Code.org Computer Science A

Code.org's Computer Science A (CSA) curriculum is a **full-year**, **rigorous** curriculum that introduces students to software engineering and object-oriented programming and design using the Java programming language. This curriculum covers a broad range of topics, including the design of solutions to problems, the use of data structures to organize large sets of data, the development and implementation of algorithms to process data and discover new information, the analysis of potential solutions, and the ethical and social implications of computing systems. All teacher and student materials are provided for free online and can be accessed at code.org/csa.

Curriculum Overview and Goals

Software development and design is a core component of writing apps and software that users interact with every day. This curriculum seeks to develop software engineering skills and expose students interested in continuing their computer science education to best practices used in the industry while learning the Java programming language.

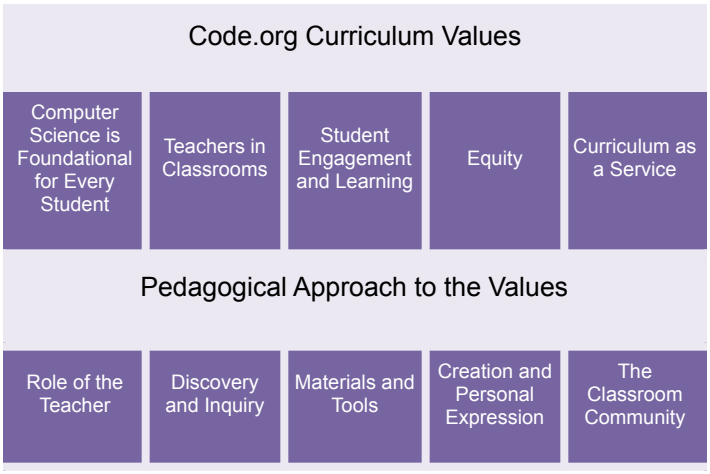
Unit		Description
Unit 1	**Object-Oriented Programming** *20 45-minute lessons*	Students learn fundamental Java concepts, discover their identity as software engineers, and use debugging strategies and code reviews to improve their programming skills.
Unit 2	**Class Structure and Design** *15 45-minute lessons*	Students develop an understanding of data encapsulation and object-oriented design decisions.
Unit 3	**Arrays and Algorithms** *15 45-minute lessons*	Students use one-dimensional (1D) arrays to store lists of primitive values and object references.
Unit 4	**Conditions and Logic** *15 45-minute lessons*	Students improve decision-making in their programs using relational and logical operators to evaluate primitive values and object references and use the `Math` class to perform calculations and generate random numbers.
Unit 5	**Two-Dimensional Arrays** *15 45-minute lessons*	Students use two-dimensional (2D) arrays to expand storage and organization capabilities in their programs.
Unit 6	**ArrayLists and String Methods** *15 45-minute lessons*	Students use `ArrayLists` to store program data and work with methods in the `String` class to manipulate and evaluate `String` objects.
Unit 7	**Method Decomposition and Recursion** *15 45-minute lessons*	Students practice software design and development using the skills and programming constructs they have learned and explore recursion.
Unit 8	**Searching and Sorting** *15 45-minute lessons*	Students implement searching and sorting algorithms to improve the efficiency and capabilities of their programs.
Unit 9	**AP Exam Review and Practice** *15 45-minute lessons*	Students prepare for the AP CSA Exam by practicing strategies for the multiple-choice and free response questions.

CSA Vision and Goals

In 2020, underrepresented racial/ethnic groups[1] only made up 15% of AP® CSA exam takers. Female students made up 25%[2]. We aim to improve these numbers by rethinking the structure of the CSA course and provide support and resources to assist teachers in delivering equitable content.

Talking with our industry partners, we discovered that while Computer Science Principles has led to an increasingly diverse group of intern applicants, these students are less prepared for internships than their peers with CSA experience. In particular, industry partners noted that students need more practice reading, debugging, and explaining code. If CSA is perceived as a gatekeeper for Higher Ed and industry internships, there's a need for a new way to engage students who thrived in CSP but struggle with the CSA materials.

We believe our curriculum values[3] represent the Code.org curriculum brand.



Goal #1: Develop an Equitable Course

Develop Independent Learners

In the book *Culturally Responsive Teaching and the Brain*, author Zaretta Hammond discusses the importance of supporting students in becoming independent learners who can tackle challenging concepts. Hammond explains culturally responsive teaching (CRT) as:

> An educator's ability to recognize students' cultural displays of learning and meaning-making and respond positively and constructively with teaching moves that use cultural knowledge as a scaffold to connect what the student knows to new concepts and content in order to promote effective information processing. All the while the educator understands the importance of being in a relationship and having a social-emotional connection to the student in order to create a safe space for learning.[4]

[1] URG or underrepresented racial/ethnic groups refers to students from marginalized racial/ethnic groups underrepresented in computer science including students who are Black/African American, Hispanic/Latino/Latina/Latinx, Native American/Alaskan, and Native Hawaiian/Pacific Islander
[2] https://code.org/promote/ap
[3] https://code.org/educate/curriculum/values

[4] Zaretta Hammond, *Culturally Responsive Teaching and the Brain* (California: Corwin, 2015), 15.

In our curriculum, we implement strategies to support teachers and students in these goals. For example, we use the Call and Response strategy in our lesson plans to build upon the practices and traditions of many oral cultures, which may be familiar to many students from underrepresented racial or ethnic groups.

Representation

Representation matters. Students need to see examples of people like themselves in the field. Our videos feature a diverse group of presenters who work in the tech field to help students visualize opportunities for their own futures.

Goal #2: Prepare Students for Higher Ed and Industry

Software Engineering for All

We believe all students can succeed in computer science, and it is foundational for many different career paths. In this curriculum, students take on the role of software engineers and practice skills that are used in the field.

Kirn Godwin, in an article about engineering, made a statement that can be generalized to include software engineers:

> If engineering educators can find ways to help students see themselves and feel recognized by their instructors and peers as the kind of people who can do engineering, stronger identities may develop.[5]

The Software Engineering for All narrative helps students visualize themselves as software engineers through representation in our videos and exposure to common practices used in the tech industry.

Objects First

Students learn the fundamentals of object-oriented programming (OOP) first to develop a foundation for the rest of the curriculum and encourage consideration of the the overall design of programs.

Code Reviews

Code reviews are a common practice for software engineers. Our curriculum features a student-friendly version of a code review developed in consultation with our advisory councils and others in the industry. Students regularly use the process while completing assignments to help students refine their programs and learn best practices for work in the tech industry.

Goal #3: Design Relevant, Engaging Activities

Problem-Solving

We design projects and practice problems that encourage students to build skills in unpacking a prompt and solving problems creatively. The curriculum begins with an introduction to object-oriented programming(OOP) inspired by the Karel J. Robot[6] model. Using this model, students solve problems that demonstrate the usefulness of OOP and the value of a well-designed program.

---

[5] Godwin, A, Kirn, A. Identity-based motivation: Connections between first-year students' engineering role identities and future-time perspectives. J Eng Educ. 2020; 109: 362– 383.

[6] https://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html

In the article "Fostering Student Engagement by Cultivating Curiosity", Hume Green and Ladd propose the following:

> Strategies that engage and motivate students promote deeper learning and foster the development of effective problem solving and critical thinking skills.[7]

We have developed activities that tap into students' engagement and motivation and provide opportunities for self-expression, real-world problems, and addressing the societal impacts of technology.

Self-Expression

As students develop their identity as software engineers, we want them to be able to express themselves creatively through open-ended projects that connect to their interests. Student ownership of their work is an important aspect of self-expression. We want students to be proud of their work and feel prepared to share it with others as examples of their progress in the curriculum and mastery over programming concepts.

Real-World Projects

We believe students are more deeply engaged when working with projects tied to the real world, especially when that intersects with their own interests. Students investigate real-world programs during class activities and make connections to their own work. We also include open-ended projects for students to demonstrate mastery of concepts.

Addressing Societal Impacts of Technology

When developing a program, we must consider the intended and unintended consequences of the technology. Students have these discussions in the safe space of a classroom where they can share their views and learn from each other. This curriculum is designed to empower young engineers with the confidence to continue on in the field, bringing change and diverse voices.

Goal #4: Incorporate Best Practices from the Field

The Advisory Councils for the CSA curriculum provide feedback by participating in conference calls and reviewing curriculum documents and tools.

CSA Education Advisory Council

The CSA Education Advisory Council considers the curriculum objectives through the lens of Higher Education. This group consists of representatives from colleges, universities, and non-profits including Historically Black Colleges and Universities, Ivy League universities, and community colleges.

CSA Industry Advisory Council

The CSA Industry Advisory Council examines the curriculum through the lens of industry. We are most interested in learning about processes and procedures that we can translate to a student-level that will help prepare students for internships and eventually jobs in various tech fields.

[7] Hulme, E., Green, D.T. and Ladd, K.S. (2013), Fostering Student Engagement by Cultivating Curiosity. New Directions for Student Services, 2013: 53-64. https://files.eric.ed.gov/fulltext/EJ1069715.pdf

CSA State Supervisors Advisory Council

The CSA State Supervisors Advisory Council consists of supervisors and coordinators of statewide CS initiatives at state education agencies. They provide feedback from the perspective of state and district adoption of the course. We want to better understand the needs of various stakeholders in making our course available to a wide audience.

Pacing

Time will always be tight in an AP CSA course. The curriculum is designed to help you successfully teach the course in a standard school year, but careful planning and attention to schedule are important to ensure you stay on track.

Pacing Considerations

The following are important pieces of information to know as you plan the pacing of your course.

Lessons are designed for 45-minute class sessions.

The lesson plans and unit calendars assume a class that meets for 45 minutes, five days a week, for the duration of the school year. For teachers on a block or other non-traditional schedule, you will need to plan on combining or modifying lessons to fit into your school schedule. Some teachers chose to do two lessons a day on the block schedule or combine “Wrap-up” and “Warm-up” activities to make lessons fit their time. Using a calendar to plan out your year will help you adjust your schedule if lessons go longer or shorter than expected.

Plan on finishing Units 1 through 5 by the end of Semester 1.

The course is structured so that students have enough time to learn the required programming concepts before the AP CSA Exam. To stay on track, plan on finishing Units 1 through 5 by the end of the first semester, or the “halfway point” of your course if you are not on a semester system. There are 80 45-minute lessons in Units 1 through 5, including the project workdays, FRQ practice lesson, and end-of-unit multiple-choice assessment.

Learning objectives are addressed in the main activity and synthesized in the wrap-up.

The **Activity** portion of the lesson is typically core curriculum that addresses the learning objectives found at the top of each lesson plan. You should aim to not significantly cut these portions of the lessons. Depending on your students’ needs, however, you likely can alter or cut warm-ups and share-outs while still hitting these objectives. The wrap-up of many lessons contain key vocabulary and takeaways that help students process their learning and, while they may be shortened, skipping the wrap-up is not advised. In all cases the learning objectives, lesson purpose, and teaching tips are designed to help you make these decisions. The teacher forums are also useful for understanding how other teachers are approaching each lesson.

Pacing Guides assume no homework.

The curriculum does not assume that you can assign homework. This is done since many students do not have access to a computer or the Internet at home, which are necessary tools for completing the curriculum. If you are certain your students do have access to technology outside of class you may optionally choose to assign some portions of lessons as homework.

Pacing Tips

Use the tips below to help you make adjustments to the curriculum in response to your classroom's pacing needs.

Minimize Frontloading, Get to the Activity

Warm-ups are intended to be quick (usually ~5 minutes) for motivating discussions and often make no assumptions about students learning the content of the day. Avoid front-loading lectures to begin lessons, get to the main activity as quickly as possible, and save the synthesizing discussion for after.

Prioritize the Classroom Environment

Pacing considerations are obviously important, but covering every lesson in the curriculum is not the only goal. Collaborative and inquiry-based activities, especially those early in the curriculum, are designed to create a welcoming classroom environment that promotes our goal of broadening participation in computing. When making cuts, aim to preserve creative, collaborative, and exploratory activities wherever possible.

End Activities Early If Students Understand the Concept

Activities, especially unplugged activities, usually have students solve a problem that highlights a concept. Often students do not need to solve the problem fully or complete every part of an activity to understand the learning objectives of the lesson. Carefully observing students during activities will help you determine if and when you can end an activity early and move on to synthesizing discussions.

Combine Warm Ups and Wrap Ups

When teaching lessons in sequence, especially on a block schedule, it is often possible to combine the wrap up of one lesson with the warm up of the next lesson. This is one way you can save time during the school year.

Opt for the One-Day Project Instead of the Three-Day Project

Each unit includes a three-day project to provide students the opportunity to show what they learned in a meaningful and creative way. These projects assess proficiency in the knowledge and skills developed in each unit. Students participate in planning and feedback activities to solve problems, reinforce software development practices and self-image as software engineers, and improve their programs. If there is not enough time for the three-day project, a one-day option achieves the targeted learning objectives and goals. While the one-day option covers the learning objectives and goals, it shortens planning and reflection time. Consider this tradeoff when deciding which version to use.

Ask for Support on the Forum

Teaching a course for the first time can be intimidating to do alone. Luckily you have a large community of support at forum.code.org. When making pacing decisions don't hesitate to reach out on the forum where you'll be able to get advice from Code.org staff and other CSA teachers.