Code.org Computer Science Principles
Syllabus and Overview

rev. January 2016

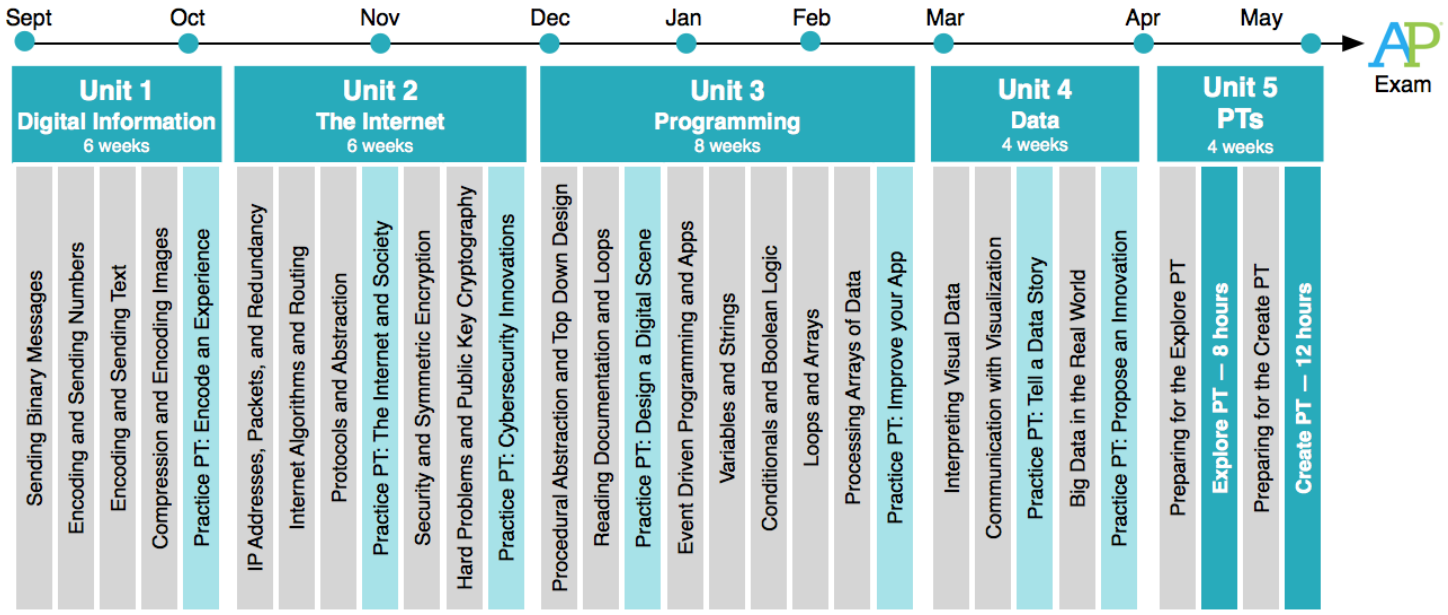
AP Computer Science Principles

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 topics that make up computing such as programming, algorithms, the Internet, big data, digital privacy and security, and the societal impacts of computing.

The course is designed around the AP Computer Science Principles Framework and **prepares students to take the AP exam and to complete the AP Performance Tasks**. For context, it is useful to be familiar with the CSP Framework before reading this document.

Course Snapshot

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 gray box in the diagram represents a group of 2-5 lessons which each take from one to two class periods to complete, assuming 50-minute class periods. A timeline showing a typical school year is shown to give a rough estimate of pacing. **Note: the AP Exam and submission deadline is typically the first week of May.**



rev. January 2016

Curriculum Overview and Goals

Computing affects almost all aspects of modern life and *all* students deserve a computing education that prepares them to pursue the wide array of intellectual and career opportunities that computing has made possible.

This course is not a tour of current events and technologies. Rather, this course 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.

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 considering the implications of a computing innovation of their own design. Along the way students learn:

- How the Internet works and its impacts on society.
- How to program and rapidly prototype small JavaScript applications both to solve problems and to satisfy personal curiosity.
- How to collect, analyze and visualize data to gain insight and knowledge.
- How to evaluate the beneficial and harmful effects to people and society brought on by computing innovations.

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, but especially in the videos that accompany the curriculum, we seek to **highlight a diverse and impressive 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 are 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. 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 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. Students should not feel intimidated that others in the class are starting with a leg up on the material.

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 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 in the description of Unit 1.

rev. January 2016

Teaching the course

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.

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 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.

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

Unit Structure: Units, Chapters, Lessons

While the layout of units appears to be modular, the units of study are loosely scaffolded, **and sequenced 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 **chapters** of a few lessons each whose content is related or connected in some way. The course snapshot on the previous 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.

rev. January 2016

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, where necessary, is built into our tools and videos.

Another goal of each lesson is to provide more resources, supports, and activities than a teacher could (or should) use in one lesson. **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 have the following structure:

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

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 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.

Computational Tools, Resources and Materials

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 and simulators 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

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 most cases there are alternatives to these materials if necessary. Costs should be low.

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, 2 and 4. We refer to chapters as supplemental reading in lesson plans as appropriate.

AP® Assessment

The AP Assessment consists of a 74-question 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

Assessments in the Curriculum

The course provides a number of assessment types and opportunities. 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.

Summative Assessments:

The curriculum contains two types of summative assessments that teachers may elect to use. They are intended to mimic the AP assessments though in more bite-sized chunks.

Fixed Response (multiple choice) Assessments

Each "chapter" of the curriculum - typically a sequence of 2-5 lessons - has an associated short multiple choice-style assessment that addresses material in those lessons.

Practice Performance Task Assessments

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.

Rubrics
The curriculum contains rubrics for assessing certain kinds of student work:

- Written and project work
- Practice PTs
- Programming projects
- Student presentations

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
All lesson materials can be accessed by students on a single platform called Code Studio. In addition to housing lesson descriptions, instructional materials, and programming exercises in App Lab, Code Studio includes features that assist the teacher in completing formative assessment including:

- 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

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

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. January 2016

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. The Big Ideas all 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.

For example, a learning objective listed under the Big Idea *Abstraction* also references the Practice of *Programming*.

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

Even though this particular learning objective highlights practice *P2: Creating Computational Artifacts,* it clearly will also engage the practice of *Abstracting.* Therefore, this single learning objective represents an intersection of two *Big Ideas*: Abstraction and Programming, while also engaging at least two *Computational Thinking Practices*.

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 computing: Creativity, Abstraction, Algorithms and Global Impacts. It is much more usefully viewed in two dimensions.

rev. January 2016

	Internet	Programming	Data
Creativity	Invent a communication protocol	Make a digital scene. Program an app.	Visualizing Data Create a visualization
Abstraction	Internet Protocols	Writing procedures and functions	Encoding images in binary
Algorithms	Routing, Encryption	String manipulation Array 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
	Unit 2	**Unit 3**	**Unit 4**

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.

What about Unit 1? **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 things, 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.

The six **computational thinking practices** are addressed continuously throughout the curriculum in a number of ways. They are woven into the curriculum, engineered into activities and projects, as well as in teaching tips for lessons. The acts of **abstracting**[P3] and **creating** and **analyzing computational artifacts** [P2 and P4] are part and parcel of many of the lessons, activities, and projects, themselves. The teacher plays a large role in ensuring that students are **connecting computing** [P1], **collaborating** [P6] effectively, and **communicating** [P5] both in writing and speaking. You can find explicit reference to the computational practices used in lessons in the unit overviews below.

rev. January 2016

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.

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.

rev. January 2016

Unit 1 Lessons

Chapters	LO [P] (Ek)	Lessons / Topics
Getting Started	7.1.1 [P4] (A-O) 7.2.1 [P1] (A-C,G) 7.3.1 [P4] (A-O) 7.4.1 [P1] (A-D)	Personal Innovations
Sending Binary Messages	2.1.1 [P3] (A-C,E) 2.1.2 [P5] (D-F) 2.3.1 [P3] (A-D) 2.3.2 [P3] (A) 3.3.1 [P4] (A-B) 6.1.1 [P3] (A-D) 6.2.1 [P5] (A,D) 6.2.2 [P4] (A-K)	Sending Binary Messages Sending Complex Messages Sending Binary Messages with the Internet Simulator Sending Bits in the Real World
Encoding and Sending Numbers	2.1.1 [P3] (A-G) 2.1.2 [P5] (A-F) 2.3.1 [P3] (A-D) 2.3.2 [P3] (A-E) 3.1.1 [P4] (A,B,D,E) 3.3.1 [P4] (A,B) 6.2.2 [P4] (D,G,H)	Number Systems - Circles, Triangles, Squares Binary Numbers Sending Numbers Encoding Numbers in the Real World
Encoding and Sending Text	2.1.1 [P3] (A-E) 2.1.2 [P5] (B-F) 2.2.1 [P2] (A,B) 2.3.1 [P3] (A-D) 2.3.2 [P3] (A-E) 3.1.1 [P4] (A,D,E) 3.1.2 [P6] (A-D) 3.1.3 [P5] (A,E) 3.3.1 [P4] (A,B,G) 4.2.1 [P1] (A-D) 4.2.3 [P1] (A-C) 4.2.4 [P4] (A,C,D) 6.1.1 [P3] (A-D) 6.2.2 [P4] (D,F-H)	Encoding and Sending Text Sending Formatted Text Bytes and File Sizes
Compression and Encoding Images	1.1.1 [P2] (A,B) 1.2.1 [P2] (A) 1.3.1 [P2] (C) 2.1.1 [P3] (A-C) 2.1.2 [P5] (D-F) 2.2.1 [P2] (A,B) 2.3.1 [P3] (A-D) 3.1.1 [P4] (A,D,E) 3.1.2 [P6] (A-D) 3.1.3 [P5] (A,E) 3.2.1 [P1] (G-I) 3.3.1 [P4] (A-E,G)	Text Compression Encoding B&W Images Encoding Color Images Lossy Compression and File Formats
Practice PT	2.1.1 [P3] (A-E) 2.1.2 [P5] (A,B,D,F) 2.2.1 [P2] (A,B)	**Practice PT - Encode an Experience**

rev. January 2016

Unit 1: Practice PT Highlight

Practice PT: Encode an Experience
Students invent a binary encoding (file format) for a real life experience. Students must figure out a way to encode or represent with data, the elements of some kind of human 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. 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 [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

rev. January 2016

Unit 2: The Internet

This unit 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.

The unit has two logical parts 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. January 2016

Unit 2 Lessons

Chapters	LO [P] (Ek)	Lessons
Getting Started	6.1.1 [P3] (B,C,E) 6.2.2 [P4] (B) 7.3.1 [P4] (A,D,E,G,L) 7.4.1 [P1] (C-E)	The Internet is for Everyone
Internet Addresses, Packets, and Redundancy	2.1.1 [P3] (A-C,E) 2.1.2 [P5] (D-F) 3.3.1 [P4] (A-F) 6.1.1 [P3] (B-E) 6.2.1 [P5] (D) 6.2.2 [P4] (B,D,G) 6.3.1 [P1] (A)	The Need for Addressing Invent an Addressing Protocol Routers and Redundancy Packets and Making a Reliable Internet
Algorithms of the Internet: Routing	4.1.1 [P2] (B,H,I) 4.1.2 [P5] (A-C,F,I) 4.2.1 [P1] (A,B) 4.2.4 [P4] (A-D,G)	Minimum Spanning Tree Shortest Path Problem How Routers Learn
Protocols and Abstraction	6.1.1 [P3] (A-I) 6.2.1 [P5] (B,C) 6.2.2 [P4] (C-E,H) 6.3.1 [P1] (B)	The Need for DNS DNS in the Real World HTTP and Abstraction
Practice PT	6.3.1 [P1] (A,B) 7.1.1 [P4] (A-D,H-K,M,0) 7.3.1 [P4] (A-Q) 7.4.1 [P1] (A,B,D,E) 7.5.1 [P1] (A,B) 7.5.2 [P5] (A,B)	**Practice PT: The Internet and Society**
Security and Symmetric Encryption	2.3.2 [P3] (A) 3.1.1 [P4] (A) 3.1.2 [P6] (A,C) 3.3.1 [P4] (B,E,F) 4.2.1 [P1] (A,C,D) 6.3.1 [P1] (C, H-K) 7.3.1 [P4] (G)	Tell Me a Secret - Encrypting Text Cracking the Code Keys and Passwords
Hard problems and Asymmetric Encryption	2.3.1 [P3] (A,B) 4.2.1 [P1] (A-D) 4.2.2 [P1] (A-D) 4.2.3 [P1] (A,D) 4.2.4 [P4] (A-C) 6.3.1 [P1] (H-L)	Hard Problems - The Traveling Salesperson Problem One Way Functions - The WiFi Hotspot Problem Asymmetric Keys - Cups and Beans Public Key Crypto
Practice PT	1.1.1 [P2] (A,B) 1.2.1 [P2] (A-C,E) 1.2.2 [P2] (A) 1.2.5 [P4] (B) 6.3.1 [P1] (A-M) 7.3.1 [P4] (A,D,G,H,L) 7.4.1 [P1] (A,B,E) 7.5.1 [P1] (A,B) 7.5.2 [P5] (A,B)	**Practice PT- Cybersecurity Innovations**

rev. January 2016

Unit 2: Practice PT highlights

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 connecting these ideas to global and social impacts. Students will practice synthesizing information, and presenting their learning in a flash talk.

Learning Objectives Addressed:
Internet: 6.3.1 [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

Practice PT: Cybersecurity Innovations
Students will complete a research project on an innovation of their choosing. Students will need to identify appropriate online resources to learn about the functionality, context, and impact of their cybersecurity innovation. After completing their research, students will present their findings both in a written summary and with an audio / visual artifact they found online. 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]

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

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 created by Code.org called *App Lab* that has many features, chief among them the ability to write JavaScript programs with click-and-drag blocks as well as 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, students transition to more event-driven apps, gradually blending in common user interface objects like buttons and text inputs, images and so on.

Students create a number of small exemplar apps during the unit each emphasizing a different aspect of programming:

- a digital scene created with turtle graphics
- a simple clicker game
- an “intelligent” digital assistant
- a coin-flipping simulation
- a drawing effects app

The unit also features two practice performance tasks. The first: *Design a Digital Scene* asks students to collaborate and share code with their team to create a small scene. The second: *Improve Your App* asks students to look back at the exemplar apps they’ve created during the unit and use one as a point of inspiration for creating their own app.

rev. January 2016

Unit 3 Lessons

Chapters	LO [P] (Ek)	Lessons
Getting Started	4.1.2 [P5] (A-C, F, I) 5.2.1 [P3] (E)	The Need For Programming Languages
Procedural Abstraction and Top-Down Design	2.2.1 [P2] (A, B) 2.2.2 [P3] (A, B) 2.2.3 [P3] (A) 5.1.2 [P2] (A-C, I) 5.1.3 [P6] (A-F) 5.2.1 [P3] (A, B) 5.3.1 [P3] (A-D, L) 5.4.1 [P4] (A-E, I)	Using Simple Commands Creating Functions Functions and Top-Down Design
Documentation and Simple Loops	2.2.1 [P2] (C) 2.2.2 [P3] (A, B) 2.2.3 [P3] (A, B) 4.1.1 [P2] (D) 5.1.2 [P2] (B-F) 5.3.1 [P3] (A, C-G, L-O) 5.4.1 [P4] (C-K)	APIs and Function Parameters Creating functions with Parameters Looping and Random Numbers
Practice PT	2.2.1 [P2] (C) 2.2.2 [P3] (A, B) 2.2.3 [P3] (A, B) 4.1.1 [P2] (D) 5.1.2 [P2] (B, C) 5.1.3 [P6] (A-F) 5.3.1 [P3] (A, C, D, F, G, L) 5.4.1 [P4] (C-K)	**Design a Digital Scene**
Event Driven Programming and Apps	1.1.1 [P2] (A, B) 1.2.1 [P2] (A-E) 2.2.1 [P2] (B, C) 5.1.1 [P2] (A-C) 5.1.2 [P2] (J) 5.2.1 [P3] (D, G, H) 5.4.1 [P4] (C, E, F, M)	Events Unplugged Event-Driven Programming and Debugging Beyond Buttons Toward Apps Introducing Design Mode Multi-screen Apps
Variables and Strings	4.1.1 [P2] (A, C) 5.1.1 [P2] (B) 5.2.1 [P3] (C, F) 5.3.1 [P3] (I)	Controlling Memory with Variables Using Variables in Apps User Input and Strings
Conditionals and Boolean Logic	1.2.3 [P2] (A-C) 1.2.4 [P6] (A-D) 1.3.1 [P2] (E) 2.2.3 [P3] (F) 4.1.1 [P2] (A-C, I) 5.1.2 [P2] (A-C) 5.1.3 [P6] (A-F) 5.3.1 [P3] (I) 5.5.1 [P1] (E-G) 7.1.1 [P4] (L-N)	Introduction to Digital Assistants Understanding Program Flow and Logic Introduction to Conditional Logic Compound Conditional Logic Digital Assistant Project
Loops and Arrays	2.3.1 [P3] (A, C, D) 2.3.2 [P3] (A-F) 3.1.1 [P4] (A) 4.1.1 [P2] (A-D, H) 4.1.2 [P5] (A-G) 5.1.1 [P2] (A, B)	While Loops Loops and Simulations Introduction to Arrays

	5.1.3 [P6] (A-F) 5.2.1 [P3] (A-F, I-K) 5.3.1 [P3] (A-D, G, K, L) 5.4.1 [P4] (B, C, E-H, K-M) 5.5.1 [P1] (D-J)	Image Scroller with Key Events
Processing Arrays of Data	1.1.1 [P2] (B) 1.2.1 [P2] (A-D) 1.2.3 [P2] (A-C) 1.3.1 [P2] (C-E) 2.2.1 [P2] (A-C) 2.2.2 [P3] (A, B) 4.1.1 [P2] (A-I) 4.1.2 [P5] (A-C, G, I) 4.2.4 [P4] (D-F, H) 5.1.1 [P2] (A-E) 5.1.2 [P2] (A-C, J) 5.2.1 [P3] (A-F, I, J) 5.3.1 [P3] (A-G, J-L) 5.4.1 [P4] (A-H, L-N) 5.5.1 [P1] (D-J)	Processing Arrays Functions with Return Values Canvas and Arrays in Apps
Practice PT	1.1.1 [P2] (A, B) 1.2.1 [P2] (A-E) 1.2.2 [P2] (A, B) 1.2.3 [P2] (A-C) 1.2.4 [P6] (A-F) 1.2.5 [P4] (A-D) 2.2.1 [P2] (A-C) 2.2.2 [P3] (A, B) 4.1.1 [P2] (A-I) 4.1.2 [P5] (A-I) 5.1.1 [P2] (A-E) 5.1.2 [P2] (A-J) 5.1.3 [P6] (A-F) 5.2.1 [P3] (A-F, I-K) 5.3.1 [P3] (A-O) 5.4.1 [P4] (C, E-H, J, L-N) 5.5.1 [P1] (A-J)	**Improve Your App**

Unit 3 Practice PT Highlights

Practice PT: 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 [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

rev. January 2016

Practice PT: Improve Your App
To conclude their introduction to programming, 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. Students can either complete the project individually or with a partner. Every student will need a collaborative partner with whom they will give and receive feedback.

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

rev. January 2016

Unit 4: Data

Being able to digitally manipulate data, visualize it, identify patterns, trends and possible meanings are important practical skills that computer scientists do every day. The data rich world we live in also introduces many complex questions related to public policy, law, ethics and societal impact. Understanding where data comes from, having intuitions about what could be learned or extracted from it, and being able to use computational tools to manipulate data and communicate about it are the primary skills addressed in the unit.

Chapters	LO [P] (Ek)	Lessons / Topics
Getting Started	3.2.1 [P1] (A,B,C) 5.1.1 [P2] (F) 7.1.1 [P4] (C) 7.2.1 [P1] (A,B,G)	Introduction to Data - The Tracker Project
Interpreting Visual Data	1.1.1 [P2] (A,B) 1.2.1 [P2] (A,B,E) 1.2.5 [P4] (A-D) 3.1.1 [P4] (A,B,D,E) 3.1.2 [P6] (A-F) 3.1.3 [P5] (A-E) 3.2.1 [P1] (A-E) 7.1.1 [P4] (E-G) 7.4.1 [P1] (A,C,D)	Telling Stories with Visualizations - Google Trends Good v. Bad Visualization Digital Divides
Communicating with Visualization	1.1.1 [P2] (A,B) 1.2.1 [P2] (A-C) 1.2.4 [P6] (A,B,F) 3.1.1 [P4] (A-E) 3.1.2 [P6] (A-F) 3.1.3 [P5] (A-C) 3.2.1 [P1] (A-G,I) 3.2.2 [P3] (C,G) 3.3.1 [P4] (F) 7.3.1 [P4] (G)	What's the story? Chart it - using Visualization for Discovery Cleaning and Manipulating your data Summarizing Data in Tables
Practice PT	1.2.1 [P2] (A-C,E) 1.2.2 [P2] (A,B) 1.2.5 [P4] (A-D) 3.1.3 [P5] (A-D) 7.3.1 [P4] (J) 7.5.2 [P5] (A,B)	Tell a Data Story
Data in the Real World	1.2.5 [P4] (A-D) 3.1.1 [P4] (C-E) 3.1.2 [P6] (F) 3.2.1 [P1] (A-D,G) 3.2.2 [P3] (A-D,G,H) 3.3.1 [P4] (A,B,F) 7.1.1 [P4] (F) 7.2.1 [P1] (A) 7.3.1 [P4] (A,D-M) 7.5.2 [P5] (A,B)	Big data - Where does it come from? Big Public Data - datasets and APIs Security and Privacy in the world of data Public policy and privacy policies
Practice PT	1.2.5 [P4] (A-D) 3.1.3 [P5] (A-E) 3.3.1 [P4] (A,B,F) 7.1.2 [P4] (D,E,F,G) 7.3.1 [P4] (G,H)	Propose an Innovation

rev. January 2016

Unit 4 Practice PT Highlights

Tell a Data Story - Communicate Data Visually
This small project culminates a series of lessons in which students, provided a set of raw data, must use digital tools to collaboratively investigate the data to discover possible connections and trends. In the end students must produce a visual explanation of their findings in the data and write a small about about what the data shows. The emphasis is on producing the visual communication. The reflection questions mimic those on the Explore PT.

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

Computational Practices Emphasized:
P1: Connecting Computing
P2: Creating Computational Artifacts
P5: Communicating
P6: Collaborating

Practice PT - Propose an Innovation
Connecting back to the very beginning of the course, students here collaboratively *propose* a computing innovation of their own imagining that would positively affect or impact some community, group, or individual. As part of the proposal students must explain how the innovation would collect or use data, develop a privacy policy around its use, anticipate the possible negative effects the innovation might have and explain tradeoffs that need to be considered. This project prepares students for various aspects of the Explore Performance Task, particularly in considering how a computing innovation produces and consumes data and the beneficial and harmful effects that might result.

Learning Objectives Addressed:
Creativity: 1.3.1 [P2]
Data: 3.1.3 [P5], 3.3.1 [P4]
Gbal Impacts: 7.1.1 [P4], 7.3.1 [P4], 7.4.1 [P1]

Computational Practices Emphasized:
P1: Connecting Computing
P4: Analyzing Problems and Artifacts
P5: Communicating
P6: Collaborating

rev. January 2016

Unit 5 - Performance Tasks

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. 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. In the official submission to the College Board, teachers will attest that all student work is original and that the appropriate amount of class time - 8 hours for *Explore*, 12 hours for *Create* - was provided.

Chapters	LO [P] (Ek)	Lessons / Topics
Explore PT Overview	7.5.1 [P1] (A,B) 7.5.2 [P5] (A,B)	Planning to do the Explore PT Research Tips and Tricks Requirements and managing time.
Explore PT		**Administration of** ***Explore* Performance Tasks** **8 hours**
Create PT Overview	5.1.1 [P2] (A, B, C) 5.1.2 [P2](A, B, C) 5.1.3 [P6] (B,C)	Planning to do the Create PT Requirements and managing time.
Create PT		**Administration of** ***Create* Performance Tasks** **12 hours**