Code.org Computer Science Principles

Syllabus and Overview

rev. June 2015

Overview

Code.org's Computer Science Principles (CSP) curriculum is a course designed around the AP Computer Science Principles Framework. It becomes an official AP® course in the 2016-17 school year. For context, it is useful to have familiarized yourself with the CSP framework before reading this document.

CS Principles is designed to be a full-year, rigorous, but entry-level course for high school students. Code.org's CSP curriculum is also written to support teachers new to the discipline with inquiry-based activities, videos, assessment support, and computing tools that have built-in tutorials and student pacing guides.

Below is a snapshot of the course. The course contains **four core units of study,** with a fifth unit devoted almost exclusively to students working on their *AP Performance Task* (PT) projects. Each unit includes a number of lessons that take from one to two class periods to complete, assuming 50-minute class periods.

Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
Digital Information	**The Internet**	**Programming**	**Data**	**Performance Tasks**
17 lessons ~6 weeks	20 Lessons ~6 weeks	34 Lessons ~8 weeks	23 Lessons ~6 weeks	Student time ~4 weeks

Sept	Oct	Nov	Dec	Jan	Feb	Mar	Apr

A timeline showing a typical Sept-May school year is shown to give a rough estimate of pacing. The month of May is not shown because the AP Exam and submission deadline will typically be during the first week of May.

Each unit contains at least one *practice* Performance Task that mimics the style of the Advanced Placement PTs. These practice PTs are smaller in scope than the real PTs and are intended to focus on particular elements or skills required to complete the PTs at the end of the course.

Unit 5 is set aside as time for students to work independently on their PTs to complete them for submission to the College Board as part of the official assessment for the course. Please note that the College Board mandates that a certain number of class hours be devoted solely to student work-time on these *Performance Tasks* - so we include it in the calendar to make sure this time is accounted for. However, a teacher may choose to incorporate this work time throughout the course in a variety of ways.

rev. June 2015

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.

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. 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.) and at least some experience with computer programming obtained through self-instruction, an online course, or other formal computer science training or coursework.

Technical Requirements

The course requires 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 web browser. While the course features many "unplugged" activities away from the computer, daily access to a computer is essential for every student. It is not required that students have access to computers at home, but because almost all of the materials are online, students with access to computers outside of class and at home will find it more convenient and easier to keep up with the pace of the lessons.

Resources & Materials

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

- Instructional guides for every lesson
- Formative and summative assessments, exemplars, rubrics, and teacher dashboard
- Student videos - including tutorials, instructional and inspirational videos
- Teacher videos - including lesson supports and pedagogical tips and tricks
- Widgets and simulators for exploring individual computing concepts
- *App Lab* - Code.org's JavaScript programming environment for making apps

A few lessons call for typical classroom supplies and manipulatives such as poster paper, markers, dixie cups, string, playing cards, a handful of Lego blocks, etc. In many cases there are alternatives to these materials as well. Costs should be low.

rev. June 2015

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 be accessed online **free of cost**. Many of its chapters are excellent supplemental reading for our course, especially for material in Units 1 and 2. We refer to chapters as supplemental reading in lesson plans as appropriate.

Addressing Diversity, Equity, and Broadening Participation in the Curriculum

Broadening participation and engaging diverse learners is a central goal of this course. To this end, we have worked to provide examples and activities that are culturally relevant and topical enough for students to connect back to their interests. 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 work of providing an accessible classroom doesn't stop with curriculum-- the classroom environment and teaching practice must also be structured such that all learners can access and engage with the material at a level that doesn't advantage a few 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. Broadening student participation in computer science is a national goal, and effectively a social justice issue. Fancy tools and 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 coming into the course.

As such the start of this CSP course purposefully addresses material that is fundamental to computing but with which many students, even those with computers at home or who have done some programming, are unfamiliar. This levels the playing field for participation and engagement right from the beginning of the course. We seek to establish classrooms in which at the outset every student, regardless of background, has a legitimate stake in the proceedings.

Curriculum Overview

The Internet and Innovation provide a narrative arc for the course, a thread connecting all of the units. The course starts with learning about what is involved in sending a single bit of information from one place to another, and ends with students developing small applications of their own design that live on the web.

Unit Structure

While the layout of units appears to be modular, the units of study are intended to be taught in the order suggested, and each not only builds students' skills and knowledge through the course, but there are many strong connections that can be made across the units. Each unit contains at least one summative assessment or project that asks students to do things similar to the official PTs. Sometimes these come mid-unit, sometimes closer to the end.

Lesson Structure and Philosophy

Lessons are designed to be student-centered and to engage students with inquiry-based and concept-discovery activities. The course does not require the new-to-computing teacher to lecture or present on computer science topics if they do not want to. Direct instruction is built into our tools and videos. The teacher plays a large role making choices and ensuring that the activities, inquiry, and reflection are engaging and appropriate for their students, as well as assessing student learning.

Most lessons follow a basic routine:

- A warm-up activity to activate prior knowledge and present a thought-provoking problem
- An activity that varies but is typically one of:
    - Unplugged concept invention, and problem solving scenarios
    - Creating digital artifacts (including programming)
    - Research / reflection / presentation
- A wrap-up activity or reflection

Assessment

The AP Assessment consists of a multiple choice exam and two "through-course" assessments called the *AP Performance Tasks* (PTs). For context it would be useful to familiarize yourself with the College Board documents. There are two:

- Explore Performance Task
- Create Performance Task

Summative Assessments

As mentioned previously there are several lessons in the curriculum that outline projects that are very similar to the AP PTs. We call them *Practice PTs*. Each unit contains at least one Practice PT and some have two. It is highly recommended that the teacher use these in order to help students prepare for the actual Performance Tasks.

rev. June 2015

Formative Assessments:

The curriculum provides teachers many opportunities for formative assessment (such as checks for understanding). These include, but are not limited to:

Assessments in Code Studio

- The Code Studio environment which serves as the central location for students to access resources for each lesson has built-in assessments related to the lesson. Some of these are multiple choice or matching questions. Some are free-response text fields where students may input their answer.
- Code Studio also keeps track of student work from the programming environment, and other digital tools and widgets
- The teacher may access and view any of these student works through their account

Worksheets and Activity Guides

- Many lessons contain worksheets or activity guides that ask students to write, answer questions, and respond to prompts (Answer keys provided).
- These can be collected as a form of formative assessment

Rubrics

- Rubrics are provided for the many opportunities for students to share work and/or present in the class
- Rubrics are provided for written work
- Rubrics for free-form programming assignments are also provided

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.

rev. June 2015

Coverage of Computational Thinking Practices (CTP)

As the framework says: *computational thinking practices capture important aspects of the work that computer scientists engage in*. These practices are essential to the experience of doing work in computing. These practices are not something that one covers once and then is done. Rather they represent higher order thinking skills, behaviors, and habits of mind that need to be constantly visited, repeatedly honed, and refined over time.

In this curriculum ***every lesson strives to incorporate multiple computational thinking practices.*** In each unit of study students will engage in each CTP multiple times. For reference here are the computational thinking practices as outlined in the CSP Framework

P1: Connecting Computing

- Identify impacts of computing.
- Describe connections between people and computing.
- Explain connections between computing concepts.

P2: Creating Computational Artifacts

- Create an artifact with a practical, personal, or societal intent.
- Select appropriate techniques to develop a computational artifact.
- Use appropriate algorithmic and information management principles.

P3: Abstracting

- Explain how data, information, or knowledge is represented for computational use.
- Explain how abstractions are used in computation or modeling.
- Identify abstractions.
- Describe modeling in a computational context.

P4: Analyzing Problems and Artifacts

- Evaluate a proposed solution to a problem.
- Locate and correct errors.
- Explain how an artifact functions.
- Justify appropriateness and correctness of a solution, model, or artifact.

P5: Communicating

- Explain the meaning of a result in context.
- Describe computation with accurate and precise language, notations, or visualizations
- Summarize the purpose of a computational artifact.

P6: Collaborating

- Collaborate with another student in solving a computational problem.
- Collaborate with another student in producing an artifact.
- Share the workload by providing individual contributions to an overall collaborative effort.
- Foster a constructive, collaborative climate by resolving conflicts and facilitating the contributions of a team member
- Exchange knowledge and feedback with a partner or team member.
- Review and revise their work as needed to create a high-quality artifact.

rev. June 2015

Coverage of the CS Principles Framework

The CS Principles Framework is not intended to be taught in any particular order. Similar to the 6 Computational Thinking Practices, the **7 Big Ideas** of CS Principles are not ideas you can "cover" one at a time. The learning objectives associated with each overlap, intersect, and reference each other and multiple big ideas. For example, a learning objective listed under the big idea *Abstraction* also references *Programming*: "Develop an abstraction when writing a program or creating other computational artifacts".

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 more general principles of computing: Creativity, Abstraction, Algorithms and Global Impacts. It is much more usefully viewed in two dimensions (see below).

Unit 1 actually addresses items from almost all of the big ideas, but heavily emphasizes items from the big ideas **Abstraction** and **Creativity.** Students invent, solve problems and create many artifacts in Unit 1 related to the digital representation of information and the implications of attempting to encode information in ways that computers can process (in binary). See the full unit descriptions for more information.

For Units 2, 3 and 4, we treat the Big Ideas *Internet*, *Programming*, and *Data* as major topics of study. We ensure that we cover all aspects of those topics by looking at their intersections with the other 4 big ideas: *Creativity*, *Abstraction*, *Algorithms*, *Global Impact*. The chart below shows the intersections of the big ideas and examples of topics addressed in the curriculum.

	Unit 2	Unit 3	Unit 4
	Internet	Programming	Data
Creativity	Invent a communication protocol	Make a digital scene. Program an app.	Make your own data-backed app
Abstraction	Internet Protocols	Writing procedures and functions	Visualizing Data
Algorithms	Routing, Encryption	string manipulation image processing	Searching and data mining
Global Impact	Security, Privacy, Hacking	Software can solve some but not all problems	Implications of collection and storage of big data

rev. June 2015

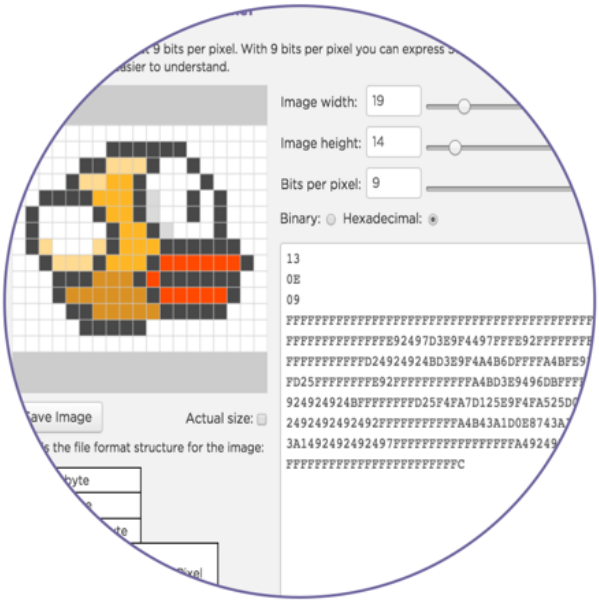
Unit Overviews

What follows are more in-depth descriptions of each unit of study which explain the topics covered and what students will be doing. 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 Digital Representation of Information

This unit sets the foundation for thinking about the digital (binary) representation of information and how that affects the world we live in. 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 - numbers, text, images, and communication protocols.

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 devise and build their own bit-sending "machines." Complexity increases as students adapt their machines to handle multi-bit messages, and increasingly complex information. For encoding information that can be sent between devices we use an Internet Simulator that allows students to develop and test binary encodings and communication protocols of their own invention.



rev. June 2015

Unit 1 Lessons

01 - The Impact of Innovation
02 - Sending Binary Messages
03 - Sending Complex Messages
04 - Sending Binary Messages with the Internet Simulator
05 - Sending Bits in the Real World
06 - Number Systems - Circles, Triangles, Squares
07 - Encoding Numbers in Binary
08 - Sending Numbers
09 - Number Systems in the Real World
10 - Encoding Text
11 - Sending Formatted Text
12 - Bytes and File Sizes
13 - Text Compression
14 - Encoding B&W Images
15 - Encoding Color Images

16 - Practice PT - Encode a complex thing

Project: Encode a Complex Thing
Students invent a binary encoding (file format) for a piece of information that they find personally meaningful. How might you encode smell? 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. While the project is done individually the lesson helps students through an iterative feedback process with a partner. This assignment emphasizes the writing process, and giving and incorporating feedback from peers.

Learning Objectives Addressed:
Creativity: 1.1.1, 1.2.4
Abstraction: 2.1.1, 2.1.2, 2.2.1
Data: 3.2.1, 3.3.1

Computational Thinking Practices Emphasized:
P1: Connecting Computing
P3: Abstracting
P5: Communicating
P6: Collaborating

rev. June 2015

Unit 2: The Internet

This unit largely explores the structure and design of the Internet and the implications of those design decisions including the reliability of network communication, the security of data, and personal privacy. Topics include the Internet Protocol (IP), DNS, TCP/IP, cryptography and other security and hacking concerns. Students are introduced to algorithms formally in this unit by considering shortest path problems for routing. The unit also makes the link between the existence of computationally hard problems and encryption schemes that are “hard” for computers to crack.

The unit starts with students being presented with a more robust Internet Simulator that students will use to solve some of the classic problems of network communication such as addressing devices, routing traffic, and developing packet switching. Students work together to invent solutions and protocols to many of the problems that arise. The second half of the unit asks students to consider how information might be encrypted to ensure privacy and some of the tradeoffs involved.

rev. June 2015

Unit 2 Lessons

Part 1: How the Internet Works

01 - Building the Internet
02 - The Need for Addressing
03 - Invent an Addressing Protocol
04 - Routers and Addresses as Numbers
05 - The Need for Packets
06 - Algorithms - Minimum Spanning Tree
07 - Algorithms - Shortest Path Problem
08 - Algorithms - How Routers Learn
09 - The Need for DNS
10 - DNS in the Real World
11 - HTTP and Abstraction
12 - Practice PT- Challenges of the Internet

Part 2: Security, Privacy and Crime

13 - Tell Me a Secret - Encrypting Text
14 - Cracking the Code
15 - Encryption Algorithms
16 - Alice and Bob and Asymmetric keys
17 - Computationally Hard Problems - TSP
18 - One Way Functions - Ice Cream Vans
19 - Public Key Crypto

20 - Practice PT- Security and Hacking in the Real World

Project: Security and Hacking
This assignment mimics many of the elements of the *Explore Performance Task*. Students will investigate and research an issue related to internet security or privacy examine it with a critical eye to demonstrate a deep understanding of the issue, its functionality, and its potential impact on people and society. Students may investigate such topics as Net Neutrality, cybercrime, or other legal, political, or societal issues that stem from the structure and and usage of the Internet. A key element of the assignment is *communicating* and explaining the interplay between the technology and societal issue.

Learning Objectives Addressed:
Data: 3.3.1
Internet: 6.1.1, 6.2.1, 6.2.2, 6.3.1
Global Impacts: 7.3.1, 7.4.1

Computational Thinking Practices Emphasized:
P1: Connecting Computing
P5: Communicating

rev. June 2015

Unit 3: 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. Students will program in an online programming environment called *App Lab* that has many features, chief among them the ability to write JavaScript programs with click-and-drag blocks or just typing text - allowing the user to switch back and forth at will. This should greatly ease the transition to typing text-based programming languages.

The unit begins with students solving problems with classic turtle-style programming, focusing on the power of procedural abstraction and personal expression with code. After learning some basics of programming with the turtle, we transition to more event-driven apps, gradually blending in common user interface objects like buttons and text inputs, images and so on. Over the unit students create 4 apps of some significance - each emphasizing a different aspect of programming. Students will continue programming in the next unit of study, but this unit provides an introduction to programming from the ground up, almost literally.

rev. June 2015

Unit 3 Lessons

Part 1: Turtle Programming and Procedural Abstraction
01 - The Need For Programming Languages
02 - Using Simple Commands
03 - Creating Procedures - Part 1
04 - Creating Procedures - Part 2
05 - Abstraction and Programming Efficiency
06 - APIs and Function Parameters
07 - Looping and Random Numbers
08 - Creating functions with Parameters

09 - Scene Design Part 1: Plan
10 - Scene Design Part 2: Code
11 - Scene Design Part 3: Reflect

Part 2: Event Driven Programming
12 - Events Unplugged
13 - Buttons on the Screen
14 - Unfortunate Events
15 - Beyond Buttons Toward Apps
16 - Cookie Clicker
17 - Improving the Clicker Game
18 - Controlling Memory: variable basics
19 - Clicker Variable Basics: scope and arithmetic
20 - Conditional Basics
21 - More Variables: types
22 - Permanent Data Storage and Clicker Game

Part 3: String Processing
23 - Natural Language Processing and Chat Bots
24 - Chained Conditionals
25 - Compound Conditionals
26 - Strings and Substrings
27 - Chatbot Challenge

Part 4: Loops and Arrays
28 - While Loop Basics
29 - While Loops - Counting Flips
30 - Arrays - Intro
31 - Arrays - Photo Album
32 - Images are Arrays!
33 - Arrays and Loops - Altering Images
34 - Arrays and Loops - Hidden Images

Project: Digital Scene Design
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, and some insight is given into how real engineering teams do this. Students are asked to reflect on their experience in a way that is similar to the *Create* performance task. In terms of programming, a heavy emphasis is on writing functions (procedures) that can be easily incorporated into others' code.

Learning Objectives Addressed:
Creativity: 1.1.1, 1.2.1, 1.2.4, 1.3.1
Abstraction: 2.2.1, 2.2.2
Algorithms: 4.1.1
Programming: 5.1.1, 5.1.3, 5.3.1

Computational Practices Emphasized:
P2: Creating Computational Artifacts
P3: Abstracting
P6: Collaborating

rev. June 2015

Unit 4: Data

In this unit students continue programming and building apps, but now with a heavier focus on data. Being able to extract knowledge from data is an important aspect of CS Principles and in this unit students will do that in a number of ways. Students will write programs that generate data to model or simulate a scenario they wish to investigate. Students will process large lists of data imported from other sources and also pull data from live data APIs. Students will also more fully use App Lab's cloud data storage capabilities to create databases to use with their own apps.

The unit begins with students designing and running monte carlo-type experiments to investigate the answer to data-driven questions that can be simulated on the computer with many trials. Students then write programs that process large lists of data to perform simple searches or aggregations. The unit concludes with some big data investigations that encourages students to query a remote API that can return data and artifacts they can use in their apps.

rev. June 2015

Unit 4 Lessons

Introduction to Big Data
01 - What is Big Data?
02 - APIs and access to large data sets
03 - Structuring data: JSON

Collection & Storage
04 - Usable and/or Useful Data?

05 - Make an app that collects data

06 - Data Persistence: What Happens Online Stays Online
07 - When storage goes wrong

Security & Privacy
08 - What the internet knows about you
09 - Classroom debate: Privacy vs. Utility
10 - What hackers do when they find databases

11 - Make a data privacy policy for your app

Extraction & Cleaning
12 - Needles and Haystacks - How to do you find anything in big data?
13 - Indexing/counting/sorting
14 - Import and clean data

15 - What can we extract from our own data?

Analysis & Visualization
16 - Are we seeing the same thing?
17 - Simple analysis through visualization
18 - Important tools for analysis

19. Plan your visualization

Knowledge Discovery
20 - Uses of statistical analysis & Data Mining
21 - Cluster, Outliers, Associations, Regressions, Classifications
22 - Tools for analysis

Project:
23 - Make visual explanation and analysis

Project: Make a Web App
This unit features a large ongoing project that students will continually build, edit, and revisit during the unit. Students will each create an app of their own design that collects data of some kind about its users. This app is used as a reference point to address many real-world issues that arise with data collection, in both technical and ethical realms. The student's app serves as a constant reminder that *anyone* can create apps that collect data, even early learners. With that power comes the responsibility of understanding the implications of what you're doing, and insight into what others are doing as well. This last project visits elements of all seven of the Big Ideas of the Framework.

Learning Objectives Addressed:
Creativity: 1.1.1, 1.2.1, 1.2.2, 1.2.4
Abstraction: 2.1.1, 2.2.2
Data: 3.1.1, 3.1.2, 3.1.3, 3.2.1, 3.3.1
Algorithms: 4.1.1, 4.1.2, 4.2.4
Programming: 5.1.1, 5.2.1, 5.3.1, 5.4.1, 5.5.1
Internet: 6.3.1
Global Impacts: 7.3.1, 7.4.1

Computational Thinking Practices Emphasized:
P1: Connecting Computing
P2: Creating Computational Artifacts
P3: Abstracting
P4: Analyzing Problems and Artifacts
P5: Communication

rev. June 2015

Unit 5 - Performance Tasks

This unit is primarily set aside to ensure that students have enough time in class to work on and complete their performance tasks for submission to the college board. 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.

Create Performance Task (Create PT)

For a full description of the Create PT see the College Board website for AP CS Principles.

For lessons related to the *Create PT* we will revisit the structure of the Digital Scene Design project from Unit 3, which had students go through a 3-stage process to develop the final artifact: 1) Plan 2) Code 3) Reflect. Emphasis will be placed on the planning portion to help students come up with realistic goals for writing code, writing reflective prose, and preparing the project for submission to the College Board.

Explore Performance Task (Explore PT)

For a full description of the Create PT see the College Board website for AP CS Principles.

For lessons related to the Explore PT we will revisit lessons learned from the Unit 2 lesson: Security and Hacking in the Real World which was a research project similar to what is asked for in the Explore PT. Students should have an innovation in mind that they want to research coming into this Unit. Emphasis again will be placed on planning a realistic timeline to get the necessary work done and prepared for submission to the College Board.

rev. June 2015

Unit 5 Lessons

Part 1: Create Performance Task (Create PT)

- Revisit Digital Scene Design
  - Plan
  - Code
  - Reflect
- Individual and Group Work Time

Part 2: Explore Performance Task (Explore PT)

- Revisit Security and Hacking in the Real World
  - Research
  - Write
  - Reflect
- Individual and Group Work Time

Teacher Guidance
In Units 1-4 students engaged in projects to learn and practice the skills and content they needed to know 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. In the official submission to the College Board, teachers will attest that all student work is original and no work from earlier assignments is included.

Learning Objectives Addressed:
Creativity: 1.2.1, 1.2.2, 1.2.3, 1.2.4, 1.2.5
Abstraction: 2.2.1, 2.2.2
Data: 3.3.1
Algorithms: 4.1.1, 4.1.2
Programming: 5.1.1, 5.1.2, 5.1.3, 5.2.1, 5.3.1, 5.4.1, 5.5.1
Global Impacts: 7.1.1, 7.2.1, 7.3.1, 7.4.1

Computational Thinking Practices Emphasized:
P1: Connecting Computing
P2: Creating Computational Artifacts
P3: Abstracting
P4: Analyzing Problems and Artifacts
P5: Communication
P6: Collaborating