them in a "dictionary" of patterns for later recall. The challenge is to see how much students can compress an a piece of text - the catch is that there is no way to *actually* know what the "best" is. Compression is a type of computationally hard problem, and the best solution is to experiment, and come up a **heuristic** - a process that is likely to lead to a good-enough solution.

Unit 2 Chapter 2: Manipulating and Visualizing Data

Big Questions: What is the relationship between data, information and knowledge? What are the best ways to find, see, and extract meaningful trends and patterns from raw data? Where and how does human bias affect the collection, processing and interpretation of data?

week **Unit 2 Chapter 2 - Lessons**

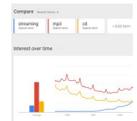
Lesson 7: Introduction to Data

Students examine sources of data in the world around them and how that data is collected. The Class Data Tracker project is introduced, and students predict what they will find after all the data has been collected.



Lesson 8: Finding Trends with Visualizations

Students use the Google Trends tool in order to identify patterns in historical search data. Students present their findings, and must differentiate between explanations of what the data shows versus plausible explanations for discovered patterns.



4 Lesson 9: Check Your Assumptions

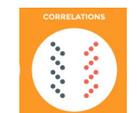
Students examine the assumptions they make when interpreting data and visualizations by first reading a report about the "Digital Divide," which challenges the assumption that data collected online is representative of the population at large. Students also evaluate a series of scenarios in which data-driven decisions are made based on flawed assumptions.

Lesson 10: Good and Bad Data Visualizations

As a precursor to creating their own data visualizations, students examine collections of (mostly bad) data visualizations, rate them and discuss the characteristics of good versus bad visualizations.

Lesson 11: Making Data Visualizations

Students follow a guide to learn how to make scatter, bar, and line charts out of provided data using a spreadsheet tool (such as Google Sheets or MS Excel).



5 Lesson 12: Discover a Data Story

Students collaboratively investigate some datasets (provided) to "discover a data story." Students choose one dataset, create a visualization, identify a trend, and accurately write about it.

Lesson 13: Cleaning Data

Students begin working with the data that they have been collecting for the Class Data Tracker project by first "cleaning" it to prepare it for visualization and other analyses. Each team makes their own copy of the data to examine, correct errors, categorize ambiguous items, and perform other cleaning tasks.

Movie	Genre	Year	Director	Rating
Toy Story	Animation	1995	John Lasseter	9.3
Toy Story 2	Animation	1999	John Lasseter	9.0
Toy Story 3	Animation	2010	Lee Unkrich	8.3
Toy Story 4	Animation	2019	Josh Cooley	8.3

6

Lesson 14: Creating Summary Tables

Students learn how create summary tables, also known as pivot tables, from some raw datasets provided in a spreadsheet tool. Then students create and use summary tables to investigate data they've collected as a class.

Movie	Avg. Rating
Air Force One	3.615384615
Chasing Amy	4
Contact	4.05
Toy Story	3.615384615
Twelve Monkey	4
Willy Wonka	3.818181818

Lesson 15: Practice PT - Tell a Data Story

Students continue to analyze their class tracker project data to discover, visualize, write about, and present a trend or pattern they find. The writing prompts are reflective of prompts from the AP Explore Performance Task.



Unit 2 Chapter 2 - What's the story?

The story of this chapter is about how data can be manipulated to extract or reveal new information. Up to this point we have been focused primarily on bits and what they can be used to represent. Now we're taking a big step back to do the inverse: we want to use tools meant for viewing, manipulating, and visualizing data in order to *extract* or find new information.

The lessons in this chapter often have two things going on at once. In the background, the class is daily collecting some data about themselves (the "Class Data Tracker project") in order to accumulate data to process later on. In the interim, students are learning about and developing skills with spreadsheet and visualization tools. The goal is for students to learn a few basic skills, see lots of examples, and then apply what they know to the *Tell A Data Story* project at the end of the chapter.

A big part of the story here is for students to understand the *computer scientist's role* in working with data, which means emphasizing how to use tools to manipulate, compute, and visualize the data. We look at things like making sure that data type choices support the way we intend to process it later (e.g. don't collect text when you need a number). Data inevitably gets "dirty" during collection and needs to be cleaned. Computers are really useful for doing some aggregations and visualizations to look for patterns. Along the way, we need to understand how human bias can be introduced at each step so that we can accurately convey what patterns in the data are or are not telling us. These activities help build toward the enduring understanding that there are trade offs when representing information as digital data.

Our Approach to the Content

The lessons in this chapter lean heavily on external tools, especially **spreadsheets**. The benefit is that students will gain experience with **real tools and real data** for the first time. The pitfall is that, because the tools are external, they are not scaffolded or designed for learning. We have tried to provide tutorials and curated data sets to ease the burden as much as possible, but ultimately you're operating in the real world. While confined to the world of your classroom, the Class Data Tracker project should provide some **authentic** examples, scenarios, and sometimes headaches related to data collection and processing in the real world.

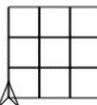
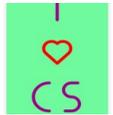
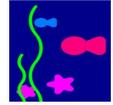
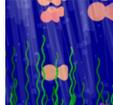
As the teacher it's important to keep in mind the goals of CS Principles because it can be enticing with these lessons to dig into "hardcore" data analysis techniques and statistics. While these are important, they are beyond the scope of CS Principles. Thus, we treat data analysis and statistics a bit like an electric fence: get close, but don't touch. Students should be able to extract interesting things as the result of letting the tools do the work. We provide some large sets of curated data that came from real sources. The data is big enough that you have to apply some computation to make sense of it. We show how to use spreadsheets to do basic aggregations (such as grouping, counting, clustering) and computations (such as average, median, etc.), without turning it into a lesson on statistics and data analysis. We want to build toward the enduring understanding that computing facilitates exploration and the discovery of connections in information.

The idea behind the Class Data Tracker project is that we have found that when students work with data that they collected themselves, especially if the data is *about* the, it is easier and **intrinsically motivating** for students to dig in. To accumulate enough data, we collect it in increments during the time they're building up other skills with data tools. You should connect the skills students are learning in the exercises to similar things they might do with the class tracker data for the *Tell a Data Story* project.

Unit 3 Chapter 1: Programming Languages and Algorithms

Big Questions: Why do we need algorithms? How is designing an algorithm to solve a problem different from other kinds of problem solving? How do you design a solution for a problem so that is programmable? What does it mean to be a “creative” programmer? How do programmers collaborate?

week **Unit 3 Chapter 1 - Lessons**

1	<p>Lesson 1: The Need for Programming Languages Students write instructions for building a small arrangement of LEGO® blocks and trade with a classmate to see if they can follow the commands to construct the same arrangement. The lesson derives the need for a well-defined programming language which leaves no room for interpretation.</p> <p>Lesson 2: The Need for Algorithms Students design algorithms with a "Human Machine Language" designed to perform operations on playing cards. The lesson highlights the connection between programming and algorithms.</p> <p>Lesson 3: Creativity in Algorithms Students continue to work with the “Human Machine Language” to solve more problems and challenges.</p>	  
2	<p>Lesson 4: Using Simple Commands Students use the App Lab programming environment for the first time and become acquainted with the turtle. The chief problem is to find the most “efficient” way to draw an image of a 3x3 grid using a limited set of only 4 commands.</p> <p>Lesson 5: Creating Functions Students learn to define and use their own procedures (or “functions”) in order to create and give a name to a group of commands for easy and repeated use in their code.</p> <p>Lesson 6: Functions and Top-Down Design Students learn about top-down design strategies for solving more complex programming problems by breaking the problem down into small parts that can be named and represented as functions.</p>	  
3	<p>Lesson 7: APIs and Using Functions with Parameters Students read and use App Lab’s API documentation to learn about new turtle commands that they must use to complete a series of drawing puzzles.</p> <p>Lesson 8: Creating Functions with Parameters Students practice using and creating functions with parameters to generalize behavior that can vary, and make use of App Lab’s randomNumber function add variation to the scene.</p> <p>Lesson 9: Looping and Random Numbers Students learn to use a simplified version of a for loop to add repetition to their code</p>	  
4	<p>Lesson 10: Practice PT - Design a Digital Scene Students work in groups of 3 or 4 to design and write the code for a program that draws a digital scene of their choosing. Each member contributes portions of the code that are combined at the end to create the full scene. The project includes written reflection prompts similar to those on the AP Performance Tasks.</p>	

Unit 3 Chapter 1 - What's the story?

The underlying story to this chapter is that programming is a creative endeavor, and a form of personal expression about how to get a machine to solve problems. The first three lessons we use **unplugged** activities to highlight the connections between algorithms, human language, and programming languages. The remainder of the unit we spend **writing programs in App Lab** using the classic “turtle,” or an arrow that you can control with code to create drawings. While students will learn the basic syntax of JavaScript, the main focus is on top-down design and problem solving strategies.

In the unplugged activities we derive the **need for programming languages** first, and then focus on how to express algorithms using them. Algorithms are building blocks for solving computational problems. You can design them to solve individual small tasks and then **combine them and recombine** them to solve others. Furthermore, there is a good deal of creativity involved in designing algorithmic solutions to problems. It may come as a surprise that, given only a few machine instructions and a simple problem, two people may write code to solve it in completely different ways. This might be because their overall approach (algorithm) is different or simply because they went about writing code to express it differently.

When we shift to solving problems with code in App Lab, we hold the context of the problem (solving turtle drawing problems) constant in order to focus on problem-solving behaviors and techniques students can use with JavaScript. In particular we want to **emphasize managing complexity with abstraction**. This entails breaking a problem down into its constituent parts, writing functions to solve each part individually, and then combining them together to solve the overall problem. Students should be very familiar with the general ideas behind “top-down” design from the previous units of study. Just like figuring out a way to encode a complex piece of information in binary, computer programs can be thought of as complex systems that can be broken down repeatedly to gain insight into the subsystems that make it up.

Our approach to the Content

To call programming a creative form of personal expression sounds counterintuitive to a lot of beginners. Our hope is that this fact resonates with new students since we see it as illustrating our teaching philosophy of computer-science-as-a-liberal-art. The mindset is expressed eloquently by Donald Knuth: “Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.” It is with this sentiment in mind that we start the unit by doing just that - writing out algorithms to be executed by hand to gain insights into what makes programming challenging and creative. The unplugged activities reveal why programming languages need to exist, and foreshadow some of the ideas that come later in this unit and throughout the course. They serve as a **common point of reference** that many future lessons can be tied back to.

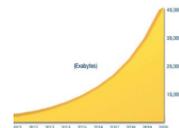
Our sequence of lessons to introduce programming is a little unconventional, especially our choice to place early emphasis on writing functions and procedural abstraction, but there is a method to the madness. A major skill that programmers have is the ability to look at a problem and know how to break it down into smaller and smaller parts that can be more easily programmed into reusable procedures. In JavaScript, procedures are called “functions,” and the generic term for encapsulating the solution to a problem in a named procedure is called “**procedural abstraction**.” We focus on **functions and procedural abstraction** for three main reasons: (1) we want to keep the focus on problem solving and collaboration rather than getting lost in syntax, (2) we are reinforcing the enduring understandings that multiple levels of abstraction are used to write program, and (3) just to be practical, the interactive event-driven apps we make in Unit 5 require a solid foundation in writing functions.

Unit 4 Chapter 1: The World of Big Data and Encryption

Big Questions: What opportunities do large data sets provide for solving problems and creating knowledge? How is cybersecurity impacting the ever-increasing number of Internet users? How does cryptography work?

week **Unit 4 Chapter 1 - Lessons****Lesson 1: What is Big Data?**

Students are introduced to the concept of “big data,” where it comes from, what makes it “big,” and how people use big data to solve problems, and how much of their lives are “datafied” or could be.

**Lesson 2: Rapid Research - Data Innovations**

Students “rapidly research” a topic of personal interest and respond to questions about how that innovation produces, uses, or consumes data.

**Lesson 3: Identifying People With Data**

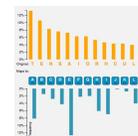
Students investigate some of the world’s biggest data breaches to get a sense for how frequently data breaches happen what kinds of data is lost or stolen.

**Lesson 4: The Cost of Free**

Students examine some of the economic concerns and consumer tradeoffs related to apps and websites that collect and track data about you in exchange for providing you a service free of cost.

**Lesson 5: Simple Encryption**

Students are introduced to encryption and use a widget to attempt cracking Caesar and random substitution ciphers.

**Lesson 6: Encryption with Keys and Passwords**

Students use a widget to experiment with the Vigenère cipher to learn about the relationship between cryptographic keys and passwords.

**Optional Lessons: Traveling Salesperson Problem, The WiFi Hotspot Problem****Lesson 7: Public Key Cryptography**

In this big, multi-step lesson, students learn how the basic mechanics and underlying mathematical principles of public key encryption work. Public key encryption is the basis for most secure transactions on the internet.

**Lesson 8: Rapid Research - Cybercrime**

Students pick a type of cyber attack or cybercrime and do some “rapid research” to learn more about it. The lesson is a precursor to the Practice Performance Task about Big Data and Security.

**Lesson 9: Practice PT - Big Data and Cybersecurity Dilemmas**

Students complete a small research project about a dilemma related to either Big Data or Cybersecurity. The project mimics elements of the Explore Performance Task.



Unit 4 Chapter 1 - What's the story?

The story of this chapter is about coming to terms with the world of Big Data that we now inhabit, and addressing the new modern dilemmas that come along with it. In many ways, this unit acts as a current events unit, since the daily news is filled with examples: should the government get “backdoor keys” to encryption algorithms in order to unlock a cell phone used by a terrorist? Should a social media site be able to use the data it has about you and your relationships to direct advertising at you, or sell information about you to others? At the end of the unit, students are asked to develop an opinion supported by their own research about a dilemma related to either cybersecurity or personal privacy.

There are two main threads in this unit, which are interwoven: (1) Big Data and (2) Encryption/Security. Since it is nearly impossible to talk about big data without delving into the issues related to security and privacy, it's a useful time to learn about encryption and how it works. Encryption can be an engrossing subject in its own right, since it involves interesting algorithms, mathematics, and problem solving, not to mention the aspects of societal impact. Indeed, there are entire undergraduate degrees on the subject - The main goal of our encryption lessons is to work up to understanding how public key encryption works, including the primary mathematical principles that make it possible for two people (or computers) to send encrypted messages to each other over the internet in a way that only the intended recipient can decrypt it.

Ready for the Explore PT? We think that the end of this unit represents a minimum point at which students could complete a successful Explore performance task. Check out the Performance Task pacing section on page 32 for more details.

Our Approach to the Content

Many of the lessons in this unit are designed as **practice for elements of the Explore Performance Task**. In particular, lesson 2 “Rapid Research” is good practice for the relatively quick research and writing students will have to do for the Explore PT. The goal is for students to become adept with looking up sources, reading/skimming articles for their main points, and being able to explain both sides of an argument or dilemma related to big data, security and privacy. Since issues about personal privacy and security affect students' daily lives, the research should be **relevant and engaging** for students.

Most of the activities in these first two weeks call for students to be engaged in online research, to use online tools to investigate issues, as well as to discuss and write about the issues. These lessons in particular are a great entry point for the teacher to assume the role of **lead learner**. The first week in the unit will feel like: *big data is great!* The second week, however, might feel like: *big data is scary!* Especially in the latter case, the teacher might need to attend to their students' runaway paranoia about the harmful effects of big data. We want to present a balanced view, but it is a dilemma! Big data is both great and scary at times. Knowledge and awareness are the best tools to protect yourself.

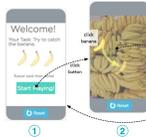
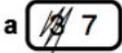
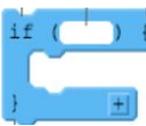
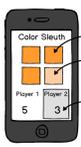
The activities in the third week around data encryption should look and feel similar to lessons from Units 1 and 2. The general pattern is to introduce a concept through an unplugged activity or thinking prompt, and then “plug it in” by using a **widget** to explore the concept further. The purpose of the widgets is to allow students time to play with some of the encryption ideas, which are often mathematical in nature, to get sense for how they work. We want to give students space to be curious and use the tools to **experiment, explore and discover** some essential properties of encryption.

We encourage teachers to use the practice PT to dry-run some of the procedures and processes for the Explore Performance Task. In particular: put time constraints on the research and writing, grant students latitude to research what they want, monitor the appropriateness of students' research choices, and help them get “unstuck,” or push them to be more specific, by appealing to the scoring guidelines and rubric.

Unit 5 Chapter 1: Event-Driven Programming

Big Questions: How do you program apps to respond to user “events”? How do you write programs to make decisions? How do programs keep track of information? How creative is programming? How do people develop, test, and debug programs?

week **Unit 5 Chapter 1 - Lessons**

1	<p>Lesson 1: Introduction to Event-Driven Programming Students are introduced to Design Mode in App Lab, which allows students to easily design the User Interface (UI) of their apps and add simple event handlers to create a simple game.</p> <p>Lesson 2: Multi-Screen Apps Students improve the chaser game by learning how to add multiple “screens” to an app and by adding code to switch between them. Students learn to use console.log to display simple messages for debugging purposes.</p> <p>Lesson 3: Building an App: Multi-Screen App Students design and create a 4-screen app on a topic of their choosing. Students may collaborate with a classmate as a “thought partner,” similar to the recommendation for the Create Performance Task.</p>	  <p>Screen</p> 
2	<p>Lesson 4: Controlling Memory with Variables Students learn to create and assign values to variables and are navigated through common misconceptions.</p> <p>Lesson 5: Building an App: Clicker Game Students learn about global versus local variables, and use variables to track the score in a simple game.</p> <p>Lesson 6: User Input and Strings Students develop a simple Mad Libs® app, learning to collect and process text strings as input from the user.</p>	<p>a </p> <p>b </p>   <p>Text Input</p>
3	<p>Lesson 7: if statements unplugged Students trace simple robot programs on paper to develop a sense of how to read and reason about code with if statements in it. The code is the same pseudocode used on the AP exam.</p> <p>Lesson 8: Boolean Expressions and if Statements Students learn how to write and use if statements in JavaScript by debugging common problems, solving simple problems, or adding conditional logic into an existing app or game.</p>	 
4	<p>Lesson 9: if-else-if and Conditional Logic Students are introduced to the boolean (logic) operators NOT, AND, and OR as well as the if-else-if construct as tools for creating compound boolean conditions in if statements.</p> <p>Lesson 10: Building an App: Color Sleuth Students follow an imaginary conversation between two characters, Alexis and Michael, as they solve problems and make design decisions in the multiple steps required to construct the “Color Sleuth” App. Students must implement elements of the code along the way.</p>	  

Unit 5 Chapter 1 - What's the story?

This chapter establishes the basic story of “What’s an app?” The first week is dedicated to introducing App Lab’s **design mode**, and becoming familiar with the **event-driven mindset** for programming. The largest difference between this unit and previous programming unit (unit 3) is the the **event-driven** paradigm for programming. In Unit 3 (turtle programming) everything was procedural: you click “run” on the program and it starts executing from the first line of code, and runs until completion. An event-driven program never ends! It is constantly waiting to react to user input like clicking a button, or moving your mouse. You write programs by deciding which events you want to respond to and by writing a discrete function to respond to that specific event. As part of its execution that function may run some loops, perform calculations, call other functions, and so on.

Next we cover **variables, user input (including text strings), Boolean expressions and if-statements**. It’s quite a blitz through a gamut of fundamental programming concepts. The story to tell is that these concepts are behind features of apps that you are familiar with. Want to keep score in a game? Variables. Want to respond to something the user types? Text input. Want your app to exhibit different behaviors based on certain conditions? Boolean expressions and if-statements. The **Color Sleuth** game/app is an important culminating project that ties together all the concepts learned in this chapter. It is a unique lesson in which the student follows a conversation between two fictional students collaborating to plan, design and write the code for their project. The student follows along in and writes the code to match the plans of the fictional students.

Ready for the Create PT? We think that the end of this chapter represents a minimum point at which students could complete a successful Create performance task. Check out the Performance Task pacing section on page 32 for more details.

Our Approach to the Content

Our approach to teaching these concepts is somewhat “traditional” in terms of the sequence of concepts and how they build on each other. If you study the lessons you will notice a rough pattern to how we scaffold the learning for each concept which is typically as follows. (1) Introduce the concept in an unplugged or discussion-based way to activate prior knowledge and motivate the students’ need to learn the concept. (2) Learn about and practice the code related to that concept. Students read about and work through a series of exercises, which they can do in pairs or solo, to practice using any new code related to the concept, as well as solving and debugging a few problems. (3) Follow a Building an App lesson, which walks students through the construction of an app from scratch. In the lesson students progressively build parts of it, and submit a final version. These apps allow room for some student creativity and indeed students should be encouraged to “make it their own” while still using the underlying concepts.

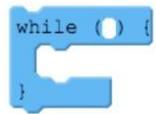
The concepts covered in this chapter, especially variables, conditional logic and if-statements carry a lot of classic misconceptions. Our lessons often try to lead students into those misconceptions by asking them to debug or problem-solve around them. This means that there is a risk for some students becoming frustrated or confused by these lessons. We’ve tried to provide a number of supports and resources you might use to help clear up confusion. One key resource is the video related to each concept. The videos are dense enough that we recommend you use them to sense-make after students have been through a programming experience, as well as before. Another resource to be aware of would be the “maps,” which are static pages that explain the code for a concept and contain diagrams with other helpful information that are meant to serve as a reference and an introduction to the concept in the first place.

Unit 5 Chapter 2: Programming with Data Structures

Big questions: How are real world phenomena modeled and simulated on a computer? How do you write programs to store and retrieve lots of information? What are “data structures” in a program and when do you need them? How are algorithms evaluated for “speed”?

week **Unit 5 Chapter 2 - Lessons****Lesson 11: While Loops**

Students are introduced to the "while loop" construct by first analyzing a flow chart and then by completing a series of exercises in Code Studio. The "while loop" repeats a block of code based on a boolean condition.

**Lesson 12: Loops and Simulations**

Students make a simple computer simulation to model a coin flipping experiment that is possible, but unreasonable, to do by hand. Students write code that uses while loops to repeatedly "flip coins" (random number 0 or 1) until certain conditions are met.

**Lesson 13: Introduction to Arrays**

Students learn about arrays in JavaScript as a means of storing lists of information within a program. Students build a simple app, My Favorite Things, which stores and cycles through a list of words describing their favorite things.

**Lesson 14: Building an App: Image Scroller**

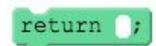
Students extend the My Favorite Things app to manage and display a collection of images instead of words. Students also learn to make the program respond to keys (left and right arrow) by using the "event" parameter that is created when an event is triggered.

**Lesson 15: Processing Arrays**

In this is long lesson, students learn to use for loops to process lists (arrays) of data in a variety of ways to accomplish various tasks like searching for a particular value, or finding the smallest value in a list. Students also reason about linear vs. binary search.

**Lesson 16: Functions with Return Values**

Students learn to write functions that calculate and return values, first through an unplugged activity by playing Go Fish, then by practicing in Code Studio, and finally by writing functions that return values in a simple turtle driver app.

**Lesson 17: Building an App: Canvas Painter**

Canvas Painter is a culminating project brings together processing arrays, functions with return values, and handling keystroke events. The app allows a user to draw an image while recording in an array every single x,y location the mouse passes over on the canvas. By processing this array in different ways, the image can be redrawn in different styles, like random, spray paint, and sketching.

**Lesson 18: Practice PT - Create Your Own App**

Students design an app based off of one they have previously worked on in the programming unit. Students choose the kinds of improvements they wish to make and write responses to reflection questions similar to those they will see on the AP® Create Performance Task.

Unit 5 Chapter 2 - What's the story?

This chapter tells the story of the real power of computers, which is to quickly and precisely perform many of computations on data to produce a result. There are two pieces to this puzzle. (1) We need to better understand how to control **iteration** (loops) beyond a simple repeat loop. (2) We need to be able to store and process lists of data rather than single variables.

Knowledge and facility with loops and lists opens an almost infinite number of doors to different types of programs you can write and problems you can solve. The projects and examples in this chapter merely scratch the surface of what's possible. List processing is a core pattern for much of computation. The point in these lessons is for students to see the pattern in action *a few times* to get the gist.

For example, in computer science, writing computer programs to **model and simulate** real world events is a hugely important topic. The idea of using randomness or random sampling over a large number of trials to obtain a numerical result is a foundational practice in computing. We address it briefly here with the coin flipping experiment, in which students write a program to model flipping a coin repeatedly while keeping track of the results in various ways. This type of method is broadly known as the “Monte Carlo method” and could be used in any situation to determine the probabilities of certain outcomes. Monte Carlo methods have been used to model drivers' behavior in traffic, the flow of multiple fluids, business risk models, and so on.

Our Approach to the Content

The teaching patterns for this chapter are similar to prior lessons in Unit 5 -- (1) introduce and motivate the concept, (2) do some skill-building and practice with the code related to that concept, (3) complete a project.

We want to encourage students to continue working with a partner or “programming buddy” - a person that they can check their work with, clarify instructions, etc. The model we suggest is two students sitting side by side, one with the instructions up on her screen, while the other writes code on hers. Since the projects are usually individual and creative, there is no risk in students helping each other along the way.

Even though the coin flipping experiment seems simplistic, it has the same root elements of more sophisticated models. The main takeaway for students should be this: when some computation is too long or complicated to do by hand with mathematics, if you can think of how to represent or model the thing using the tools of programming such as variables, loops, and conditions (lists), then you can run a simulation a million times to approximate a result.

It's also worth pointing out a deliberate connection between processing arrays in this chapter and the “**Human Machine Language**” problems students worked on in Unit 3, where they designed algorithms and programs to process a list of playing cards. You can appeal to some of those exercises in this work here.

There are numerous aspects to using lists, and the concept takes some time to sink in. Doing a linear pass over an array (a loop that starts at the front of a list and does something to or with each element one at a time until it reaches the end) is the most sophisticated programming technique students will encounter in the course and be expected to reason about in an exam situation.

Performance Tasks

In Units 1-5 students learn the content and practice the skills they need in order to succeed on the AP CSP Performance Tasks. Here, in the “Performance Task” section, the curriculum includes lessons to help instructors and students prepare and make a plan for completing the AP Performance Tasks in the time allotted.

Hours **AP Tech Setup**

2

Tech Setup: Your AP Digital Portfolio and other tools

Contains instructions for setting up and registering your AP Digital Portfolio - the website students need to use to upload the content and written responses for their Performance tasks. It also contains suggestions and information about some of the other technical aspects of digital project submission such as how to make a PDF, how to capture video of your screen, and how to use the written prompt templates.

Hours **Explore Performance Task Lessons**

10

Part 1: Prepare and Plan

Explore PT Prep: Reviewing the Task

Students complete activities to familiarize themselves with Explore Performance Task, how it is scored, and some sample task submissions provided by the College Board.

Explore PT Prep: making a plan

Students make a plan for completing the Explore PT within the allotted time. Students 1) Make a checklist of task elements 2) Estimate the number of hours required for each 3) Prioritize task elements. In addition, students will review good research practices and strategies.

Part 2: Complete

Explore PT: complete the task

This lesson contains guidelines for teachers during administration of the Explore PT. It includes reminders about how you can interact with students while they are working on their projects, and suggestions about timeline. The Explore PT requires a minimum of 8 hours of class time. At the end, students will submit their computational artifact and written responses through their AP digital portfolio.

Hours **Create Performance Task Lessons**

5

Part 1: Prepare and Plan

Create PT Prep: Reviewing the Task

Students familiarize themselves with Create Performance Task, how it is scored, and some example tasks provided by the College Board. Students are introduced to the "while loop" construct by first analyzing a flow chart and then by completing through a series of exercises in Code Studio. The "while loop" repeats a block of code based on a boolean condition.

Create PT Prep: making a plan

Students make a plan for completing the Create PT within the allotted time, estimating the number of hours required for each portion, and prioritizing program features. In addition, students will review good collaborative programming practices.

Part 2: Complete

Create PT: complete the task

This lesson contains guidelines for teachers during administration of the Create PT. It includes reminders about how you can interact with students while they are working on their projects, and suggestions about time line. The College Board requires your provide a minimum of 12 hours of class time for the Create PT. At the end, students will submit their program code, program video, and written responses through their AP digital portfolio.

Materials List & Tech Requirements

Technical Requirements

The course requires and assumes a 1:1 computer lab or setup such that each student in the class 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 modern web browser. For more details on the technical requirements, please visit: code.org/educate/it

While the course features many “unplugged” activities designed to be completed away from the computer, daily access to a computer is essential for every student. It is not required that students have access to internet-connected computers at home to teach this course, but because almost all of the materials are online, it 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 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
- Poster paper
- Markers
- Post-it notes
- Graph paper, etc.

The following items are called for in lessons, but alternatives are also suggested:

Lesson	Materials	Alternatives
Unit 1 Lesson 2	Assortment of craft materials for constructing physical devices. Recommendations: cups, string/yarn, construction paper, flashlights, slinkies, noise makers, markers, and glue, etc.	none
Unit 3 Lesson 1	A handful of LEGO® blocks for every pair students	post-it notes, construction paper
Unit 3 Lessons 2,3	Playing cards (1 deck per 6 students)	write numbers of post-it notes.
Unit 4 Lesson 8	Clear dixie cups, dried beans	Any clear container (ziplock bag, empty water bottle, etc) with any small item (beads, little tinfoil balls, raisins, coffee beans, etc)

AP[®] Endorsement and Alignment

The AP Course Audit Process

Why? In order to have an official “AP” course listed on your student’s transcripts, the curriculum that you intend to teach must be “audited” by the College Board. The process is intended to ensure that (a) a teacher and school administration have confidence that the course meets the necessary guidelines and requirements for AP and (b) colleges and universities have confidence that AP courses that appear on students’ transcripts meet the AP criteria across all high schools.

How it works. If you intend to teach CS Principles using Code.org’s curriculum, the audit process is relatively brief and painless. *Code.org’s curriculum and syllabus have been pre-approved and endorsed by the College Board* as meeting all of the necessary standards and criteria. (If you use your own syllabus you would need to provide and submit this evidence yourself). If you are completely new to this process here are the broad strokes of what you need to do:



1. Create a teacher account on the College Board website
2. Log in to your teacher account, “Add a New Course” and choose AP CS Principles
3. Fill out the Audit form
4. “Adopt Sample Syllabus” and choose the Code.org Syllabus
5. Your school administrator will verify the submission

Detailed instructions for completing the audit can be found on [Code.org’s CS Principles Home Page](http://code.org/csp), (<http://code.org/csp>). Look for the “AP Endorsed” insignia on the page (shown on the right).

After that, since the Code.org syllabus is pre-approved, you should be done and ready to go. You and your school administrator will be notified that you have completed the audit process. Once you have an approved AP course you get access to:

- College Board AP CS Principles teacher community
- Full 74-question practice exam

Planning for the AP Exam and Performance Tasks

The CS Principles AP assessment includes: (1) a 74-question, 2 hour multiple choice exam (2) two *Performance Task* projects that students prepare and submit to the College Board prior to taking the exam.

Preparing for the Multiple Choice Exam

Students should practice multiple choice problems in advance of the exam. Multiple choice questions are derived from the Essential Knowledge statements in the framework.

- **Within the curriculum** the short chapter assessments within each unit are a useful resource for practicing multiple choice questions.
- **The College Board website** has a complete 74-question practice exam you can view once you have completed the audit process
- **In the public domain**, because AP CS Principles is relatively new, there is an ever-growing set of resources available to help students prepare for the exam. We encourage you to seek out and share whatever you can find with CS Principles community and Code.org forum.

Planning for the Performance Tasks

The AP Performance Tasks (PTs) *require* you to set aside a significant number of class hours: 8 hours for the *Explore* Task, 12 hours for the *Create* Task. In your planning you will need to decide when, where and how to allocate those hours. Your decision will likely be based largely on your specific school context, classroom setup, and students.

Before starting the tasks

It is likely that you will need to spend some amount of additional class time simply preparing to complete the tasks. Our curriculum provides some activities related to preparation for the tasks, recommending a total of about **5 hours** engaged in **preparation** activities.

- **1 hour** for setting up and becoming familiar with students' **AP Digital Portfolios**.
The AP Digital Portfolio is where students submit the artifacts for their performance task projects as well as the required written responses that accompany each task.
- **2 hours** preparing for the Create Performance Task
- **2 hours** preparing for the Explore Performance Task
For both tasks we recommend that 2 hours be spent actively reviewing the Performance Task to understand the requirements while also making a plan to complete it that accounts for the necessary hours.

Using the Practice Performance Tasks

It is *highly recommended* that you use the Practice Performance Tasks provided in the curriculum, which model certain elements from the real Performance Tasks and offer good opportunities to preview and foreshadow the the AP Performance Tasks that students will complete later in the course.

The conclusion of the Practice Performance Tasks in Units 2 and 3 is a good time to alert students about the AP tasks and what they entail. The practice task at the end of Unit 2 has similarities to the Explore PT, and the practice task at the end of Unit 3 has similarities to the Create PT. It is useful if students are largely familiar with the tasks, and have them in the back of their mind for a good period of time, *before* starting the task in earnest.

See the *Planning the Year* section later in this document to see suggestions and strategies for allocating hours for Performance Tasks throughout the year.

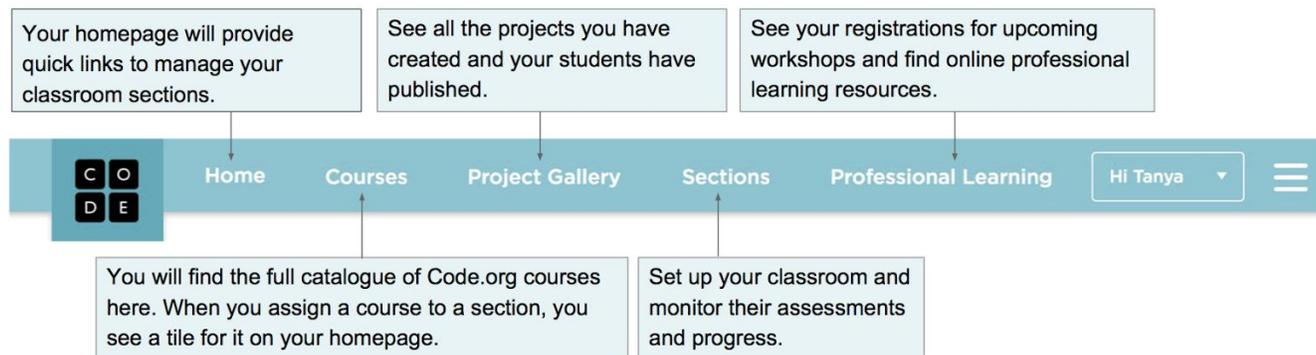
Alignment with AP CS Principles Framework

For more details on how the Code.org curriculum maps to the College Board framework for CS Principles, see the full course syllabus in the **appendix** at the end of this document. Alignment information is also available in each lesson plan.

Planning and Pacing

Navigating the Code.org Website

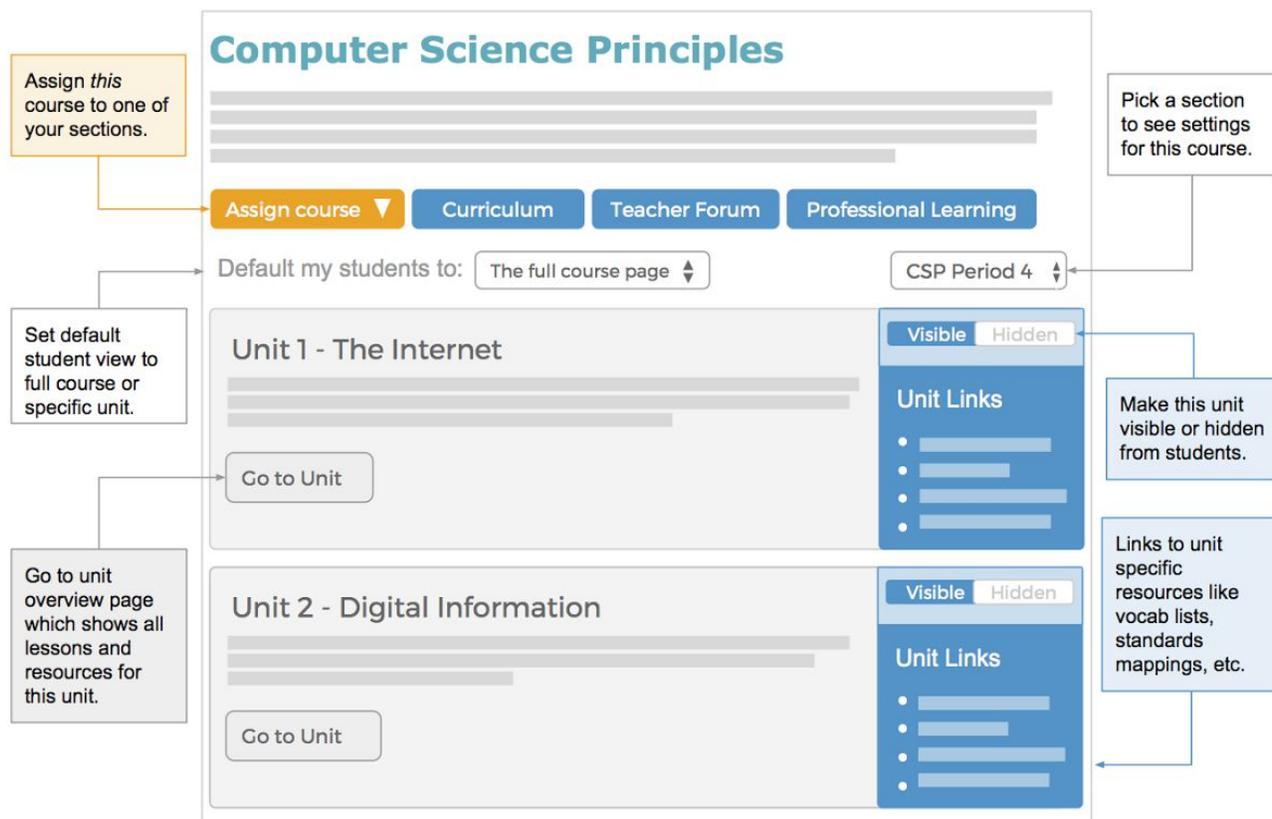
Log into Code.org with your teacher account. The website header will help you navigate the site:



Once you've assigned CS Principles 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:



How to use the Curriculum Unit Overview Page

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

The screenshot shows the 'Unit 4 - Big Data and Privacy' overview page. At the top, there is a navigation bar with 'UNIT 4', 'Ch. 1', and lesson numbers 1-9. Below this is a 'Weekly calendar' showing lessons 01-09 across three weeks. The main content area includes 'Chapter 1: The World of Big Data and Encryption', 'Big Questions', 'Enduring Understandings', and a list of lessons (Lesson 1, Lesson 2, Lesson 6, Lesson 7, and an Optional Lesson). A 'Chapter Commentary' section is also visible.

Callout Boxes:

- Top Left:** Get to this page by clicking on the big UNIT number from any lesson plan (or the overview link in the purple link bar)
- Top Right:** Handy links to cumulative resources for the whole unit. E.g. Vocab list for the whole unit.
- Right Side (Top):** Weekly calendar shows rough pacing of lessons.
- Right Side (Middle):** Listing of Enduring Understandings from the CSP framework that get particular focus in this chapter.
- Right Side (Bottom):** Every unit is a "story" that is broken into "chapters". Each chapter is a collection of lessons.
- Bottom Right:** Optional lessons are enrichment activities that explore certain concepts more deeply.
- Left Side (Top):** Big questions, also known as "framing" questions give the big picture of the chapter.
- Left Side (Middle):** Week by week listing of every lesson with brief summary and links to the full lesson plan.
- Left Side (Bottom):** Direct links to lesson resources e.g. videos, handouts, guides.
- Bottom Left:** Explains the "story" of the chapter and the code.org approach to the content.

How to interpret the suggested pacing calendar

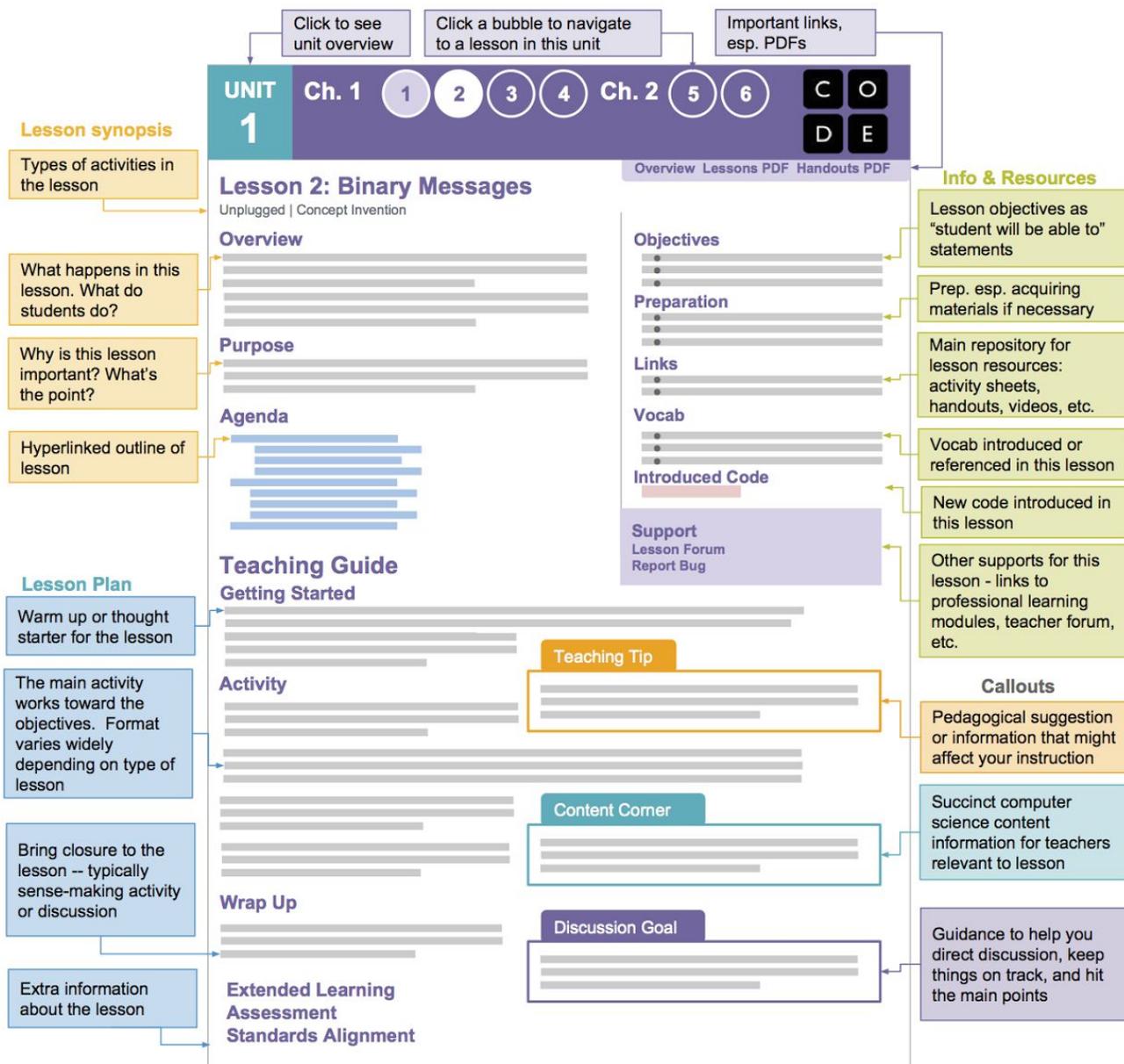
The calendar on the unit overview page shows the relative "size" of each lesson and suggests what you might be able to get through in a week.

If lessons span less than the full width of the calendar (like weeks 2 and 4 at right), it means you've got a little buffer time to work with. You can absorb the time if the previous week went long, or start the next week early, time permitting.



Understanding 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.



CS Principles lessons are big by design

Lessons in CS Principles are written for a wide variety of classrooms, and most lessons include optional activities and extensions to help you tailor the course to your students' needs and interests. If you use every suggested activity, lessons will easily *run longer than a typical 45-60 minute class period*. You should assume the "Activity" portion of the lesson is core, and depending on your students' needs you can alter or cut pieces of the getting-started activities, wrap-ups, share-outs, and optional extensions. The learning objectives, lesson purpose, and teaching tips are designed to help you make these decisions, and our teacher forums can help you understand how other teachers are approaching each lesson.

Other 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.

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.

Activity (40 mins)

Content Corner

Remarks

Discussion Goal

Guide: Encoding Color Images

Teaching tip

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.

Unit 5 Lesson 2 Introduction **Student Overview**

Levels 02 03 04 (click tabs to see student view)

Debugging with Console.log **Teacher Overview** **Student Overview**

View on Code Studio

Teaching Tips

This is a level that might be worth pausing on. There are a few things that are important to reinforce:

- Every programming language has some way of display simple plaintext output
- Printing to the console is a very common technique not only for debugging but also as an aid in program construction and development.
- In other words, you can use console.log to send messages to yourself to verify the program is doing what you think it's doing which helps prevent errors down the line.

Levels 06 07 08 (click tabs to see student view)

Student Instructions

Add onEvent from Design Mode!

Design Mode has a shortcut for you to add `onEvent` to your code for specific UI Element. It's quite handy.

You try it: Add an event handler to a button from Design Mode

Design Mode * A Red button has already been added to the app. * In Design Mode click on the button to select it * Click the **EVENTS** tab and then click the link that says **Insert and Show Code** * You may change the ID if you like

PROPERTIES EVENTS

id redButton

text Button

width (px) 80

height (px) --

Planning for the year

When planning for the school year, the largest thing you need to account for - beyond the typical constraints of school calendar, class hours, etc. - is making sure that your students have time to complete and submit the Performance Task projects. Working backwards from the AP deadlines, we recommend you do the following:

1. Decide when/how to allocate performance task hours based on your school calendar
2. Slot each unit into your school calendar
3. Allocate time for in-class assessments
4. Decide what material to trim if necessary

1. Allocating Performance Task Hours

Given the nature of the Performance Tasks, school calendars, and student habits, you must plan ahead and allocate the hours for each task and decide when to start them. It's also important to note the constraints and expectations from the College Board with regard to Performance Task completion. You can learn more about the task administration requirements on pages 76 (Create) and 83 (Explore) of the Course and Exam Description, but key pieces include:

1. You must provide 8 dedicated in-class hours for the Explore Task and 12 dedicated in-class hours for the Create Task
2. During task administration you cannot intervene and help students (so all prep needs to happen before they start the task)
3. Tasks must be submitted by the end of April 2018

There are several implementation options for Performance Tasks, and the primary constraints on your planning come from the fact that you cannot provide guidance while students are completing the tasks. This means you will need to cover all of the concepts and provide opportunities to practice the task before starting the real thing. As noted on page 24, the Code.org CS Principles curriculum has a set of lessons designed to lead directly into each performance task. These lessons (which take approximately 2 hours each for the Create and Explore Tasks) provide space for students to do a final review of the tasks, rubrics, and sample submissions, and to make a plan for how they will spend their in-class task hours.

You know your school and students best. Familiarize yourself with each task and then think about where the hours will fall by planning against your school calendar. Key considerations for each performance task are as follows:

Consideration	Explore Task	Create Task
Total time needed for task review, planning, and completion	10 hours	12 hours
Instructional units to cover before completing the task	through Unit 4	through U5 L10

Also, examine the possibility of completing different elements of the tasks over time. Here are some ideas for spreading tasks over time:

- Preview early in the course**
 Preview the real performance tasks as part of completing end-of-unit practice tasks (for example, look at the Explore task with the Unit 2 practice PT and Create along with the Unit 3 practice PT)
- Start early, start briefly**
 Give students 1-2 hours to start on tasks (select topics, do preliminary research for Explore, map out some program features for Create, etc.) before they've finished all instructional units required for the task
- Change it up from time to time**
 Mix days spent on the task with days spent covering unrelated instructional units (for example, students could work on the Explore Task one day a week while moving through unit 5)

Two example performance task pacing implementations are listed below. The example on the left demonstrates how you might spread the tasks out over time. The example on the right shows how you might complete the tasks in continuous blocks of time immediately following the final instructional unit required for each task:

Curriculum Break Point	Pacing Example 1 <i>split things up over time</i>	Pacing Example 2 <i>complete the task fully as soon as possible</i>
End of Unit 2 along with Practice Explore PT	Preview EXPLORE Task	
End of Unit 3 along with Practice Create PT	Preview CREATE Task Setup Digital Portfolio and other AP tech requirements	Set up Digital Portfolio and other AP tech requirements (2 hours)
End of Unit 4	Start EXPLORE Task ~3 hours	Start and Complete all of the EXPLORE Task (10 hours)
After Unit 5 - Lesson 10	Start CREATE Task ~4 hours	Start and Complete all of the CREATE Task (12 hours)
While doing Unit 5 - Lessons 11-19*	Interleave completion of both tasks and written exam prep with U5 Lessons 11-19 <ul style="list-style-type: none"> ~4 hours EXPLORE (upload draft) ~8 hours CREATE (upload draft) Prep for Exam 	Prep for the written exam while completing Unit 5 Lessons 11 - 19
Submit PTs at end of April 2018 Sit for written AP Exam in early May 2018		

2. Slot each unit into your school calendar

If you look at the Course-at-a-Glance page and add up the weeks, you'll see that Units 1-5 contain 26 weeks of material. For a school that starts in September, after Labor day, and given typical school breaks and off time, you have about 30 weeks until the AP exam. Thus, our curriculum and pacing is built around 26 weeks of material, plus 4 weeks for the performance tasks.

In reality, you would have to maintain a fairly frenetic pace to complete *every part* of Units 1-5 and the Performance Tasks in that time frame, but given your plan to introduce and complete the performance tasks, you should slot in the units given the suggested pacing to see where school breaks fall and how much of squeeze it will to be. Doing this will help you prioritize activities later.

3. Allocate time for assessments

The end-of-chapter assessments in the course are not huge undertakings, but they are unaccounted for in the general pacing schedule. It's recommended that you add a day of buffer to your planning for each chapter assessment.

Additionally, you should ensure that these assessments, or others you design, suit your needs in terms of your school's marking periods. See more details about assessment types included in the course materials below.

4. Make decisions about what / how to trim

General rule: Go slow to go fast

As a general rule we recommend “go slow early in order to go fast later”. In other words, cut things out or take shortcuts *later* rather than earlier in the course. If you're worried about keeping pace, the time and effort you put in early on to establish the classroom environment, will enable both you and your students to move more quickly later on.

Key Understanding: lessons as standalone v. sequenced entities

Every lesson in CSP is written as though it needs to operate as a standalone entity that contains a core activity (or activities) that covers a topic, complete with all the trappings of engaging warm up and wrap up activities, extensions for learning more, and so on. The theory is: if this lesson were ripped out of the curriculum and delivered as an one-off would it hold up? would it make sense? As a result, each lesson is very robust.

When taken in sequence, though, some of those additional trappings can be trimmed, or even ignored because of the material that comes immediately before and after it. Sometimes you might find that even the core activity can take less time (or might not even be necessary) because students “get the point” quickly due to the prior lessons.

One of your key roles as a teacher of this course is to make on the fly decisions about whether or not to include an element of a lesson. These kinds of gut decisions get easier after you've taught the course a few times.

Small adjustments

An easy way to buy time is to merge the wrap-up and getting started discussions between two lessons. Frequently this is possible, especially for lessons with related content. You might even decide to do a few activities back-to-back and do a combined wrap-up as a sense-making activity or assessment afterwards.

If you can give homework, a good strategy would be to assign discussion questions as writing prompts to both give students practice writing and to get some student thinking time done outside of class.

Medium Adjustments

It might be possible to cut or trim whole activities, but it should be done thoughtfully and judiciously. **For example:** whole class unplugged activities are extremely important for a variety of reasons (engagement, equity, multi-sensory, memorable, etc.) especially early on in the course or to establish a foundation for a new concept. However once you've established a good classroom rapport and students are engaged in learning, you might decide that the payoff for doing an unplugged activity isn't worth the time. Modifications include shortening the unplugged activity or in rare cases replacing it entirely with a brief reading, or just explaining it yourself.

Big Adjustments

If you need to modify large chunks of units, it's possible, though obviously there are tradeoffs. The usual tradeoff means using direct instruction or reading to save time over a more prolonged learning experience. In keeping with our recommendation to cut later rather than earlier, here are some key considerations for each unit, working backwards from Unit 5:

Unit 5 Chapter 2 <i>Lessons 11 - 19</i>	While loops, arrays, and functions that return values are important for the multiple choice exam, but they're not required for the Create Task, and students can prepare for exam questions by alternative means.
Unit 5 Chapter 1 <i>Lessons 1 - 10</i>	These programming concepts are essential for doing the CREATE AP Performance task. In particular, conditionals (if-statements) are important for demonstrating an "algorithm that integrates mathematical and/or logical concepts"
Unit 4	These concepts build up to the Explore Performance task. The keys to the unit are about doing internet research and understanding the beneficial and harmful effects of technology. This unit could easily be merged with Explore PT prep.
Unit 3	These lessons can be trimmed significantly, especially if every student has some prior programming experience. The key outcomes from this unit that are necessary for later on (including the Create Task) are (1) decomposing a problem/procedural abstraction and (2) collaborating with others to write a program as described in the final project: Design a Digital Scene.
Unit 2	These lessons build to creating computational artifacts from datasets and spreadsheets, but the artifact required for the EXPLORE performance task no longer require this. Consider cutting lessons with cleaning and analyzing data.
Unit 1	For purposes of the AP exam, students need to understand the "what" of the internet more than than the "how". You may find you can easily shore up understanding with a reading or other resources, and might choose to go into less depth on some of the discovery lessons that emphasize the "how".

Assessments

The course materials contain a number of assessment types and opportunities which can be used formatively (to check for understanding) or summatively (for evaluation).

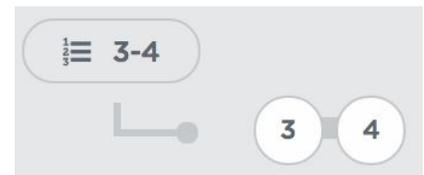
For students, the goal of the assessments is to prepare them for the AP exam and performance tasks. For teachers, the goal is to use assessments to help guide instruction, give feedback to students, and make choices about what to emphasize in lessons.

Code Studio includes features that assist the teacher in completing formative and summative assessments:

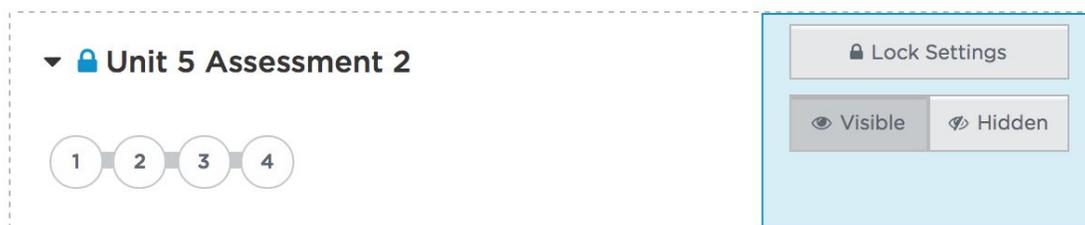
- Multiple choice or matching questions related to questions on the chapter summative assessment.
- Free-response text fields where students may input their answer.
- Access to student work within the App Lab programming environment and other digital tools and widgets used in the curriculum.
- The ability for students to submit final versions of App Lab projects

Fixed Response Assessments on Code Studio

Lesson assessment items - You will find assessment items (multi-choice, free-response text) embedded in individual lessons on code studio, typically as the last few “bubbles” for a lesson, indicated with an assessment icon. These are intended to be used as *formative* assessment items. Students can always see them and change their responses at any time.



Chapter assessments (lockable) - are typically 10-15 question multiple choice tests intended to mimic AP-style questions. They look like their own lesson on code studio and have *lock settings* as well as the usual visibility settings. These can be use for formative or summative assessments.



Lock settings enable or disable students from modifying their answers. For example, you may want to hide an assessment before students get to it, but after students take it, you might want it to remain visible but locked, while you review the answers.

Practice Performance Tasks

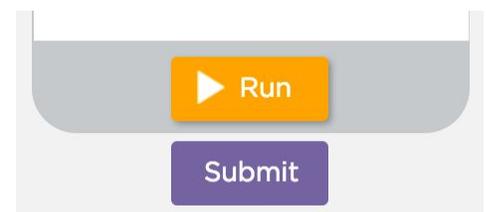
Each unit contains at least one project designed in the spirit of the Advanced Placement Performance Tasks (PTs). These **Practice PTs** are smaller in scope, contextualized to the unit of study and are intended to help prepare students to engage in the official administration of the AP PTs at the end of the course. Practice PTs come with:

- Project description
- Written response prompts similar to AP-style
- Project rubrics modeled after the AP rubrics

Practice PTs are the most authentic way for you and your students to prepare for the *real* PTs. Use them!

Submittable Programming Projects

When the curriculum calls for a programming project for assessment the App Lab environment will show a “submit” button below the typical “run” button. When a student submits a project it is submitted with a timestamp and locked for teacher review. It also shows up in a special area of the teacher dashboard. The teacher can release the project back to the student as well.



Worksheets and Activity Guides

Worksheets and activity guides are good opportunities for assessment. Worksheets or activity guides often ask students to write, answer questions, and respond to prompts. When available, answer keys for worksheets and activities are provided through the “teacher only” panel on Code Studio.

It is up to the classroom teacher:

- to determine the appropriateness of the assessments for their classrooms
- to decide how to use, or not to use, the assessments for grading purposes. The curriculum and Code Studio does not provide teachers with a gradebook, and we do not provide recommendations for how to assign grades based on performance on an assessment.

Tools and How-To guides

We use many different learning tools in the CS Principles curriculum, but each tool can be represented in one of four groups: **widgets**, **the Internet Simulator**, **App Lab**, and **external tools**. The following chart shows where in the curriculum each type of tool is used:

Tool Type	Unit	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
Widgets			Lossless Text Compression (L 2) Pixelation (L 3-4) Lossy Text Compression (L 5)		Cryptography (L 4) Public Key Cryptography Widget (L 6)	
Internet Simulator		Sending bits, encoding info (L 3, 6, 7) Broadcast and addressing (L 9) Routers and redundancy (L 10) Packets (L 11) DNS (L 12)				
App Lab				Turtle programming (all lessons)		Event-driven programming (all lessons)
External Tools			Google Trends (L8) Spreadsheet tools (i.e., Google Sheets or Excel)		Password strength checker tool (L 4)	Screen capture software to make a video

Widgets

Widgets are small digital tools that act as a playground for exploring and experimenting with a CS concept. Widgets are meant to promote discovery and creativity within a fairly narrow scope where there are typically no right or wrong answers. One way of thinking about widgets is that they are a “plugged in” version of an unplugged activity. There is often a paper and pencil corollary, but having a widget lets you experiment with different approaches to problems more quickly, and the widget can enforce the rules rather than having to eyeball it on paper.

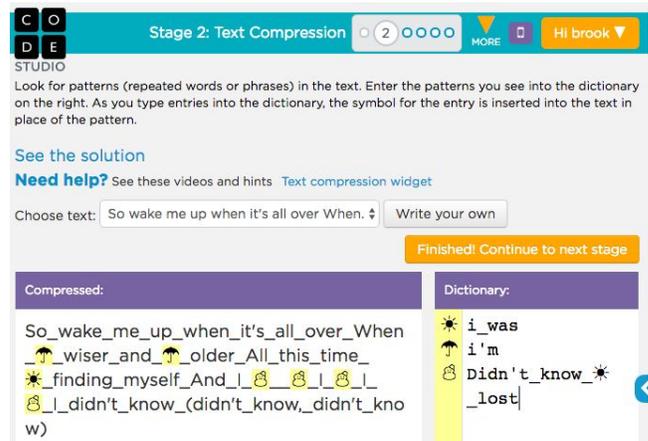
Below, we briefly describe each widget used in the course, the concepts they cover, and connections to other parts of the curriculum.

Text Compression Widget

Where we use it: Unit 2 Lesson 2 (studio.code.org/s/csp2)

Link to the stand-alone version: code.org/textcompression

This widget lets students interactively experiment with compressing a piece of text by identifying patterns in text, storing those patterns in a “dictionary,” and replacing the repeated pattern with a 1-byte symbol to create a compressed version of the text. The widget updates with every keystroke and also performs the compression calculations, so you can see if you’re increasing or decreasing the total file size in real time. Since figuring out the optimal amount of compression is a computationally hard problem (i.e. there is no known algorithm to find or verify that the optimal compression has been found), students can experiment with many different approaches rather than focusing on finding the ‘right’ answer. The most intriguing idea to play with is figuring out how to best harness the power of representing patterns of patterns. Done right, it can dramatically improve compression; done wrong, it can...dramatically go very wrong.



Concepts

The primary concept here is related to **lossless compression**, but the overarching concept is about abstraction in the representation of information. Compressing text this way can be viewed as a logical extension of text representation in binary, which is explored throughout Unit 1. The challenge is to think about how one can precisely represent the exact same information with fewer bits. The technique used in this widget - maintaining a dictionary of repeated patterns of text - is used in .zip compression, and is also more or less the same technique used to compress images in .gif format.

Curriculum connections

This widget is a very fertile example to refer back to later in the course in two main areas. First, the kinds of analysis and problem solving students do in finding repeated patterns and then expressing those patterns as a single reusable abstraction is very similar to the kinds of thinking students do in Unit 3 when developing procedures in programming. Second, this activity has a strong relationship to the computationally hard problems upon which modern encryption relies, which we address in Unit 4 of the course.

Pixelation Widget

Where we use it: Unit 2 Lesson 3 - 4 (studio.code.org/s/csp2)

Link to the stand-alone version: code.org/pixelation

This widget lets students compose an image “in binary” by filling in binary information and the widget renders the image that the binary represents. It’s like having an instant binary interpreter that obeys the rules of the agreed upon image format. The widget has a few variants of increasing sophistication that are used over a few lessons. It starts with a very simple black-and-white image format, and ends with up to 24 bits of color information for each pixel.



Concepts

What students will grapple with the most is understanding the RGB color system and the tradeoffs between the number of bits needed to represent an image (file size) and how precise the color information is. This is also a great tool for seeing the benefits of the hexadecimal number system for representing binary information. A practical takeaway should be an understanding of why an RGB color is broken into red, green, and blue values that are each represented with a number between 0-255, and the factors that influence image file sizes and how they get so large when uncompressed.

Curriculum connections

You can refer back to this lesson when RGB colors come up in the programming unit. You can also return to this activity if the idea of image compression comes up in the next lesson about lossy compression and file formats.

Frequency Analysis Widget

Where we use it: Unit 4 Lesson 5 (studio.code.org/s/csp4)

Link to the stand-alone version: code.org/frequency

This widget lets you play with two classic substitution ciphers, one known as the Caesar Shift (encryption by shifting each letter of alphabet the same amount) and random substitution (encryption via a 1:1 substitution of one letter for another, but randomly assigned as opposed to a uniform shift).

Concepts

More than anything this tool is meant to expose how simple it is to crack a substitution cipher when armed with a tool that does a very simple frequency analysis. Students will also get a feel for the kinds of techniques that have been used historically to encrypt secret messages, which is a foundation for later discussions about current encryption techniques. The widget is also a concrete thing you can use to point out key terms that come up in encryption and security contexts: cipher, encrypt, decrypt, symmetric encryption, key, plaintext, crack, etc.

Curriculum connections

The distance between a protocol for encoding information (including image encoding, text compression etc.) and encryption is pretty short. You might refer back to the text compression widget in particular to ask whether or not text compression is a form of encryption.

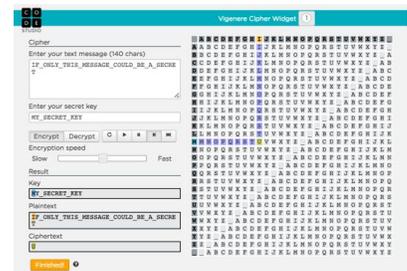


Vigenere Cipher Widget

Where we use it: Unit 4 Lesson 6 (studio.code.org/s/csp4)

Link to the stand-alone version: code.org/vigenere

This widget is used in the same lesson as the the random substitution cipher, and allows students to play with a more sophisticated encryption technique that was considered (basically) unbreakable for a very long time. The Vigenere cipher is similar to the Caesar or random substitution cipher, but the shift changes for each letter based on a secret key.



Concepts

This tool gives students a sense of the relationship between password strength and the strength of the resulting encryption. While modern encryption doesn't work this way anymore, the root of the concept is the same. It is an example of strong symmetric encryption - one where the key to encrypt and decrypt is the same. The message is only as secure as your ability to choose a strong password and keep it out of the wrong hands.

Curriculum connections

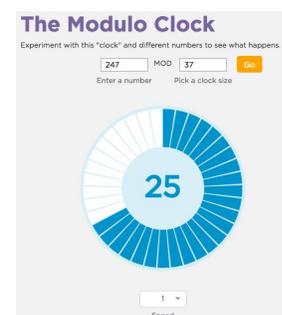
Though not given by name, this is the same technique demonstrated in the encryption video shown in unit 4 lesson 8-. In the video the widget uses a numeric key (the numbers 0-9) to dictate the shift rather than letters of the alphabet, but you can connect what's in the video to this widget. After using the Caesar and Vigenere widgets, an interested student can further explore history of cryptography, which can be quite a rabbit hole.

Public Key Crypto Widget

Where we use it: Unit 4 Lesson 7 (studio.code.org/s/csp4)

Link to the stand-alone version: code.org/publickey

This widget allows students to interact with a form of asymmetric encryption (one in which the keys to encrypt and decrypt are different) that mimics modern Public Key Cryptography. Students act out the roles of two people trying to send secure messages back and forth with an eavesdropper in the middle. The exchange should demonstrate how it's easy for Alice to send a Bob a secret message without agreeing on a secret password ahead of time, and in such a way that the woman-in-the-middle (Eve) has no way to determine the individual keys other than trying every password combination possible.



Concepts

Most obviously, this widget lets students experiment with public key cryptography which can be a very tricky concept for novices to explain. This lesson is also the most directly mathematical since certain mathematical properties must exist between two keys when one can only encrypt and the other can only decrypt. The widget also shows a visual way of representing the mathematical operation of modulo (as a clock). The modulo operation is at the heart of making "one way" functions (i.e. things that are easy to compute in one direction, but hard to reverse).

Curriculum connections

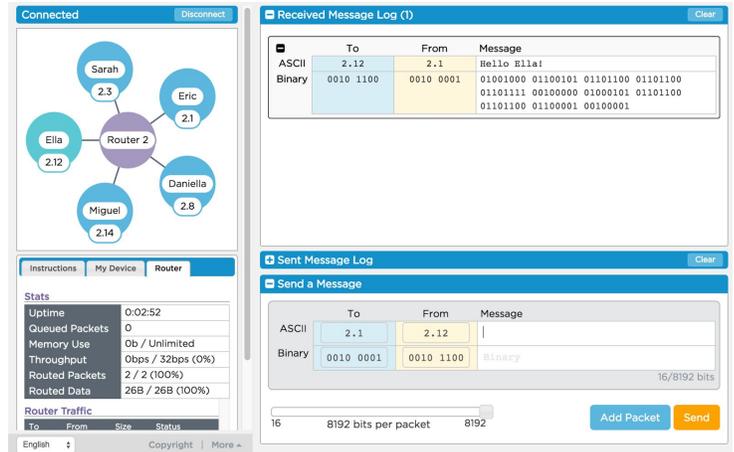
Since public key cryptography is the most common form of encryption used in just about everything today (including secure web sites), connections to real life examples abound. You can appeal to this activity in almost any discussion about modern encryption and how it works. It explains why, for example, a company can make encryption software in such a way that they cannot reverse the encryption themselves.

Internet Simulator

Where we use it: Unit 1 (studio.code.org/s/csp1)

Link to the stand-alone version: code.org/netsim

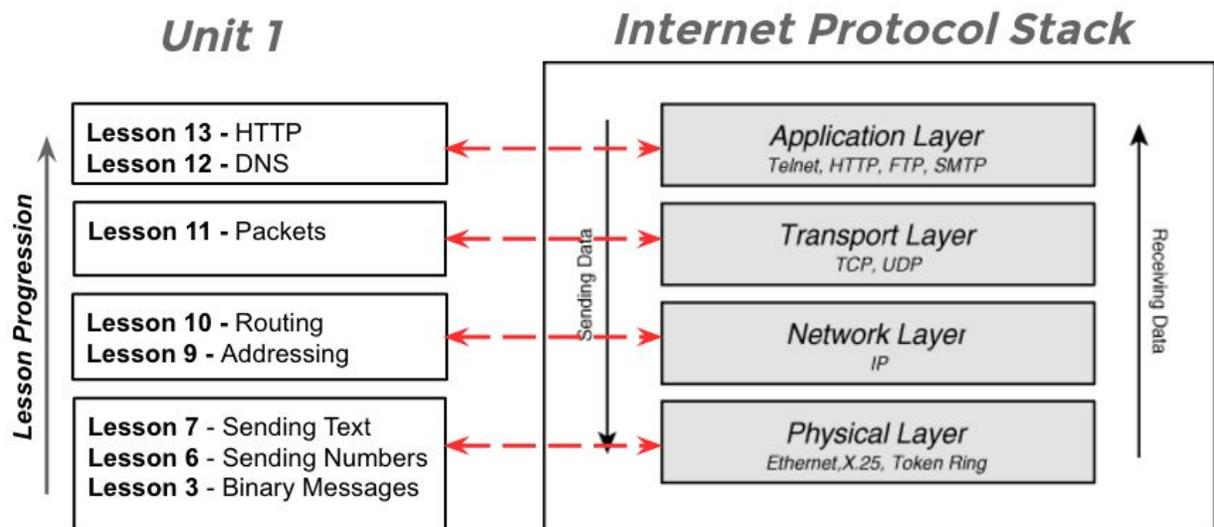
Similar to a widget, but much larger in scope, the Internet Simulator is designed to let students visualize, experiment with, and solve different kinds of problems associated with networked computers in a hands-on way. Often these problems involve inventing a communication protocol, or inventing ways to encode information that makes transporting it over the Internet feasible.



It is essential to note that we use the Internet Simulator for **much more than teaching Internet Protocols**. The Internet Simulator

contextualizes exploration of deeper concepts in computer science, like the use of abstraction-to-solve problems and the binary representation of information. The goal of the Internet Simulator is not merely to present the functionality of the different layers of the Internet, but to provide an opportunity for students to reason about why those structures exist and even develop their own solutions to the problems solved by the systems of the internet.

The simulator is configured differently in each lesson to enforce different rules or to expose different behaviors of the internet that students must creatively problem solve around. Specifically, each version of the Internet Simulator is configured to mirror a high level version of the layered Internet Protocol stack. With each lesson the Internet Simulator changes to incorporate the solution to the previous problem students solved. In this way we work from the bottom up, first solving physical coordination problems with sending bits back and forth, then addressing (IP), then packeting (TCP), then name-to-address mapping (DNS), and finally HTTP.

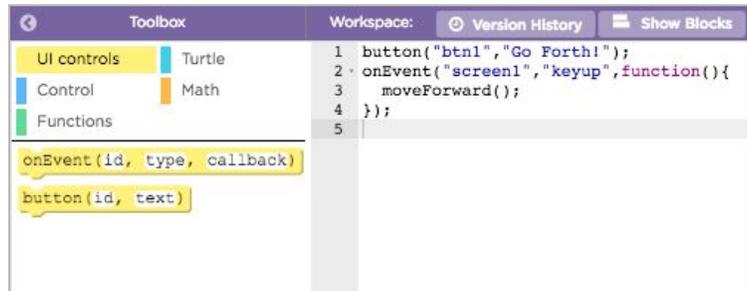


Internet Simulator in the Curriculum

Lesson 3 Sending Binary Messages with the Internet Simulator	<p>Configuration: Peer-to-Peer - Sending Bits on a Shared Wire (“NetSim A/B”)</p> <p>Details: This is the most primitive configuration, which simulates two bodies connected by a wire that can hold one of two states - state A or state B - and you can either set the state of the wire, or read its current state. The problem to solve is a coordination problem. For example: if I set the state of the wire to state A, and then I read the wire, it might be in state A - but is it that way from how I just set it? Or did my partner also try to set it to A? To communicate we need to agree on some rules for who can send what and when.</p>
Lesson 6 Sending Numbers	<p>Configuration: Peer-to-Peer - Sending Numbers (Binary)</p> <p>Details: In this configuration you can send and receive streams of bits (0s and 1s) with a partner, and anything sent to you will automatically show up in your received messages queue. You can also view the decimal versions of the binary numbers but you must set the “chunk size.” For example, if you receive 16 bits, that could be one 16-bit number, two 8-bit numbers, four 4-bit numbers, and so on. The interpretation depends on your protocol. The problem to solve here is how to encode a list of numbers that represent the coordinates of line drawing.</p>
Lesson 7 Encoding and Sending Formatted Text	<p>Configuration: Peer-to-Peer - Sending ASCII (Binary/Decimal/ASCII text)</p> <p>Details: In this configuration you are shown the ASCII representation of the binary messages sent between partners - essentially creating a text messaging device. The problem to solve is how to represent formatted text when you can only send ASCII characters. It requires inventing some kind of formatting code (like HTML).</p>
Lesson 9 The Need for Addressing	<p>Configuration: Broadcasting Messages (“Broadcast mode”)</p> <p>Details: This is the first “networked” configuration that students see. In this mode you join multiple people (up to 6) at the same time, and every message is sent is “broadcast” to all the others simultaneously. This simulates multiple computers sharing the same network backbone. The problem to solve is how to ensure that messages get to and from their intended recipients - some sort of addressing protocol must be invented.</p>
Lesson 10 Routers and Redundancy	<p>Configuration: Routers and Addresses (“Router Mode”)</p> <p>Details: In this configuration every person is assigned an 8-bit “IP address.” When you start, you “join” a router, which also has an address. You must enter the proper “IP address” for a message to get the intended recipient and each message “hops” across multiple routers to get to its final destination. In this configuration you can also view the router logs for every router in the network to see how the traffic bounces around to get where it was going. The lesson is primarily investigative in nature versus problem-solving.</p>
Lesson 11 Packets and Making a Reliable Internet	<p>Configuration: Packets and Reliability (“Router Mode with dropped packets”)</p> <p>Details: In this configuration the packet size is limited to only a few characters beyond the to and from addresses required by the protocol. Traffic is routed, but unreliably. Roughly 10-20% of packets get “dropped” or lost on the way to their destination. This simulates the (true) fact that computer networks experience problems all the time resulting in lost or dropped packets. The problem to solve is how to invent a protocol that can reliably get all the pieces of information to the intended destination (eventually) when the network is so unreliable. Students should invent something akin to the TCP system of numbering packets with some protocol for requesting or re-sending missing packets.</p>
Lesson 12 The Need for DNS	<p>Configuration: Automatic DNS (“DNS mode”)</p> <p>Details: In this configuration users’ IP addresses are hidden, but each router has a DNS server attached to it whose IP address is known. You can request a user’s IP address by sending a GET request to the the DNS server for example: “GET janesmith1” and the DNS server will send back a message with janesmith1’s IP address. This mode is primarily used as an investigation into the rudiments of DNS more than solving a problem. In the lesson the problem is solved as an unplugged activity.</p>
Lesson 13 HTTP and Abstraction on the Internet	<p>Configuration: “Free Play mode” (optional)</p> <p>Details: While not featured directly in this lesson, the full-featured version of the Internet Simulator (“free play mode”), in which all of the dials and settings are available to use and tweak, serves as a useful reference point when thinking about challenges involved in developing rich protocols that communicate using only plain text like HTTP.</p>

App Lab

App Lab is an Integrated Development Environment (IDE) for building web applications in JavaScript. The tool is designed for new learners with the general idea that a student should be able to take an idea in their head and rapidly create a functional prototype in App Lab. A powerful feature of App Lab is that with it you can program either by drag-and-drop blocks or by typing text, and you can easily toggle back and forth between them. This allows for the best of both worlds: composing a piece of code with block structures, but getting into the text to tweak it.



```
1 button("btn1", "Go Forth!");
2 onEvent("screen1", "keyup", function(){
3   moveForward();
4 });
5
```

App Lab is also chock full of supports for learners and users including:

- integrated documentation with fully working examples
- interactive debugging console
- a full debugger with breakpoints and line stepping
- WYSIWYG “design mode” that lets you compose an app screen with drag-and-drop HTML/CSS.

Like any full-fledged programming environment, taking it all in at the beginning can be intimidating. Thus, in the curriculum we use App Lab through the Code Studio environment in which the programming “toolbox” is scoped to the lesson or problem at hand. This dramatically reduces cognitive load and allows the student to focus on solving the problem within the constraints of the environment. As the course proceeds, more and more of the commands and features of App Lab are exposed.

External Tools

One purpose of the course is to enable students to use digital tools in the real world. Where appropriate, the curriculum uses a variety of other tools such as Google Trends, Spreadsheet software, and video screen capture software. We use an external tool in the curriculum under one or more of the following conditions: 1) an existing widget-like tool does a good job allowing discovery of an idea in line with the pedagogical ethos of our course (e.g. password strength checker), 2) there is no good substitute for the “real world” version of it (e.g. spreadsheet and graphing tools), 3) the student will benefit more from using the external tool than something we develop in-house. In particular, we want to prepare students to use tools that they will need to make documents (pdf) and videos (screen capture) for the AP Performance tasks.



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 the 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		
<i>What</i> questions do you have?	<i>Where</i> can you find answers those questions?	<i>When</i> do you plan to get answers to those questions?
Things to explore further		
<i>What</i> topics in the curriculum do you want to further explore before you start teaching the course?	<i>Who</i> can you work with in exploring these topics?	<i>When</i> 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	<i>5 weeks</i>			
Unit 2	<i>6 weeks</i>			
Unit 3	<i>4 weeks</i>			
Unit 4	<i>4 weeks</i>			
Unit 5	<i>7 weeks</i>			

Pacing and Planning: AP Exam and Performance Tasks

Use the space below to document your pacing plan for completing the performance tasks and AP Exam Prep.

What	Duration	When do you plan to start?	When do you plan to finish?	Notes or special considerations
AP Tech Setup	2 hours			
Create Task	14 hours		(Due end of April 2018)	
Explore Task	10 hours		(Due end of April 2018)	
Written Exam Prep	<u> </u> write in amount of time you plan to spend			

Use the space below to map out specific AP-related activities you plan to do and when, relative to key break points in the curriculum.

Curriculum Break Point	Explore PT Activities	Create PT Activities	Written Exam Activities
End of Unit 1			
End of Unit 2- Practice Explore PT			
End of Unit 3- Practice Create PT			
End of Unit 4			
End of Unit 5 - Lesson 10			
Unit 5 - Lessons 11-19			

Appendix B: 2-D View of the Framework

A 2-Dimensional View of the AP CSP Framework

The [CS Principles Framework](#) outlines seven “Big Ideas” of computing, and six “Computational Thinking Practices”. Activities in the course should ensure that students are engaging in the Computational Thinking Practices while investigating the Big Ideas.

These *Big Ideas* and *Practices* are not intended to be taught in any particular order, nor are they units of study. In fact, a single learning objective like:

LO 2.2.1 Develop an abstraction when writing a program or creating other computational artifacts. [P2]

actually represents an intersection of two *Big Ideas*: Abstraction and Programming, while also engaging at least two *Computational Thinking Practices*.

Intersections

This curriculum takes the view that the 7 *Big Ideas* actually represent a body of knowledge in which topics of study (The Internet, Programming and Data) intersect with cross-cutting principles of computing (Creativity, Abstraction, Algorithms and Global Impacts). For example, if Programming is a topic of study then to “cover” programming, you need to address Creativity, Abstraction, Algorithms, and Global Impact. In the same vein, you know that you’ve “covered” the cross-cutting idea of Abstraction, when you have learned about abstractions on the Internet, with Data and in Programming.

A mental framework for the course is much more usefully viewed in two dimensions, plotting activities or learnings at the intersection of two big ideas.

	Internet	Data	Programming
Creativity	Invent a communication protocol	Visualizing Data Create a visualization	Make a digital scene. Program an app.
Abstraction	Internet Protocols	Encoding images in binary	Writing procedures and functions
Algorithms	Routing, Encryption	Data Compression, Searching and data mining	String manipulation Array processing
Global Impact	Security, Privacy, Hacking	Implications of collection and storage of big data	Software can solve some but not all problems
	Units 1, 4	Units 2, 4	Units 3, 5

For the Code.org curriculum **units 1, 2 and 3** survey the the Big Ideas *Internet*, *Data* and *Programming* as major topics of study. For coverage we look at their intersections with the other four big ideas: *Creativity*, *Abstraction*, *Algorithms*, *Global Impact*. **Unit 4** goes deeper into big data, especially related to the societal implications of big data. And Unit **Unit 5** goes deeper into programming concepts.



Appendix C: Course Syllabus

AP[®] Computer Science Principles

Code.org’s Computer Science Principles (CSP) curriculum is a **full-year, rigorous, entry-level course** that introduces high school students to the foundations of modern computing. The course covers a broad range of foundational topics such as programming, algorithms, the Internet, big data, digital privacy and security, and the societal impacts of computing. The course is designed for typical school settings with teachers in classrooms. All teacher and student materials are provided for free online.

AP Endorsed

Code.org is recognized by the College Board as an endorsed provider of curriculum and professional development for AP[®] Computer Science Principles (AP CSP). This endorsement affirms that all components of Code.org CSP’s offerings are aligned to the AP Curriculum Framework standards, the AP CSP assessment, and the AP framework for professional development. Using an endorsed provider affords schools access to resources including an AP CSP syllabus pre-approved by the College Board’s AP Course Audit, and officially recognized professional development that prepares teachers to teach AP CSP.



AP is a trademark registered and owned by the College Board.

Curriculum Overview and Goals

Computing affects almost all aspects of modern life and *all* students deserve access to a computing education that prepares them to pursue the wide array of intellectual and career opportunities that computing has made possible. Here is a brief summary of each of the units in the Code.org CSP curriculum.

Unit 1: The Internet	Learn how the multi-layered systems of the Internet function as you collaboratively solve problems and puzzles about encoding and transmitting data, both ‘unplugged’ and using Code.org’s Internet Simulator.
Unit 2: Digital Information	Use a variety of digital tools to look at, generate, clean, and manipulate data to explore the relationship between information and data. Create and use visualizations to identify patterns and trends.
Unit 3: Algorithms and programming	Learn the JavaScript language with turtle programming in Code.org’s App Lab. Learn general principles of algorithms and program design that are applicable to any programming language.
Unit 4: Big Data and Privacy	Research current events around the complex questions related to public policy, law, ethics, and societal impact. Learn the basics of how and why modern encryption works.
Unit 5: Building Apps	Continue learning how to program in the JavaScript language. Use Code.org’s App Lab environment to create a series of applications that live on the web. Each app highlights a core concept of programming.

This course is not a tour of current events and technologies. Rather, it seeks to provide students with a “future proof” foundation in computing principles so that they are adequately prepared with both the knowledge and skills to live and meaningfully participate in our increasingly digital society, economy, and culture.

Addressing Diversity, Equity, and Broadening Participation in the Curriculum

A central goal of Code.org’s CSP curriculum is for it to be accessible to all students, especially those in groups typically underrepresented in computing. To this end, we have worked to provide examples and activities that are relevant and topical enough for students to connect back to their own interests and lives. Wherever possible, and especially in the videos that accompany the curriculum, we seek to **highlight a diverse array of role models** in terms of gender, race, and profession from which students can draw inspiration and “see themselves” participating in computing.

The curriculum assumes no prior knowledge of computing and is written to support *both* students and teachers who are new to the discipline. Activities are designed and structured in such a way that students with diverse learning needs have space to find their voice and to express their thoughts and opinions. The activities, videos, and computing tools in the curriculum strive to have a broad appeal and to be accessible to a student body diverse in background, gender, race, prior knowledge of computing, and personal interests.

Broadening student participation in computer science is a national goal, and effectively an issue of social justice. Motivational marketing messages only get you so far. We believe that the real key to attracting students to computer science and then sustaining that growth has as much to do with the teacher in the classroom as it does with anything else. The real “access” students need to computing is an **opportunity to legitimately and meaningfully participate in every lesson** regardless of the student’s background or prior experience in computing coming into the course. For example, the course begins with material that is challenging but typically unfamiliar even to students who have some prior experience or knowledge of computer science.

Who Should Take This Course?

There are no formal prerequisites for this course, though the College Board recommends that students have taken at least Algebra 1. The course requires a significant amount of expository writing (as well as writing computer code, of course). For students wishing to complete the requirements of the AP Exam and Performance Tasks, we recommend they be in 10th grade or above due the expectations of student responsibility and maturity for an AP course.

The curriculum itself does not assume any prior knowledge of computing concepts before entering the course. It is intended to be suitable as a **first course in computing** though students with a variety of backgrounds and prior experiences will also find the course engaging and with plenty of challenges. While it is increasingly likely that students entering this AP course in high school will have had *some* prior experience in computer science (particularly with programming), that experience is equally likely to be highly varied both in quantity and quality. It is for this reason that the course *does not* start with programming, but instead with material that is much more likely to put all students on a level playing field for the first few weeks of class. Read more about this below in the description of Unit 1.

Who Should Teach This Course?

The curriculum is designed so that a teacher who is new to teaching this material has adequate support and preparation - especially for those who go through Code.org's professional development program. A teacher who is motivated to teach a course like this, but who has limited technical or formal computer science experience should be able to be successful. At a minimum, we strongly recommend that the teacher have a reasonable level of comfort using computers (using the web, email, downloading and saving files, basic troubleshooting, etc.).

Teachers of this course should be especially interested in creating and nurturing equitable, engaging classrooms. The work of providing an equitable classroom doesn't start or stop with curriculum -- the classroom environment and teaching practices must also be structured such that all learners can access and engage with the material at a level that doesn't advantage some at the expense of others. **Equitable teaching practices** are inextricably linked and woven into the design and structure of our lessons, and in some cases the reason for their existence.

The curriculum provides a number of resources for the teacher, such as assessment support, computing tools that are designed for learning specific concepts, and the programming environment, App Lab. These resources have been specifically curated *for each step of each lesson*, which allows the teacher to act in the role of facilitator and coach when addressing unfamiliar material, rather than having to worry about presenting or lecturing.

Code.org CS Principles Course Snapshot

The chart on the following page is intended to show the big picture of the entire course.

- Each unit is broken into "chapters" - groups of lessons that address a related set of topics. Each chapter ends with some kind of assessment.
- Each chapter shows every lesson title along with suggested pacing guidance
- Note the *Performance Tasks* cluster of lessons at the end. They are not a formal unit of the course because work on the tasks can take place at various points throughout the course. However, you must plan to allocate time toward completing them.

CS Principles Course Snapshot

Unit 1 - The Internet

Ch. 1: Digital Information

- wk 1** Personal Innovations
Sending Binary Messages
Sending Messages with the Simulator
- 2** Number Systems
Binary Numbers
Sending Numbers
- 3** Encoding and Sending Formatted Text
Unit 1 Chapter 1 Assessment

Ch. 2: Inventing The Internet

- The Internet is for Everyone
- 4** The Need for Addressing
Routers and Redundancy
Packets and Making a Reliable Internet
- 5** The Need for DNS
HTTP and Abstraction
Practice PT - The Internet and Society
Unit 1 Chapter 2 Assessment

Unit 2 - Digital Information

Ch. 1: Encoding and Compressing Info

- wk 1** Bytes and File Sizes
Text Compression
Encoding B&W Images
- 2** Encoding Color Images
Lossy Compression and File Formats
- 3** Encode an Experience
Unit 2 Chapter 1 Assessment

Ch. 2 - Manipulating and Visualizing Data

- 4** Intro to Data
Finding Trends with Visualizations
Check Your Assumptions
Good and Bad Data Visualizations
- 5** Making Data Visualizations
Discover a Data Story
- 6** Cleaning Data
Creating Summary Tables
Practice PT - Tell a Data Story
Unit 2 Chapter 2 Assessment

Unit 3 - Intro to Programming

Ch. 1 - Programming Languages & Algorithms

- wk 1** The Need For Programming Languages
The Need for Algorithms
Creativity in Algorithms
- 2** Using Simple Commands
Creating Functions
Functions and Top-Down Design

Unit 3 -Intro to Programming (con'd)

- wk 3** APIs and Function Parameters
Creating functions with Parameters
Looping and Random Numbers
- 4** Practice PT - Design a Digital Scene
Unit 3 Chapter 1 Assessment

Unit 4 -Big Data and Privacy

Ch. 1: The World of Big Data and Encryption

- wk 1** What is Big Data?
Rapid Research - Data Innovations
Identifying People with Data
- 2** The Cost of Free
Simple Encryption
- 3** Encryption with Keys and Passwords
Public Key Crypto
Rapid Research - Cybercrime
- 4** Practice PT - Big Data and Security Dilemmas
Unit 4 Chapter 1 Assessment

Unit 5 -Building Apps

Ch. 1: Event-Driven Programming

- wk 1** Buttons and Events
Multi-screen Apps
Building an App: Multi-Screen App
- 2** Controlling Memory with Variables
Building an App: Clicker Game
Unit 5 Assessment 1
User Input and Strings
- 3** "If" Statements Unplugged
Boolean Expressions and "If" Statements
- 4** "if-else-if" and Conditional Logic
Building an App: Color Sleuth
Unit 5 Assessment 2

Ch. 2: Programming with Data Structures

- 5** While Loops
Loops and Simulations
Introduction to Arrays
- 6** Building an App: Image Scroller
Unit 5 Assessment 3
Processing Arrays
Functions with Return Values
- 7** Building an App: Canvas Painter
Unit 5 Assessment 4
Practice PT: Create
Unit 5 Assessment 5 - AP Pseudocode Practice

Performance Tasks

- 1 hr** Tech Setup (Can be completed at any time)
- 10 hrs** Explore prep (Can be completed after Unit 4)
Explore PT (8 class hours)
- 14 hrs** Create Prep(Can be completed after Unit 5 Chapter 1)
Create PT (12 class hours)

While the layout of units appears to be modular, the units of study are scaffolded **and sequenced to build students' skills and knowledge toward the Enduring Understandings of the CSP Course Framework**. The lessons for each unit assume that students have the knowledge and skills obtained in the previous units. There are also many thematic connections that can be made between and among lessons and units.

Each **unit** attempts to “tell a story” about a particular topic in computing from a more primitive beginning to a more complex end. The lessons in each unit are grouped into one or two **chapters** of a few weeks worth of lessons whose content is related or connected in some way. The course snapshot on the first page shows the chapters for each unit. Each **lesson** is intended to be a complete thought that takes the student from some motivational question or premise to an activity that builds skills and knowledge toward some learning objective(s).

Each unit contains at least one summative assessment, project, or Practice PT that asks students to complete tasks similar to the official PTs. Sometimes these come mid-unit, and sometimes they come closer to the end.

Technical and Material Requirements

The course requires and assumes a 1:1 computer lab or setup such that each student in the class 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 modern web browser. For more details on the technical requirements, please visit: code.org/educate/it

While the course features many “unplugged” activities designed to be completed away from the computer, daily access to a computer is essential for every student. It is not required that students have access to internet-connected computers at home to teach this course. But because almost all of the materials are online, it is certainly an advantage. PDFs of handouts, worksheets and readings are available on the course website.

Computational Tools, Resources and Materials Provided

The Code.org CSP curriculum includes almost all resources teachers need to teach the course including:

Lesson Plans

- Instructional guides for every lesson
- Activity Guides and handouts for students
- Formative and summative assessments
- Exemplars, rubrics, and teacher dashboard

Videos

- **Student videos** - including tutorials, instructional and inspirational videos
- **Teacher videos** - including lesson supports and pedagogical tips and tricks

Computational Tools

- **Widgets** for exploring individual computing concepts
- **Internet Simulator** - Code.org's tool for investigating the various “layers” of the internet
- **App Lab** - Code.org's JavaScript programming environment for making apps

Required Materials / Supplies:

One potentially significant cost to consider 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, a some lessons call for typical classroom supplies and manipulatives such as:

- Student Journal
- Poster paper
- Markers
- Post-it notes
- Graph paper, etc.

There is a complete materials list in the curriculum front matter. Besides printing costs, all other materials are highly suggested, and are low cost (cups, string, playing cards, etc.).

Suggested Text

Blown to Bits <http://www.bitsbook.com/>

This course does not require or follow a textbook. *Blown to Bits* is a book that can online **free of cost**. Many of its chapters are excellent supplemental reading for especially for material in Units 1, 2 and 4. We refer to chapters as supplemental lesson plans as appropriate.



be accessed
our course,
reading in

AP[®] Assessment

The AP Assessment consists of a 74-question multiple choice exam and two “through-course” assessments called the *AP Performance Tasks* (PTs). The tasks can be found in the official [AP CS Principles Exam and Course Description](#).

- Create Performance Task (p. 108)
- Explore Performance Task (p. 111)

Coverage of the AP CS Principles Framework and Computational Thinking Practices

The [CS Principles Framework](#) outlines seven “Big Ideas” of computing, and six “Computational Thinking Practices”. Activities in the course should ensure that students are engaging in the Computational Thinking Practices while investigating the Big Ideas.

Seven Big Ideas

The course is organized around seven big ideas, which encompass ideas foundational to studying computer science.

- Big Idea 1: Creativity
- Big Idea 2: Abstraction
- Big Idea 3: Data
- Big Idea 4: Algorithms
- Big Idea 5: Programming
- Big Idea 6: The Internet
- Big Idea 7: Global Impacts

Six Computational Thinking Practices

Computational thinking practices capture important aspects of the work that computer scientists engage in.

- P1: Connecting Computing
- P2: Creating Computational Artifacts
- P3: Abstracting
- P4: Analyzing Problems and Artifacts
- P5: Communicating
- P6: Collaborating

These **Big Ideas** and **Practices** are not intended to be taught in any particular order, nor are they units of study in and of themselves. The *Big Ideas* overlap, intersect, and reference each other. The *practices* represent higher order thinking skills, behaviors, and habits of mind that need to be constantly visited, repeatedly honed, and refined over time.

Unit Overviews

On the pages that follow are more in-depth prose descriptions of each unit of study which explain the topics covered, what students will be doing and how the lessons *build toward the Enduring Understandings* in the framework.

We also show how each lesson maps to the CSP Framework for the *Enduring Understandings* as well as the specific *Learning Objective* and *Essential Knowledge* statements. For example you might see a table like:

Unit 1: The Internet	← Unit Title
Ch 1: Representing Information	← Chapter Title
...	
2. Sending Binary Messages	← Lesson Name
2.1 2.1.1[P3] (ABCE)	← EU LO [P] (EKs)
2.1 2.1.2[P5] (DEF)	↳ EK = <u>E</u> ssential <u>K</u> nowledge statements
3.3 3.3.1[P4] (AB)	↳ [P] = Computational Thinking <u>P</u> ractice
	↳ LO = <u>L</u> earning <u>O</u> bjective
	↳ EU = <u>E</u> nduring <u>U</u> nderstanding

See the CSP Framework document for listings of all the Enduring Understandings.

Each unit also highlights a particular lesson, project or assignment of interest, explaining what students do and showing which learning objectives and computational thinking practices that particular assignment addresses.

Unit 1: The Internet

This unit explores the technical challenges and questions that arise from the need to represent digital information in computers and transfer it between people and computational devices. Topics include: the digital representation of information - especially, numbers, text, and communication protocols. The first unit of this course purposefully addresses material that is fundamental to computing but with which many students, even those with computers at home or who have some prior experience with programming, are unfamiliar. This levels the playing field for participation and engagement right from the beginning of the course.

Chapter 1 of the unit begins with a consideration of what is involved in sending a single bit of information from one place to another. In the *Sending Binary Messages* lesson students work with a partner to invent and build their own bit-sending “device.” Complexity increases as students adapt their machines to handle multi-bit messages and increasingly complex information. Students use an Internet Simulator that allows them to develop and test binary encodings and communication protocols of their own invention. These should be an illustrative set of activities that helps build toward the enduring understandings that: **A variety of abstractions built upon binary sequences can be used to represent all digital data (2.1)** and that **characteristics of the Internet influence the systems built on it (6.2)**.

Chapter 2 of the unit is called “Inventing the Internet” because through the unit students continue to solve problems similar ones that had to be solved to build the real Internet. Students design their own versions of protocols, each one layered on the previous one, in a process that mimics the layered sets of protocols on the real Internet and builds toward the enduring understandings that **The Internet is a network of autonomous systems (6.1)** and that **How the Internet was designed (layers of protocols) affects the systems built on top of it (6.2)** as well as its ability to grow and adapt.

For the practice Performance Task at the end of the unit students research a modern societal issue related to the Internet such as “Net Neutrality” or internet censorship, which layers in enduring understandings about the fact that **computing has a global affect -- both beneficial and harmful -- on people and society (7.3)**.

Unit 1: Practice PT Highlight

Practice PT: The Internet and Society

Students will research and prepare a flash talk about a social issue related to the Internet. Students pick one of: Net Neutrality, Internet Censorship, or Computer/Network Surveillance. This lesson is good practice for certain elements of the Explore Performance Task, which students will complete later in the school year. Students will do a bit of research about impacts of the Internet, explain some technical details related to ideas in computer science, and connect these ideas to global and social impacts. Students will practice synthesizing information, and presenting their learning in a flash talk.

Learning Objectives Addressed:

Internet: 6.1.1[P3], 6.2.2[P4], 6.3.1 [P1]

Global Impacts: 7.1.1 [P4], 7.3.1 [P4], 7.4.1 [P1], 7.5.2 [P5]

Computational Thinking Practices Emphasized:

P1: Connecting Computing

P5: Communicating

Unit 1: The Internet

Ch 1: Representing and Transmitting Information

1. Personal Innovations

- 7.1 7.1.1[P4] (ABCDEFGHJKLMNO)
- 7.2 7.2.1[P1] (ABCG)
- 7.3 7.3.1[P4] (ABCDEFGHJKLMNO)
- 7.4 7.4.1[P1] (ABCD)

2. Sending Binary Messages

- 2.1 2.1.1[P3] (ABCE)
- 2.1 2.1.2[P5] (DEF)
- 3.3 3.3.1[P4] (AB)

3. Sending Binary Messages with the Internet Simulator

- 2.1 2.1.1[P3] (ABCE)
- 2.1 2.1.2[P5] (DEF)
- 2.3 2.3.1[P3] (ABCD)
- 2.3 2.3.2[P3] (A)
- 3.1 3.1.3[P5] (A)
- 3.3 3.3.1[P4] (AB)
- 6.1 6.1.1[P3] (AC)
- 6.2 6.2.2[P4] (DJK)

4. Number Systems

- 2.1 2.1.1[P3] (ABCDE)
- 2.3 2.3.1[P3] (AB)
- 2.3 2.3.2[P3] (ABC)

5. Binary Numbers

- 2.1 2.1.1[P3] (ABCDEG)
- 2.1 2.1.2[P5] (ABCDEF)
- 2.3 2.3.1[P3] (ABCD)
- 2.3 2.3.2[P3] (ABCDE)
- 3.1 3.1.3[P5] (AB)

6. Sending Numbers

- 2.1 2.1.1[P3] (ABCDEG)
- 2.1 2.1.2[P5] (ABCDEF)
- 2.3 2.3.1[P3] (ABCD)
- 2.3 2.3.2[P3] (ABCDE)
- 3.1 3.1.3[P5] (ABDE)
- 6.2 6.2.2[P4] (DGH)

7. Encoding and Sending Formatted Text

- 2.1 2.1.1[P3] (ABCDE)
- 2.1 2.1.2[P5] (BDEF)
- 2.2 2.2.1[P2] (AB)
- 3.1 3.1.3[P5] (AE)
- 3.3 3.3.1[P4] (AB)
- 6.1 6.1.1[P3] (ABCD)
- 6.2 6.2.2[P4] (DFGH)

Ch. 2: Inventing the Internet

8. The Internet Is for Everyone

- 6.1 6.1.1[P3] (BCE)
- 6.2 6.2.2[P4] (E)
- 7.3 7.3.1[P4] (ADEGL)
- 7.4 7.4.1[P1] (CDE)

9. The Need for Addressing

- 6.1 6.1.1[P3] (CDFH)
- 6.2 6.2.1[P5] (C)
- 6.2 6.2.2[P4] (D)
- 6.3 6.3.1[P1] (A)

10. Routers and Redundancy

- 3.3 3.3.1[P4] (AF)
- 6.1 6.1.1[P3] (BCE)
- 6.2 6.2.1[P5] (AD)
- 6.2 6.2.2[P4] (B)

11. Packets and Making a Reliable Internet

- 6.2 6.2.1[P5] (AD)
- 6.2 6.2.2[P4] (ABG)

12. The Need for DNS

- 6.1 6.1.1[P3] (G)
- 6.2 6.2.1[P5] (B)
- 6.2 6.2.2[P4] (CD)

13. HTTP and Abstraction on the Internet

- 6.1 6.1.1[P3] (I)
- 6.2 6.2.2[P4] (H)

14. Practice PT - The Internet and Society

- 6.3 6.3.1[P1] (AB)
- 7.1 7.1.1[P4] (ABCDHIJKMO)
- 7.4 7.4.1[P1] (ABDE)

Unit 2: Digital Information

This unit further explores the ways that digital information is encoded, represented and manipulated. In this unit students will use a variety of tools including Code.org widgets and external data manipulation and visualization tools (such as Excel or Google Sheets).

The unit begins by continuing to look at the possibilities and limitations of encoding information in binary and building on the enduring understanding that **there are trade offs when representing information as digital data (3.3)**. Students create an image that they encode with binary by hand, and also look at a variety of data compression techniques for text and images. The *Practice PT: Encode an Experience* has students devise their own completely new data encoding scheme for something complex like a human experience which has students deeply consider how a **variety of abstractions built upon binary sequences can be used to represent all digital data (2.1)**.

In the second chapter students develop skills interpreting visual data and using spreadsheet and visualization tools to create their own digital artifacts. Through an ongoing project - the “class data tracker” - students learn how to collect and clean data, and to use a few common tools for computing aggregations and creating visualizations. These activities build toward the enduring understandings that **people use computer programs to process information to gain insight and knowledge (3.1)** and that **computing facilitates exploration and the discovery of connections in information (3.2)**.

As students explore ways in which **computing enhances communication, interaction, and cognition (7.1)**, they also examine how human error during data analysis can lead to inaccurate and potentially damaging conclusions. This leads to a deeper understanding that there are **trade offs**, and potentially **beneficial and harmful effects when representing information as digital data (3.3, 7.3)**.

Unit 2: Practice PT Highlights

Practice PT: Encode an Experience

Students invent a binary encoding (file format) for a real life experience. How might you encode a birthday party? or a soccer game? or the brush strokes of a real painting? Students come up with their own creation and present their work in a format similar to that of a Performance Task. This assignment emphasizes the writing process, and giving and incorporating feedback from peers.

Learning Objectives Addressed:

Creativity: 1.1.1 [P2], 1.2.4 [P6]

Abstraction: 2.1.1 [P3], 2.1.2 [P5], 2.2.1 [P2]

Data: 3.2.1 [P1], 3.3.1 [P4]

Computational Thinking Practices Emphasized:

P1: Connecting Computing **P3:** Abstracting **P5:** Communicating **P6:** Collaborating

Practice PT: Tell a Data Story

This small project culminates a series of lessons in which students must use digital tools to collaboratively investigate a data set to discover possible connections and trends. Students must produce a visual explanation of their findings in the data and write a short piece about what the data shows.

Learning Objectives Addressed:

Creativity: 1.1.1 [P2], 1.2.1 [P2], 1.2.4 [P6], 1.2.5 [P4]

Data: 3.1.1 [P4], 3.1.2 [P6], 3.1.3 [P5], 3.2.1 [P1]

Global Impacts: 7.1.1 [P4], 7.4.1 [P1]

Computational Practices Emphasized: P1:

Connecting Computing **P2:** Creating Computational Artifacts **P5:** Communicating **P6:** Collaborating

Unit 2: Digital Information

Ch. 1: Encoding and Compressing Complex Information

1. Bytes and File Sizes

- 2.1 2.1.1[P3] (BC)
- 2.1 2.1.2[P5] (BCEF)
- 3.3 3.3.1[P4] (G)

2. Text Compression

- 2.1 2.1.1[P3] (ABC)
- 2.2 2.2.1[P2] (B)
- 3.1 3.1.1[P4] (ADE)
- 3.1 3.1.2[P6] (ABCD)
- 3.1 3.1.3[P5] (AE)
- 3.3 3.3.1[P4] (A)
- 4.2 4.2.1[P1] (ABCD)
- 4.2 4.2.2[P1] (BC)
- 4.2 4.2.3[P1] (ABC)
- 4.2 4.2.4[P4] (ACD)

3. Encoding B&W Images

- 1.1 1.1.1[P2] (AB)
- 1.2 1.2.1[P2] (A)
- 1.3 1.3.1[P2] (C)
- 2.1 2.1.1[P3] (ABC)
- 2.1 2.1.2[P5] (AB)
- 2.3 2.3.1[P3] (ABCD)
- 3.1 3.1.1[P4] (ADE)
- 3.1 3.1.2[P6] (ABCD)
- 3.1 3.1.3[P5] (AE)
- 3.2 3.2.1[P1] (GHI)
- 3.3 3.3.1[P4] (A)

4. Encoding Color Images

- 1.1 1.1.1[P2] (AB)
- 1.2 1.2.1[P2] (A)
- 1.3 1.3.1[P2] (C)
- 2.1 2.1.1[P3] (ABCDF)
- 2.1 2.1.2[P5] (DEF)
- 2.2 2.2.1[P2] (AB)
- 2.3 2.3.1[P3] (ABCD)
- 3.1 3.1.1[P4] (ADE)
- 3.1 3.1.2[P6] (ABCD)
- 3.1 3.1.3[P5] (AE)
- 3.2 3.2.1[P1] (GHI)

5. Lossy Compression and File Formats

- 3.3 3.3.1[P4] (ACDEG)

6. Practice PT - Encode an Experience

- 2.1 2.1.1[P3] (ABCDE)
- 2.1 2.1.2[P5] (ABDF)
- 2.2 2.2.1[P2] (AB)

Ch. 2: Manipulating and Visualizing Data

7. Introduction to Data

- 2.2 2.2.3[P3] (E)
- 3.1 3.1.1[P4] (BCDE)
- 3.1 3.1.3[P5] (D)
- 3.2 3.2.1[P1] (ABC)
- 7.3 7.3.1[P4] (HJ)

8. Finding Trends with Visualizations

- 3.1 3.1.1[P4] (ABE)
- 3.1 3.1.2[P6] (ABCDEF)
- 3.1 3.1.3[P5] (ABCE)
- 3.2 3.2.1[P1] (ABCDE)

9. Check Your Assumptions

- 3.1 3.1.1[P4] (E)
- 3.1 3.1.2[P6] (ABCDF)
- 3.2 3.2.1[P1] (ABC)
- 7.4 7.4.1[P1] (ABCD)

10. Good and Bad Data Visualizations

- 1.2 1.2.5[P4] (ABCD)
- 3.1 3.1.1[P4] (DE)
- 3.1 3.1.2[P6] (ABCDF)
- 3.1 3.1.3[P5] (ABCDE)

11. Making Data Visualizations

- 1.2 1.2.5[P4] (ABCD)
- 3.1 3.1.1[P4] (DE)
- 3.1 3.1.2[P6] (ABCDF)
- 3.1 3.1.3[P5] (ABCDE)

12. Discover a Data Story

- 1.1 1.1.1[P2] (AB)
- 1.2 1.2.1[P2] (ABC)
- 1.2 1.2.2[P2] (A)
- 1.2 1.2.4[P6] (A)
- 1.2 1.2.5[P4] (D)
- 1.3 1.3.1[P2] (E)
- 3.1 3.1.1[P4] (DE)
- 3.1 3.1.2[P6] (ABCDF)
- 3.1 3.1.3[P5] (ABCD)

13. Cleaning Data

- 1.2 1.2.1[P2] (ABCDE)
- 1.2 1.2.4[P6] (AB)
- 3.1 3.1.1[P4] (AB)
- 3.1 3.1.2[P6] (ABCDEF)
- 3.2 3.2.1[P1] (ABCDEF)
- 3.2 3.2.2[P3] (BCG)
- 7.1 7.1.2[P4] (CD)

14. Creating Summary Tables

- 1.1 1.1.1[P2] (AB)
- 1.2 1.2.1[P2] (ABE)
- 1.2 1.2.4[P6] (AB)
- 3.1 3.1.1[P4] (ABCDE)
- 3.1 3.1.2[P6] (DEF)
- 3.1 3.1.3[P5] (ABCD)
- 3.2 3.2.1[P1] (CF)

15. Practice PT - Tell a Data Story

- 1.2 1.2.1[P2] (ABCE)
- 1.2 1.2.2[P2] (AB)
- 1.2 1.2.5[P4] (ABCD)
- 3.1 3.1.3[P5] (ABCD)
- 7.3 7.3.1[P4] (J)

Unit 3: Algorithms and Programming

This unit introduces students to programming in the JavaScript language and creating small applications (apps) that live on the web. This introduction places a heavy emphasis on understanding general principles of computer programming and revealing those things that are universally applicable to any programming language.

To start the unit we use unplugged activities to introduce algorithms and highlight the need for a programming language to implement them on a computer. These activities will involve the whole class working in small groups to solve problems using simple manipulatives like playing cards or blocks. We want to draw connections here between the rules of Internet protocols developed earlier in the course, in which students acted as the computer processing the information. Many of the structured and systematic thinking that goes into developing communication protocols feels similar to designing algorithms - ultimately you're designing a series of steps to solve a problem that a machine could follow. We want to establish the dual enduring understandings that **algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages (4.1)** and **people write programs to execute algorithms (5.2)**.

Students are introduced to the App Lab programming environment by writing programs to control a “turtle”, an imaginary character that moves around the screen and can draw. In the lessons students learn features of the JavaScript language by going through a series of short tutorials to familiarize students with the environment, and new concepts. There is a heavy emphasis on writing procedures (functions in JavaScript), and using top-down program design - a process by which a large problem is broken down into smaller and more manageable parts. These lessons highlight the way **multiple levels of abstraction are used to write programs (2.2)**.

Along the way students create more and more sophisticated drawings culminating in the **Practice PT: Design a Digital Scene** in which small groups must collaborate to design and share code to create a small vignette created with turtle art. Through the lessons and PTs we want to build toward some enduring understandings that **creative development can be an essential process for creating computational artifacts (1.1)** and that collaboration and **computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem (1.2)**.

Unit 3: Practice PT Highlight

Practice PT: Design a Digital Scene

In this project students work with a small team to create a digital scene with turtle graphics. They plan the scene together, code the parts separately and bring them together to make a whole. An important focus of this project is on how teams of programmers work together. Students reflect on their experience in a way that is similar to the *Create* performance task. A heavy programming emphasis is on writing functions (procedures) that can be easily incorporated into others' code.

Learning Objectives Addressed:

Creativity: 1.1.1 [P2], 1.2.1 [P2], 1.2.4 [P6], 1.3.1 [P2]

Abstraction: 2.2.1 [P2], 2.2.2 [P3]

Algorithms: 4.1.1 [P2]

Programming: 5.1.1 [P2], 5.1.3 [P6], 5.3.1 [P3]

Computational Practices Emphasized:

P2: Creating Computational Artifacts **P3:**

Abstracting P6: Collaborating

Unit 3: Algorithms and Programming

Ch. 1: Programming Languages and Algorithms

1. The Need for Programming Languages

- 4.1 4.1.2[P5] (ABCFI)
- 5.2 5.2.1[P3] (E)

2. The Need for Algorithms

- 4.1 4.1.1[P2] (ABCDHI)
- 4.1 4.1.2[P5] (ABG)
- 4.2 4.2.4[P4] (F)
- 5.4 5.4.1[P4] (FI)

3. Creativity in Algorithms

- 2.2 2.2.3[P3] (C)
- 4.1 4.1.1[P2] (ABCDEFGHI)
- 5.2 5.2.1[P3] (ABDEJ)

4. Using Simple Commands

- 5.1 5.1.2[P2] (AI)
- 5.1 5.1.3[P6] (BCDF)
- 5.2 5.2.1[P3] (AB)
- 5.4 5.4.1[P4] (ADEI)

5. Creating Functions

- 2.2 2.2.1[P2] (AB)
- 2.2 2.2.2[P3] (AB)
- 5.3 5.3.1[P3] (ABCL)
- 5.4 5.4.1[P4] (ABCD)

6. Functions and Top-Down Design

- 2.2 2.2.1[P2] (A)
- 2.2 2.2.2[P3] (AB)
- 2.2 2.2.3[P3] (AD)
- 5.1 5.1.2[P2] (ABC)
- 5.1 5.1.3[P6] (ABCDEF)
- 5.3 5.3.1[P3] (ABCD)

7. APIs and Using Functions with Parameters

- 2.2 2.2.2[P3] (AB)
- 2.2 2.2.3[P3] (AB)
- 5.1 5.1.2[P2] (DEF)
- 5.3 5.3.1[P3] (DEFGMNO)

8. Creating Functions with Parameters

- 2.2 2.2.1[P2] (C)
- 2.2 2.2.2[P3] (AB)
- 5.3 5.3.1[P3] (ACDFGL)
- 5.4 5.4.1[P4] (CDEFGHIJK)

9. Looping and Random Numbers

- 4.1 4.1.1[P2] (D)
- 5.1 5.1.2[P2] (BC)
- 5.3 5.3.1[P3] (ACDFGL)
- 5.4 5.4.1[P4] (CDEFGHIJK)

10. Practice PT - Design a Digital Scene

- 2.2 2.2.1[P2] (C)
- 2.2 2.2.2[P3] (AB)
- 2.2 2.2.3[P3] (AB)
- 4.1 4.1.1[P2] (D)
- 5.1 5.1.2[P2] (BC)
- 5.1 5.1.3[P6] (ABCDEF)
- 5.3 5.3.1[P3] (ACDFGL)
- 5.4 5.4.1[P4] (CDEFGHIJK)

Unit 4: Big Data and Privacy

The data rich world we live in also introduces many complex questions related to public policy, law, ethics and societal impact. In many ways this unit acts as a unit on current events. It is highly likely that there will be something related to big data, privacy and security going on in the news at any point in time. The major goals of the unit are 1) for students to develop a well-rounded and balanced view about data in the world around them and both the positive and negative effects of it and 2) to understand the basics of how and why modern encryption works.

During the first two weeks of the unit students will research and discuss **innovations enabled by computing in a wide variety of fields (7.2)**. During this time views about the benefits - “Big Data is great!” - and drawbacks - “Big Data is scary!” will swing quickly. We primarily want to build toward the dual enduring understandings that **Computing facilitates exploration and the discovery of connections in information (3.2)** and that **Computing innovations influence and are influenced by the economic, social, and cultural contexts in which they are designed and used (7.4)** while the **beneficial and harmful effects (7.3)** of these things must be weighed and kept in balance.

The activities in the third week around data encryption follow a pattern: introduce an encryption concept through an unplugged activity or thinking prompt, and then “plug it in” by using a Code.org widget to explore the concept further. The purpose of the widgets is to allow students time to play with some of the ideas - often mathematical in nature - underlying different methods of encryption and why they might be susceptible to being “cracked.” These explorations lead towards an understanding of computationally hard problems and the fact that **algorithms can solve many but not all computational problems (4.2)**.

In particular students should come away with a high level understanding of how asymmetric encryption works and why it makes certain things possible (sending encrypted data without a shared key) and certain things basically impossible (cracking a key). By investigating some of the mathematical foundations of encryption we build toward the enduring understanding that **cybersecurity is an important concern for the Internet and the systems built on it (6.3)** and as always **There are trade offs when representing information as digital data (3.3)**.

Unit 4 Practice PT Highlights

Practice PT: Big Data and Cybersecurity Dilemmas

Students will complete a research project on a current issue or event related to Big Data, security and encryption. Students will need to identify appropriate online resources to learn about the issues and find good artifacts to include with their findings. The written components and audio / visual artifact students will identify are similar to those students will see in the AP Performance Tasks.

Learning Objectives Addressed:

Data: 3.3.1 [P4]

Internet: 6.1.1 [P3], 6.2.1 [P5], 6.2.2 [P4], 6.3.1 [P1]

Global Impacts: 7.1.1 [P4], 7.3.1 [P4], 7.5.1 [P1], 7.5.2 [P5]

Computational Thinking Practices Emphasized:

P1: Connecting Computing

P5: Communicating

Unit 4: Big Data and Privacy

Ch. 1: The World of Big Data and Encryption

1. What is Big Data?

- 3.2 3.2.2[P3] (ABCDEFGH)
- 7.2 7.2.1[P1] (ABCDEFGH)
- 7.5 7.5.2[P5] (AB)

2. Rapid Research - Data Innovations

- 1.2 1.2.3[P2] (C)
- 1.2 1.2.5[P4] (AD)
- 3.2 3.2.2[P3] (ABCDEFGH)
- 7.1 7.1.1[P4] (DEFGHIJKLMNO)
- 7.4 7.4.1[P1] (ABCDE)
- 7.5 7.5.2[P5] (AB)

3. Identifying People With Data

- 3.2 3.2.2[P3] (D)
- 3.3 3.3.1[P4] (BF)
- 7.3 7.3.1[P4] (GJKL)

4. The Cost of Free

- 3.3 3.3.1[P4] (ABF)
- 7.3 7.3.1[P4] (AGHJKLMN)

5. Simple Encryption

- 1.2 1.2.2[P2] (A)
- 3.3 3.3.1[P4] (BF)
- 6.3 6.3.1[P1] (CHIK)
- 7.3 7.3.1[P4] (G)

6. Encryption with Keys and Passwords

- 2.3 2.3.2[P3] (A)
- 3.1 3.1.1[P4] (A)
- 4.2 4.2.1[P1] (ABCD)
- 6.3 6.3.1[P1] (CHIJK)

7. Public Key Cryptography

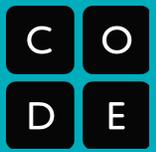
- 4.2 4.2.1[P1] (ABCD)
- 4.2 4.2.2[P1] (A)
- 4.2 4.2.3[P1] (A)
- 4.2 4.2.4[P4] (ABC)
- 6.3 6.3.1[P1] (HIL)

8. Rapid Research - Cybercrime

- 6.2 6.2.2[P4] (H)
- 6.3 6.3.1[P1] (CDEFGH)
- 7.3 7.3.1[P4] (G)

9. Practice PT - Big Data and Cybersecurity Dilemmas

- 1.1 1.1.1[P2] (AB)
- 1.2 1.2.1[P2] (ABCE)
- 1.2 1.2.2[P2] (A)
- 1.2 1.2.5[P4] (B)
- 6.3 6.3.1[P1] (ABCDEFGHIJKLM)
- 7.3 7.3.1[P4] (ADGHL)
- 7.4 7.4.1[P1] (ABE)



Unit 5: Building Apps

This unit continues to develop students' ability to program in the JavaScript language, using Code.org's App Lab environment to create a series of small applications (apps) that live on the web, each highlighting a core concept of programming. In this unit students transition to creating event-driven apps. The unit assumes that students have learned the concepts and skills from Unit 3, namely: writing and using functions, using simple repeat loops, being able to read documentation, collaborating, and using the Code Studio environment with App Lab.

The first chapter begins by introducing App Lab's "Design Mode" which allows students to rapidly prototype an app. Again, we want to highlight the enduring understanding that **Computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem (1.2)**. As students construct simple apps that respond to user actions and inputs, the lessons progress through some core concepts of programming; Specifically, the lessons cover variables, boolean logic and conditionals, which enforces the understanding that **Programming uses mathematical and logical concepts (5.5)**.

The second chapter goes deeper into core programming concepts including looping, arrays, and the **use of models and simulation to develop new insight and knowledge (2.3)**. We want to reinforce the idea that **programs are developed, maintained, and used by people for different purposes (5.4)**. Each app also emphasizes a different core concept and skill of programming allowing us to further the connections that **people write programs to execute algorithms (5.2)** and that **programs employ appropriate abstractions (5.3)** (such as list structures) as well as **mathematical and logical concepts (5.5)** to **extend traditional forms of human expression and experience (1.3)**.

The Practice PT: *Create Your Own App* asks students to look back at the apps they've created during the unit and use one as a point of inspiration for creating their own app. This project is designed to highlight the way **programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (5.1)** and is designed to closely mirror the actual Create PT which students will complete in the following unit.

Unit 5: Practice PT Highlight

Practice PT: Create Your Own App

Students will design an app based off of one they have previously worked on in the programming unit. Students will choose the kinds of improvements they wish to make to a past project in order to show their ability to add new abstractions (procedures and functions) and algorithms to an existing program. The project concludes with reflection questions similar to those students will see on the AP Create Performance Task.

Learning Objectives Addressed:

Creativity: 1.1.1 [P2], 1.2.1 [P2], 1.2.2 [P2], 1.2.3 [P2], 1.2.4 [P6], 1.3.1 [P2]

Abstraction: 2.2.1 [P2], 2.2.2 [P3]

Algorithms: 4.1.1 [P2], 4.1.2 [P5]

Programming: 5.1.1 [P2], 5.1.2 [P2], 5.1.3 [P6], 5.2.1 [P3], 5.3.1 [P3], 5.4.1 [P4], 5.5.1 [P1]

Computational Practices Emphasized:

P2: Creating Computational Artifacts **P3:**

Abstracting **P5:** Communicating **P6:** Collaborating

Unit 5: Building Apps

Ch. 1: Event-Driven Programming

1. Introduction to Event-Driven Programming

1.1	1.1.1[P2] (B)	5.1	5.1.2[P2] (J)
1.2	1.2.1[P2] (ABE)	5.4	5.4.1[P4] (ACDEFHKM)
5.1	5.1.1[P2] (AB)		

2. Multi-Screen Apps

1.1	1.1.1[P2] (B)	5.1	5.1.1[P2] (BC)
1.2	1.2.1[P2] (A)	5.1	5.1.2[P2] (J)
1.2	1.2.3[P2] (A)	5.1	5.1.3[P6] (D)
1.2	1.2.4[P6] (ADE)	5.4	5.4.1[P4] (CEFM)

3. Building an App: Multi-Screen App

1.1	1.1.1[P2] (AB)	5.1	5.1.2[P2] (ABCDEFJ)
1.2	1.2.1[P2] (ABCE)	5.1	5.1.3[P6] (BCD)
5.1	5.1.1[P2] (AB)	5.4	5.4.1[P4] (EFGJL)

4. Controlling Memory with Variables

5.2	5.2.1[P3] (CF)
-----	----------------

5. User Input and Strings

5.1	5.1.1[P2] (B)
5.3	5.3.1[P3] (L)

6. 'if-statements unplugged

4.1	4.1.1[P2] (AC)
4.1	4.1.2[P5] (ABG)
5.2	5.2.1[P3] (ABD)

7. Boolean Expressions and 'if' Statements

4.1	4.1.1[P2] (ABC)
5.5	5.5.1[P1] (EG)

8. 'if-else-if' and Conditional Logic

4.1	4.1.1[P2] (C)
5.1	5.1.2[P2] (J)
5.5	5.5.1[P1] (EG)

9. Building an App: Color Sleuth

1.1	1.1.1[P2] (B)	4.1	4.1.2[P5] (G)
1.2	1.2.4[P6] (ACDF)	5.1	5.1.2[P2] (A)
3.1	3.1.2[P6] (B)	5.1	5.1.3[P6] (ABCDEF)
4.1	4.1.1[P2] (AC)	5.5	5.5.1[P1] (DE)

Ch. 2: Programming with Data Structures

10. While Loops

3.1	3.1.1[P4] (A)	5.2	5.2.1[P3] (ABCDIJK)
4.1	4.1.1[P2] (ABCDH)	5.4	5.4.1[P4] (BEFGHKLM)
4.1	4.1.2[P5] (ABCDEFG)	5.5	5.5.1[P1] (DEFG)

11. Loops and Simulations

2.3	2.3.1[P3] (ACD)	4.1	4.1.2[P5] (AB)
2.3	2.3.2[P3] (ABDEFGH)	5.1	5.1.1[P2] (A)

12. Introduction to Arrays

1.1	1.1.1[P2] (B)	5.3	5.3.1[P3] (ABCKL)
5.1	5.1.2[P2] (A)	5.5	5.5.1[P1] (HIJ)

13. Building an App: Image Scroller

5.1	5.1.1[P2] (A)	5.4	5.4.1[P4] (BCGM)
5.2	5.2.1[P3] (EFIJK)	5.5	5.5.1[P1] (HIJ)
5.3	5.3.1[P3] (ABCDGKL)		

14. Processing Arrays

1.2	1.2.3[P2] (A)	5.1	5.1.2[P2] (AB)
4.1	4.1.1[P2] (ABCD)	5.3	5.3.1[P3] (KL)
4.2	4.2.4[P4] (DEFH)	5.5	5.5.1[P1] (EJ)

15. Functions with Return Values

1.1	1.1.1[P2] (B)	4.1	4.1.1[P2] (ABCDEF)
1.2	1.2.3[P2] (AC)	5.1	5.1.2[P2] (ABCJ)
2.2	2.2.1[P2] (AC)	5.3	5.3.1[P3] (ABCDEFGKL)
2.2	2.2.2[P3] (AB)	5.5	5.5.1[P1] (EJ)

16. Building an App: Canvas Painter

1.2	1.2.1[P2] (ABCD)	5.1	5.1.1[P2] (ABCDE)
1.3	1.3.1[P2] (CDE)	5.1	5.1.2[P2] (ABCJ)
2.2	2.2.1[P2] (ABC)	5.3	5.3.1[P3] (ABCDEFJKL)
2.2	2.2.2[P3] (AB)	5.4	5.4.1[P4] (ABCDEFHLMN)
4.1	4.1.1[P2] (ABCD)	5.5	5.5.1[P1] (DEFGHIJ)
4.1	4.1.2[P5] (ABCGI)		

17. Practice PT - Create Your Own App

1.1	1.1.1[P2] (AB)	4.1	4.1.1[P2] (ABCDEFGHI)
1.2	1.2.1[P2] (ABCDE)	4.1	4.1.2[P5] (ABCDEFGHI)
1.2	1.2.2[P2] (AB)	5.1	5.1.1[P2] (ABCDE)
1.2	1.2.3[P2] (ABC)	5.1	5.1.2[P2] (ABCDEFGHIJ)
1.2	1.2.4[P6] (ABCDEF)	5.1	5.1.3[P6] (ABCDEF)
2.2	2.2.1[P2] (ABC)	5.4	5.4.1[P4] (CEFGHJLMN)
2.2	2.2.2[P3] (AB)	5.5	5.5.1[P1] (ABCDEFGHIJ)

Performance Tasks

In Units 1-5 students learn the content and practice the skills they need in order to succeed on the AP CSP Performance Tasks. Still, a certain level of guidance during the PT development process is not only recommended, but vital. For example, coaching students early on helps them clarify their ideas and/or approaches to the PTs especially related to **finding, evaluating and citing sources (7.5)**.

The curriculum provides a few lessons to help instructors and students prepare and make a plan for completing the AP Performance Tasks. There are a few guided activities for teachers to run that will help students get organized and ensure they have reasonable project plans that can be achieved in the time allotted.

Please note that instructors are required to provide the appropriate amount of class time for students to complete each Performance Task - **8 hours for Explore, 12 hours for Create**. The hours are the minimum amount of class time required, and do not need to be contiguous. In the official submission to the College Board, teachers will attest that all student work is original and that the required amount of time was provided.

Performance Tasks

AP Tech Setup

1. Digital Tools and the AP Digital Portfolio

1.1	1.1.1[P2] (B)	5.1	5.1.2[P2] (J)
1.2	1.2.1[P2] (ABE)	5.4	5.4.1[P4] (ACDEFHKM)
5.1	5.1.1[P2] (AB)		

Explore Task

1. Explore PT Prep - Reviewing the Task

1.1	1.1.1[P2](AB)	7.5	7.5.1[P1](ABC)
1.2	1.2.1[P2](ABCDE)	7.5	7.5.2[P5](AB)
1.2	1.2.3[P2](ABC)		

2. Explore PT Prep - Making a Plan

1.1	1.1.1[P2](AB)	7.5	7.5.1[P1](C)
1.2	1.2.1[P2](ABCDE)		

3. Explore PT - Complete the Task (8 hours)

1.2	1.2.1[P2] (ABCDE)	3.3	3.3.1[P4] (ABCDEFGHI)
1.2	1.2.2[P2] (AB)	7.1	7.1.1[P4] (A-O)
1.2	1.2.3[P2] (ABC)	7.2	7.2.1[P1] (A-G)
1.2	1.2.5[P4] (ABCD)	7.3	7.3.1[P4] (A-Q)
3.1	3.1.3[P5] (ABCDE)	7.4	7.4.1[P1] (ABCDE)
3.2	3.2.1[P1] (A-I)	7.5	7.5.1[P1] (ABC)

Create Task

1. Create PT Prep - Reviewing the Task

1.1	1.1.1[P2] (B)	5.1	5.1.2[P2] (A)
1.2	1.2.4[P2] (ABCDEF)	5.1	5.1.3[P6] (ABCDEF)

2. Create PT Prep - Making a Plan

1.2	1.2.1[P2] (ABE)	7.5	7.5.1[P1] (ABC)
1.2	1.2.4[P6](CDEF)	7.5	7.5.2[P5] (AB)

3. Create PT - Complete the Task (12 hours)

1.2	1.2.1[P2] (ABCDE)	5.1	5.1.1[P2] (ABCDEF)
1.2	1.2.2[P2] (AB)	5.1	5.1.2[P2] (A-J)
1.2	1.2.3[P2] (ABC)	5.1	5.1.3[P6] (ABCDEF)
1.2	1.2.4[P6] (ABCDEF)	5.2	5.2.1[P3] (A-K)
2.2	2.2.1[P2] (ABC)	5.3	5.3.1[P3] (A-O)
2.2	2.2.2[P3] (AB)	5.4	5.4.1[P4] (A-N)
4.1	4.1.1[P2] (A-I)	5.5	5.5.1[P1] (A-J)
4.1	4.1.2[P5] (A-I)		