AP Computer Science A Syllabus

High School - One Year (135 hours)

Course Overview and Goals

AP Computer Science A introduces students to computer science through programming. Fundamental topics in this course include designing solutions to problems, using data structures to organize large sets of data, developing and implementing algorithms to process data and discover new information, analyzing potential solutions, and examining the ethical and social implications of computing systems. The course emphasizes object-oriented programming and design using the Java programming language.

The CodeHS AP Computer Science A course is a year-long course designed to help students master the basics of Java and equip them to successfully pass the College Board AP Computer Science A Exam at the end of the school year. All learning materials and resources teachers and students need for a successful year-long AP Computer Science A course can be found on the CodeHS website.

Course Background and Resources

Prerequisites

It is recommended that a student in the AP Computer Science A course has successfully completed a first-year high school algebra course with a strong foundation of basic linear functions, composition of functions, and problem-solving strategies that require multiple approaches and collaborative efforts. In addition, students should be able to use a Cartesian (x, y) coordinate system to represent points on a plane. It is important that students and their advisers understand that any significant computer science course builds upon a foundation of mathematical reasoning that should be acquired before attempting such a course.

This course is meant to be a first-time introduction to computer science and does not require students to come in with any computer programming experience. However, we recommend that students take our Introduction to Computer Science prior to our AP courses (more info at codehs.com/library). Students who have completed our Intro to CS course will be able to apply knowledge of concepts covered in the Intro course to the more advanced setting of the AP courses.

Learning Environment

The course utilizes a blended classroom approach. The content is fully web-based, with students writing and running code in the browser. Teachers utilize tools and resources provided by CodeHS to leverage time in the classroom and give focused 1-on-1 attention to students. Each unit of the course is broken down into lessons. Lessons consist of video tutorials, short quizzes, example programs to explore, and written programming exercises, adding up to over 100 hours of hands-on programming practice in total. Several units have free-response questions that have students consider the applications of programming and incorporate examples from their own lives.

Programming Environment

Students write and run Java programs in the browser using the CodeHS editor.

Technology Requirements

You can find the basic technology requirements for using the CodeHS platform in the Technical Set-Up Guide.

To complete all activities and exercises in this course, students must have access to the 3rd party sites and tools listed here: AP Computer Science A Course Links

AP Alignment

The CodeHS AP Computer Science A course aligns directly with the College Board AP Course and Exam Description (CED), with the CodeHS units and lessons following the order of the units and topics outlined in the CED. Each lesson in the course corresponds to a specific topic, and the lesson activities are designed to address the topic's Learning Objectives (LOs) and Essential Knowledge (EK) statements. By watching instructional videos, exploring examples, and completing programming exercises in every lesson, students develop a strong understanding of all of the knowledge and skills outlined in the CED.

For example, CodeHS lesson 1.1, *Introduction to Algorithms, Programming, and Compilers,* corresponds to AP topic 1.1 of the same name, and its activities comprehensively cover all LOs and EKs for that topic.

Detailed descriptions of each unit and its contents are provided in the Course Breakdown section of the syllabus.

In addition, by completing the CodeHS exercises throughout the course, students engage in over 20 hours of in-class, computer-based, hands-on lab experiences that fulfill the College Board's requirements. Samples of those activities and projects are listed in the Course Breakdown section of the syllabus.

AP Classroom and Assessment

Formative assessments are integrated throughout the CodeHS AP Computer Science A course to monitor student progress, provide timely feedback, and guide instruction to ensure mastery of key concepts and skills. Each lesson includes multiple opportunities to gauge student understanding, such as multiple-choice quizzes and coding exercises.

In addition to the formative assessments within the course, teachers can also use the resources within AP Classroom to assess student understanding. It offers tools like progress checks, topic questions, and practice exams that align closely with the AP Exam content and structure.

The CodeHS AP Computer Science A course maps 1:1 with the structure of AP Classroom, ensuring that you can seamlessly integrate resources from both platforms. For each lesson, there are corresponding resources available in AP Classroom, such as topic questions that reinforce specific concepts. For example, after completing the CodeHS lesson 4.3 *Array Creation and Access*, you can assign the topic questions for topic 4.3 in AP Classroom as formative assessments. These questions not only provide students with targeted practice on arrays and related AP Exam-style problems but also allow teachers to identify gaps in student understanding. Teachers can use the results of these assessments to inform their instruction by revisiting and reviewing concepts as needed to ensure students have a solid grasp of the material before moving forward.

When to Use AP Classroom

- **Topic Questions**: Assign topic questions after every lesson to provide formative assessments that target specific skills. These questions align with the material in each CodeHS lesson and give students additional practice with AP-style questions. Teachers can use the results of these assessments to identify gaps in student understanding and revisit or review concepts as needed, ensuring students build a strong foundation before progressing.

- **Progress Checks**: Use progress checks as more comprehensive formative assessments during mid-unit or end-of-unit reviews. These checks are designed to gauge students' understanding of broader topics and provide actionable insights for teachers. By analyzing the results, teachers can identify areas where further review might be needed and address gaps in understanding before moving on to practice exams or more summative assessments.
- **Practice Exams**: As the AP Exam approaches, utilize the full-length practice exams that are available on AP Classroom. These exams familiarize students with the test format, timing, and content, serving as valuable summative assessments that prepare them for exam day.

Course Breakdown

Unit 1: Using Objects and Methods (6-7 weeks or 30-35 hours)

Students are introduced to fundamental concepts that form the foundation of object-oriented programming. The unit begins with an overview of algorithms, programming, and compilers, setting the stage for understanding how software is developed and executed. Students then delve into variables, data types, and expressions, learning how to store, manipulate, and display data. Key programming constructs such as assignment statements, input handling, and compound assignment operators are explored to build fluency in coding.

The unit progresses to more advanced topics like casting and understanding variable ranges, enabling students to manage data efficiently. They also learn to leverage Application Programming Interfaces (APIs) and libraries, emphasizing reusability and modularity in programming. Proper documentation through comments is introduced to encourage maintainable code practices.

Students develop a strong grasp of object-oriented programming by studying method signatures, class methods, and object instantiation. Lessons on instance methods and string manipulation demonstrate how objects and methods can be used to solve real-world problems effectively. The Math class is introduced to perform complex calculations, rounding out their knowledge of built-in utilities. Two assessments, a mid-unit, and an end-of-unit assessment, provide opportunities to consolidate learning and evaluate understanding.

Topics Covered

- Introduction to Algorithms, Programming, and Compilers
- Variables and Data Types
- Expressions and Output
- Assignment Statements and Input
- Casting and Range of Variables
- Compound Assignment Operators
- Application Program Interface (API) and Libraries
- Documentation with Comments
- Method Signatures
- Calling Class Methods
- Math Class
- Objects: Instances of Classes
- Object Creation and Storage (Instantiation)
- Calling Instance Methods
- String Manipulation
- Mid-Unit Assessment
- End of Unit Assessment

Sample Projects, Activities & Assignments

- **Exploration: Hello World:** Students carefully read the order of the print statements, predict their outputs, and then confirm their prediction by running the program. They modify the code by removing quotation marks, removing blank lines, and adding an empty print statement, each time observing how the execution order of statements dictates the final output. By going through these steps and recording their findings, students practice analyzing the flow of an algorithm and determining the result of its statements based on the order in which they are executed. (Skill 3.A)
- **Expanded Rectangle Class:** students work with the Rectangle class, which encapsulates its data (width and height) and includes instance methods to manipulate these properties. By investigating and running the RectangleRunner code, they see how each method call changes the rectangle's dimensions or makes related calculations. (Skill 3.B)
- **Debugging: Quotes:** Students review a buggy program and identify why it does not compile or run as expected. They systematically locate syntax mistakes (e.g., missing semicolons, incorrect spellings, unmatched quotation marks) and logic errors by examining compiler error messages and comparing the actual output to the expected results. They then correct each mistake, explaining why the bug occurred and how they fixed it. (Skill 3.D)
- **Bank Account Comments:** Students examine a simple bank account program that simulates deposits and withdrawals, then add comments describing each part of the code. By explaining the steps—such as setting an initial balance, adding or subtracting money, and printing the resulting balance—they clearly describe how the program behaves at each stage. (Skill 4.A)
- **Greeter Javadocs:** Students write Javadoc comments for the greetEnglish and

introduce methods, clarifying each of their pre and postconditions. They specify the preconditions by explaining the circumstances needed (e.g., valid strings) for the methods to work as intended. (Skill 4.B)

Summative Assessments

- Mid-Unit Assessment (Lessons 1.1-1.8)
- End of Unit Assessment (Lessons 1.1-1.17)

Unit 2: Selection and Iteration (4-5 weeks or 20-25 hours)

Students expand their programming skills by exploring decision-making and repetitive structures. The unit begins with algorithms that incorporate selection and repetition, providing a conceptual foundation for building dynamic and flexible programs. Students then dive into Boolean expressions and learn how they underpin decision-making in programming, including their use in if and nested if statements.

As the unit progresses, students explore compound Boolean expressions and practice comparing them to optimize decision-making processes. They learn how to use looping structures such as while and for loops to implement repetition in their code, enabling them to solve more complex problems efficiently. Lessons on nested iteration introduce students to scenarios requiring multiple layers of repetition, further enhancing their problem-solving toolkit.

The unit also includes practical applications of these concepts, such as implementing selection and iteration algorithms and working with strings in iterative contexts. Finally, students are introduced to informal run-time analysis, helping them assess the efficiency of their algorithms. This provides a comprehensive introduction to control flow, preparing students to tackle increasingly sophisticated programming challenges.

Topics Covered

- Algorithms with Selection and Repetition
- Boolean Expressions
- if Statements
- Nested if Statements
- Compound Boolean Expressions
- Comparing Boolean Expressions
- while Loops
- for Loops
- Implementing Selection and Iteration Algorithms
- Implementing String Algorithms

- Nested Iteration
- Informal Run-Time Analysis

Sample Projects, Activities & Assignments

- **Dice Frequencies:** To complete this program, students must write an algorithm that simulates die rolls and counts the frequencies of each output. They gather user input (number of rolls), generate random die values, and update frequency counts for each side of the die. This process requires students to design and code the logic—using loops and conditional checks—to track how many times each side appears. (Skill 2.A)
- **Finding Palindromes:** Students design and code two methods that implement the algorithm for checking whether a given string is a palindrome: one method to reverse the string and another to compare the reversed result with the original. (Skill 2.A)
- **Exploration: Infinite Loops:** Students examine three infinite loops to understand why they never terminate, then apply different strategies—adjusting loop conditions, using break and return statements—to fix each one so it works as intended. By investigating the loops, identifying and correcting the root cause of the errors, students learn how to troubleshoot infinite loops and choose the most suitable approach for the problem's context. (Skill 3.D)

Summative Assessments

- Mid-Unit Assessment (Lessons 2.1-2.6)
- End of Unit Assessment (Lessons 2.1-2.12)

Unit 3: Class Creation (4-5 weeks or 20-25 hours)

Students develop a deep understanding of designing and creating their own classes, an essential aspect of object-oriented programming. The unit begins with abstraction and program design, introducing the importance of modeling real-world systems in software. Students explore the impact of program design, focusing on creating efficient, maintainable, and scalable code. They then study the anatomy of a class, breaking down its structure and components.

Key concepts such as constructors and method writing are covered, with students learning how to initialize objects and define behaviors within their classes. Lessons on passing and returning object references deepen their understanding of method functionality and object interaction. The unit also covers class variables and methods, demonstrating how shared properties and behaviors can be implemented efficiently.

To ensure clarity and proper encapsulation, students explore the concepts of scope and access modifiers, learning how to manage visibility and control over class members. The ‘this’ keyword is introduced as a tool for referring to the current object, solidifying their grasp of object-oriented principles. This unit equips students with the knowledge and skills to design and implement custom classes, laying a foundation for advanced programming techniques.

Topics Covered

- Abstraction and Program Design
- Impact of Program Design
- Anatomy of a Class
- Constructors
- Methods: How to Write Them
- Methods: Passing and Returning References of an Object
- Class Variables and Methods
- Scope and Access
- this Keyword

Sample Projects, Activities & Assignments

- **Lab #1** - **Dragon Class: Accessor and Mutator** and **Dragon Class: Full Implementation:** In these lab activities, students first implement core attributes and basic behaviors (constructor, getters, and setters) in their Dragon class, ensuring their program follows good object-oriented design principles. They then extend the same class with additional methods—like gainExperience, attack, and getDefenseRating—to handle more complex functionality. Throughout the process, they explore different solutions, refine their code by testing and debugging, and compare possible approaches for managing a dragon’s abilities.
- **Public Library System**: Students begin this activity by clearly identifying the problem—tracking books and members across multiple library branches—before determining the objects (library branches, books, and library members) that will make up their program. They design classes for each object, listing relevant attributes (like titles, member names, and branch addresses) and methods (such as borrowBook, returnBook, etc.), and create a diagram to show how the classes relate to one another. (Skill 1.A)
- **In the Tournament?:** Students write program code that uses data abstractions by defining class variables (totalTeams and MIN_TEAMS) and an instance variable (name) in a Team class. They then instantiate multiple Team objects in the TeamRunner class, which check if the minimum team requirement is met to start the tournament. By designing and managing these variables, students practice using data abstractions in their code and see how those abstractions directly determine the program’s final output. (Skill 2.B)(Skill 3.B)
- **Distance Conversions:** Students break down the problem of unit conversion into separate methods (toMiles, toYards, toLeagues, and more) in the KiloDistance class. Each method encapsulates its own task, hiding the implementation details

while providing a clear way to convert kilometers into different units. The DistanceRunner class calls these methods to convert and display the results for a particular distance. By organizing the code into smaller, specialized procedures, students demonstrate the principle of procedural abstraction. (Skill 2.C)
- **Point Utilities Class:** Students implement two class methods (movePoint and resetPoint) in the PointUtility class, each defining a specific procedure for manipulating a Point object's coordinates. They then call these methods in the PointRunner class to move a point and reset its location. (Skill 2.C)
- **Drone Delivery: Potential Impact:** Students analyze how the introduction of drone delivery affects consumers, businesses, and communities from economic, social, and ethical perspectives. They identify both positive and negative consequences—such as increased efficiency, reduced emissions, possible job displacement, and privacy concerns—and then propose ways to mitigate negative outcomes. By reflecting on who might benefit or be harmed and considering broader issues like equality and ethics, students gain a clearer understanding of how computing technologies, such as drones, can influence society, the economy, and culture. (Skill 5.A)
- **Real World System Reliability Responses:** Students examine real-world cases where glitches in financial applications (Cash App and Chase) led to significant user and corporate challenges—from unexpected charges and negative account balances to damage control efforts and legal concerns. By reflecting on these incidents, learners see firsthand how even small coding errors can have large-scale impacts on individuals' finances, corporate reputation, and overall consumer trust. (Skill 5.A)

Summative Assessments
- Mid-Unit Assessment (Lessons 3.1-3.4)
- End of Unit Assessment (Lessons 3.1-3.9)

Unit 4: Data Collection (8-9 weeks or 40-45 hours)

Students explore various data collection structures and algorithms, focusing on their creation, manipulation, and application. The unit begins with a discussion on ethical and social issues surrounding data collection, encouraging students to consider the broader implications of working with data. This foundation is followed by an introduction to using data sets and understanding their importance in programming.

Students learn to work with arrays, including their creation, access, and traversal. Lessons on implementing array algorithms build problem-solving skills through practical applications. They expand their knowledge by learning to read from and write to text files, a crucial skill for handling external data sources. Wrapper classes and the ArrayList

structure are introduced, providing tools for more flexible and dynamic data storage. Students practice using ArrayList methods, traversals, and algorithms, further enhancing their understanding of collections.

The unit also covers advanced topics like 2D arrays, including their creation, traversal, and algorithm implementation. Students study searching and sorting algorithms, gaining insight into efficient data manipulation techniques. Finally, recursion and recursive algorithms for searching and sorting are introduced, allowing students to solve complex problems using this powerful technique. This unit provides a comprehensive exploration of data collections, preparing students to handle diverse programming challenges.

Topics Covered

- Ethical and Social Issues Around Data Collection
- Introduction to Using Data Sets
- Array Creation and Access
- Array Traversals
- Implementing Array Algorithms
- Using Text Files
- Wrapper Classes
- ArrayList Methods
- ArrayList Traversals
- Implementing ArrayList Algorithms
- 2D Array Creation and Access
- 2D Array Traversals
- Implementing 2D Array Algorithms
- Searching Algorithms
- Sorting Algorithms
- Recursion
- Recursive Searching and Sorting

Sample Projects, Activities & Assignments

- **Lab #2 - Coffee Shop:** Students simulate a coffee shop inventory system by working with multiple classes: CoffeeShopRunner, CoffeeBag, and CoffeeShop. By designing methods to add coffee inventory, process sales, and display current stock, students develop a program for a real-life situation, refining and testing their solutions in order to satisfy the desired needs of a business.
- **Lab #3 - FitBit: Read in Sleep Data** and **FitBit: Analyze Sleep Data:** In the first lab activity, students write a program that reads from a file containing participant sleep data, calculates hours slept for each entry, and then prints a summary. This lays a foundation of reading data from an external source, structuring it in a meaningful way, and formatting the output. In the second lab activity, they extend their solution by storing sleep values in an array, calculating an average, and determining if the participant is getting enough sleep. Across both tasks, they

practice reading and understanding existing starter code, adding new functions, testing outcomes, and comparing strategies to ensure accurate data handling.

- **Lab #4 - X-Y Graphing:** Students work with a pair of classes to create an XYPlot that detects duplicates, verifies positive coordinates, and displays data points in a simple ASCII-based chart. They start from existing starter code, then design and implement solutions for several new methods, testing different loop strategies and data checks along the way. By experimenting with coordinate input, spotting and fixing logical or indexing errors, and comparing how various implementations handle the data, students deepen their understanding of problem-solving in a meaningful context.
- **Choosing an Appropriate Data Set:** Students examine a dataset of FitBit data and decide which of several questions can be answered based on the data provided. They see that if the dataset includes specific information (e.g., heart rate, sleep measures) then it can potentially answer certain questions, but if information is missing (e.g., weather data, diet information), those questions cannot be addressed. By assessing which questions can or cannot be answered, students learn to determine what knowledge can be extracted from the dataset. (Skill 1.B)
- **Everyday Life Data Set: Algorithm:** Students pick a question about their dataset and outline a clear set of steps to extract meaningful information from it. They review their data, decide what inquiry they want to make, and then detail how they would systematically gather, compute, and interpret the relevant values. (Skill 1.B)
- **Reverse the Playlist:** Students create and implement a reverse method that systematically reorders the songs in a playlist to reverse their positions. They then test the algorithm by creating a Playlist object, printing its current song order, calling the reverse method, and printing the playlist again to confirm the result. (Skill 2.A)
- **Exploration: Numerical ArrayList Algorithms:** Students explore three “mystery” methods in the EarningsAnalysis class, each representing a different procedural abstraction. They make predictions about the methods’ outputs, run the program to observe the results, and then document their findings. By analyzing how each method manipulates or filters an ArrayList of earnings, students actively determine and describe the program’s output (as a result of each method call). (Skill 3.C)(Skill 4.A)
- **Exploration: String ArrayList Algorithms:** Students investigate three “mystery” methods in the LineOrderAlgorithms class, each serving as a procedural abstraction that manipulates the order of students in a line. Students then predict what each method will do, run the program to observe the actual results, and test their understanding by running various scenarios. Through this process, they determine how each method affects the output and explain the specific steps that produce those changes. (Skill 3.C)(Skill 4.A)

Summative Assessments

- Mid-Unit Assessment (Lessons 4.1-4.10)
- End of Unit Assessment (Lessons 4.1-4.17)

Unit 5: Exam Review (2-4 weeks or 10-20 hours)

Students prepare for the AP Exam by practicing multiple-choice and free-response questions. Students should assess areas or topics that they need to practice.

Topics Covered

- Students know what to expect on the AP Exam
- Practice solving AP Exam type multiple choice questions
- Practice solving AP Exam type free response questions