Code.org Computer Science A Syllabus and Overview

AP® Computer Science A Overview

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.

AP Endorsed

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

Prerequisites

The Code.org CSA curriculum is recommended for any high school student who wishes to continue their computer science education after completing an introductory course such as Computer Science Principles (CSP) or Computer Science Discoveries (CSD).

Additionally, the College Board makes the following recommendation:

> 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 advisors understand that any significant computer science course builds upon a foundation of mathematical reasoning that should be acquired before attempting such a course.[1]

[1] College Board. AP Computer Science A Course and Exam Description, page 7

Code.org Computer Science A Syllabus and Overview

Our Vision

Code.org's vision is that every student in every school should have the opportunity to learn computer science (code.org/about). Our curriculum is designed so that an empowered teacher can lead a diverse group of students through experiences that are supportive, equitable, engaging, and lead to valuable learning (code.org/educate/curriculum/values).

Historically, this vision has contrasted sharply with reality. Until recently, most schools did not offer computer science, and schools that did offer computer science did not have enrollment that matched the demographics of their school population. Additionally, many students found these classes unengaging, intimidating, or disconnected from their lived experiences with technology. Thanks to efforts by many organizations and individuals, this world is beginning to change: many more schools now offer computer science courses; the diversity of schools enrolled in those courses is increasing; and more engaging, relevant, and equitable pedagogy has become the established norm. Even so, there is much work still to be done. This course is designed to continue this momentum as the collective CS education community moves towards this vision of an equitable CS education system.

How We Support Our Vision

Many aspects of Code.org's CSA curriculum are designed to bring about the eventual change we aim to see more broadly in CS education. Some of the most significant features are listed below.

Free and Open: We make our curriculum, videos, and tools free and open for anyone to adopt.

Prioritize teachers who are new to Java: Historically, only a few schools could hire trained computer scientists as teachers, which severely limited which schools could offer a CS course. Reaching all schools has meant developing our CSA course with the understanding that most of our teachers are new-to-CS and prioritizing their needs. As such, our curriculum includes some distinctive features:

- Comprehensive lesson plans and resources designed to provide new-to-CS teachers the tools they need to teach the course.
- Clear and consistent pedagogy to help teachers develop best practices.
- High-quality videos that help teachers introduce and explain Java and software engineering concepts.
- A professional learning program designed to target the needs of teachers.

Equitable Pedagogy: Our curriculum is designed to promote an equitable classroom environment for all students, with a focus on the experiences of young women and students from underrepresented groups in computing. Drawing from extensive feedback from our classrooms, as well as CS education research, our curriculum includes many features designed to support and prioritize these students:

- Pedagogy that develops a collaborative and supportive classroom environment
- Projects and activities that highlight a variety of applications of computing and frequently ask students to incorporate their own backgrounds and interests
- Curriculum videos that feature a cast of diverse role models in terms of race, gender, and profession who empower our diverse students to see themselves as part of the world of computing
- A professional learning program that highlights these features and helps teachers reflect on how best to implement them within their own classroom

Materials and Resources

The curriculum provides a comprehensive set of resources for the teacher, including detailed daily lesson plans, engaging activities and projects, formative and summative assessments, computing tools that are designed for learning specific concepts, and the Java Lab programming environment. These resources have been specifically curated to provide a unified experience for teachers and students. Together, these resources allow the teacher to act as a facilitator and coach for their students when addressing unfamiliar material. When the teacher acts as the primary source of information, generous support is provided.

All resources below can be accessed free of charge at code.org/csa.

Lesson Plans

The following resources and information can be found in each lesson plan:

- Instructions and teaching tips for conducting the lesson
- Unit Guides, activity guides and handouts, and Extra Practice for students
- Lesson slide decks
- Formative and summative assessments
- Answer keys, exemplars, and rubrics

Videos

The Code.org CSA curriculum includes videos that provide:

- Content instruction
- Software Engineering series
- Lesson and tool tutorials

Programming Environment and Tools

The following programming environment and tools are used in the curriculum:

- **Java Lab –** Code.org's Java programming environment for developing programs
- **The Neighborhood –** a package available in Java Lab to navigate mazes and create drawings
- **The Theater –** a package available in Java Lab to develop short animations with images and drawings, manipulate image pixels to create filters, and sound effects
- **Code Review –** a student-friendly version of code review tools used in the industry to allow students to easily read and share feedback on each other's code

Code.org Computer Science A Syllabus and Overview

Textbook

The curriculum is accessible online at studio.code.org/courses/csa.

Students who are new to computer science and programming may need additional support and practice beyond what is available in the Code.org CSA curriculum. Additionally, students who are absent one or more class periods may struggle to catch up on the content that they missed.

The following online textbooks can also be used as additional resources to support student learning and for students to catch up on content that they missed:

- CSAwesome AP CSA Java (online), course.csawesome.org
- CodeHS AP Computer Science A (online), codehs.com/textbook/apcsa_textbook

The following resources are available to support students who need additional practice:

- CodingBat (online), codingbat.com/java
- Open sandbox levels at the end of each unit

Getting Verified

A verified teacher is a Code.org teacher account that we can prove belongs to a teacher. You must become a verified teacher to use Java Lab and the CSA curriculum. By becoming verified, you will get access to answer keys, project exemplars, and Java Lab. Until you become verified, neither you nor your students will be able to run any Java code on Code.org. Once you are verified, your students must be assigned to one of your CSA sections on Code.org to be able to run any code on Java Lab.

You can become a verified teacher by:

- **Attending Code.org Professional Development for our CSA curriculum.** This process should happen automatically once you have attended Professional Development.
- **Being manually verified as an actual teacher.** If you have not gone through our Professional Development, you can apply to become verified by filling out the form at code.org/csa. We manually review each response - this process takes on average one business day. You will receive an email once you are verified successfully. If you don't hear back from us after a few business days, contact us at support@code.org.

Technical Requirements for CSA

The curriculum requires and assumes a 1:1 computer lab or a setup such that each student in the class has access to an Internet-connected computer every day in class. All curriculum tools and resources are available online. Tablets are not currently supported. For more details on the technical requirements, please visit code.org/educate/it.

In addition, the Code.org CSA course uses a communication standard called WebSockets, which may be blocked by some school systems' networks or device policies. Before teaching this course, please visit code.org/educate/websocket to test that your school's network system will support WebSockets.

Code.org's CSA curriculum does not require you to download any programs on your or your students' computers. The CSA curriculum only requires computers that have browser access to Code.org from Chrome, Edge, Firefox, or Safari. Internet Explorer is not supported.

While the curriculum features many unplugged activities designed to be completed without a computer, daily access to a computer is essential for every student. The curriculum is developed to be completed within the classroom – no homework or after-hours computer access is assumed.

Additional Materials and Supplies

One potentially significant cost to consider is printing. Many lessons have handouts that are designed to guide students through activities. While it is not required that all of these handouts be printed, many were designed to be printed, and we highly recommend their use.

Beyond printing, some lessons call for typical classroom supplies and manipulatives such as:

- poster paper
- markers or colored pencils
- sticky notes

Suggested substitutes can be found in individual lesson plans.

Code.org Computer Science A Syllabus and Overview

AP CSA Framework

The AP CSA Framework developed by the College Board outlines four **Big Ideas**, each consisting of Enduring Understandings, Learning Objectives, and Essential Knowledge statements.

MOD	VAR	CON	IOC
Modularity	Variables	Control	Impact of Computing

Additionally, the framework identifies five **Computational Thinking Practices**, each outlining skills that students should develop throughout the course. The Computational Thinking Practices form the basis of tasks on the AP Exam.

CTP1	CTP2	CTP3	CTP4	CTP5
Program Design and Algorithm Development	Code Logic	Code Implementation	Code Testing	Documentation

A Layered Approach

The Code.org CSA curriculum covers the content in a layered approach that is different from the outline provided in the AP CSA Course and Exam Description. This structure allows students to develop understanding of the content within the Big Ideas while developing the skills outlined in the Computational Thinking Practices. Students learn the fundamentals of object-oriented programming (OOP) first, giving students a foundation for the rest of the course and encouraging students to consider the overall design of their programs. You can learn more about our approach at tinyurl.com/csaobjectsfirst.

Code.org Computer Science A Syllabus and Overview

AP Computer Science A Curriculum at a Glance

Unit 1

wk 1	Welcome to CSA Java Lab Classes and Objects Instantiating Objects Methods
2	Methods with Parameters Loops Inheritance Writing Methods Programming Style and Feedback
3	Selection Statements Debugging Strategies Decomposition and Design Two-Way Selection Statements
4	Asphalt Art Project SpiralPainterFRQ Unit 1 Assessment

Unit 2

wk 1	Attributes No-Argument Constructors Parameterized Constructors The this Keyword Constructors and Inheritance
2	Variables Accessor Methods Operators and Expressions Mutator Methods Printing Objects
3	Store Management Project Burger Class FRQ Unit 2 Assessment

Unit 3

wk 1	One-Dimensional (1D) Arrays Modifying Elements Traversing 1D Arrays For Loops Preconditions and Postconditions
2	Polymorphism Enhanced For Loops Array Algorithms Finding Duplicates
3	Data for Social Good Project TicketTracker FRQ Unit 3 Assessment

Unit 4

wk 1	The Theater Static Variables and Methods The Math Class Casting and Rounding Random
2	Object Aliases and Equality Nested If Statements Logical Operators De Morgan's Laws Multi-Selection Statements
3	Abstract Data Art Project BingoCaller FRQ Unit 4 Assessment

Unit 5

wk 1	Two-Dimensional (2D) Arrays 2D Array Elements Row-Major Traversal Column-Major Traversal Enhanced For Loops
2	Images in The Theater 2D Array Algorithms Modifying Images Impacts of Programs
3	Personal Narrative Project GrayscaleImage FRQ Unit 5 Assessment

Unit 6

wk 1	Project Planning Substrings Integer and Double Objects ArrayLists Manipulating Elements
2	Comparing Strings Lists of Objects Removing Elements ArrayList and String Algorithms
3	Natural Language Processing Project TemperatureAction FRQ Unit 3 Assessment

Unit 7

wk 1	Project Planning Object References as Parameters Overloading Methods Private Methods
2	Overriding Methods Intellectual Property Recursion Project Development System Reliability
3	Creative Coding with The Theater Project MusicQueue FRQ Unit 7 Assessment

Unit 8

wk 1	Project Planning Searching Binary Search Selection Sort Insertion Sort
2	Merge Sort Searching and Sorting Project Development Privacy and Security
3	Creative Coding with the Console Project SeatingChart FRQ Unit 8 Assessment

Unit 9

wk 1	The AP CSA Exam MCQ Pre-Assessment MCQ Study Plan MCQ Practice
2	FRQ Pre-Assessment FRQ Study Plan FRQ Practice Mock MCQ Exam
3	Mock MCQ Exam Mock FRQ Exam

Code.org Computer Science A Syllabus and Overview

Curriculum Outline

The curriculum is divided into nine units – eight content units and one review unit. The last week of each unit makes up a **Show What You Know** week, which consists of a three-day project, FRQ practice, and a unit assessment.

The following outlines the content of each unit, including the associated big ideas and computational thinking practices that are developed.

Unit 1: Object-Oriented Programming

20 class periods

Big Ideas MOD-1, MOD-2, MOD-3

Computational Thinking Practices 4.B, 5.A, 5.B

CED Units and Topics

1.1 Why Programming? Why Java?
1.2 Variables and Data Types
2.1 Objects: Instances of Classes
2.2 Creating and Storing Objects (Instantiation)
2.3 Calling a Void Method
2.4 Calling a Void Method with Parameters
2.5 Calling a Non-Void Method
2.6 String Objects: Concatenation, Literals, and More
3.2 if Statements and Control Flow
3.3 if-else Statements
3.5 Compound Boolean Expressions
4.1 While Loops
5.1 Anatomy of a Class
5.3 Documentation with Comments
5.4 Accessor Methods
5.8 Scope and Access
9.1 Creating Superclasses and Subclasses
9.5 Creating References Using Inheritance Hierarchies

This unit introduces students to object-oriented programming principles as they explore The Neighborhood and discover their identity as a software engineer. Students learn fundamental Java concepts as they navigate and paint in The Neighborhood with `Painter` objects and extend the `Painter` class to expand the capabilities of their programs. Students practice predicting the outcome of program code and developing algorithms using sequencing, selection, and iteration to navigate mazes and paint murals. Students also learn to document program code using comments to describe the behavior of specific code segments and conduct code reviews to receive feedback from their peers.

Unit 2: Class Structure and Design

15 class periods

Big Ideas MOD-1, MOD-2, MOD-3, VAR-1, CON-1

Computational Thinking Practices 2.A, 2.B

CED Units and Topics

1.2 Variables and Data Types
1.3 Expressions and Assignment Statements
1.4 Compound Assignment Operators
2.2 Creating and Storing Objects (Instantiation)
2.3 Calling a Void Method
2.4 Calling a Void Method with Parameters
2.5 Calling a Non-Void Method
2.6 String Objects: Concatenation, Literals, and More
3.1 Boolean Expressions
5.1 Anatomy of a Class
5.2 Constructors
5.4 Accessor Methods
5.5 Mutator Methods
5.8 Scope and Access
5.9 this Keyword
9.1 Creating Superclasses and Subclasses
9.6 Polymorphism
9.7 Object Superclass

This unit expands on the object-oriented programming principles introduced in Unit 1 to explore design principles as students develop classes with attributes and behaviors and work with primitive and object data. Students learn to write no-argument constructors to assign default values and parameterized constructors to assign specific values to an object's instance variables. They explore how the **`this`** keyword can be used to reduce ambiguity and redundancies in their program and how the super keyword can be used to call a superclass constructor or method. Students also learn how to work with variables and write expressions using arithmetic and compound assignment operators and practice tracing code segments to determine the output. After working with instance variables and constructors, they write accessor and mutator methods to work with the values assigned to an object's instance variables and **`toString()`** methods to display information about an object to the console. Throughout this unit, students continue to develop software engineering skills as they learn to make design decisions and use inheritance to create class hierarchies.

Unit 3: Arrays and Algorithms

15 class periods

Big Ideas MOD-2, MOD-3, VAR-2, CON-2

Computational Thinking Practices 4.C, 5.D

CED Units and Topics

1.4 Compound Assignment Operators
2.2 Creating and Storing Objects (Instantiation)
4.1 While Loops
4.2 For Loops
4.4 Nested Iteration
5.3 Documentation with Comments
5.6 Writing Methods
6.1 Array Creation and Access
6.2 Traversing Arrays
6.3 Enhanced For Loops with Arrays
6.4 Developing Algorithms with Arrays
9.1 Creating Superclasses and Subclasses
9.5 Creating References Using Inheritance Hierarchies
9.6 Polymorphism
9.7 Object Superclass

This unit introduces students to data structures to store primitive values and object references. Students use one-dimensional (1D) arrays to store multiple related values while expanding their knowledge of loops and conditionals to analyze and process data in a 1D array. Students learn to use **`for`** loops to traverse arrays and discover that an algorithm involving loops can be implemented with either a **`for`** loop or a **`while`** loop. Throughout the unit, students develop and modify algorithms to find and manipulate elements in a 1D array while also discovering the concept of polymorphism when traversing arrays of objects. While developing algorithms, students identify preconditions and postconditions and implement solutions to ensure that these conditions are satisfied. Students continue to develop software engineering skills as they learn to make design decisions and use 1D arrays to store and analyze data.

Unit 4: Conditions and Logic

15 class periods

Big Ideas MOD-1, MOD-2, CON-1, CON-2

Computational Thinking Practices

CED Units and Topics

1.2 Variables and Data Types
1.5 Casting and Ranges of Variables
2.6 String Object: Concatenation, Literals, and More
2.9 Using the Math Class
3.4 else if Statements
3.5 Compound Boolean Expressions
3.6 Equivalent Boolean Expressions
3.7 Comparing Objects
4.1 While Loops
5.7 Static Variables and Methods

This unit expands on the use of APIs, object-oriented programming concepts, and conditional statements to develop visuals and animations using The Theater. Students learn about the functionality of the static keyword and explore the methods in the **`Math`** class to perform calculations and incorporate randomness in program decisions and behaviors. While working with conditional statements and Boolean expressions, students realize the difference between using the `==` operator and the **`equals()`** method to compare objects for equality and discover the need for overriding the **`equals()`** method in their own classes. They deepen their understanding of conditional statements and logical operators as they learn to write nested conditional statements, use the AND (**`&&`**) and OR (**`||`**) operators, and write multi-selection statements to test multiple conditions. Using their knowledge of Boolean expressions and logical operators, they practice evaluating truth tables to compare two expressions for equivalence and applying De Morgan's Laws to simplify expressions.

Unit 5: Two-Dimensional Arrays

15 class periods

Big Ideas VAR-2, CON-2, IOC-1

Computational Thinking Practices 1.B, 1.C

CED Units and Topics

2.6 String Objects: Concatenation, Literals, and More
5.4 Accessor Methods
6.4 Developing Algorithms Using Arrays
8.1 2D Arrays
8.2 Traversing 2D Arrays

This unit expands on data structures introduced in Unit 3 to create tables of data using two-dimensional (2D) arrays. Students identify similarities and differences between 1D and 2D arrays when creating, accessing, and traversing 2D arrays and apply standard algorithms to find and manipulate elements. As students analyze problems involving 2D arrays, they revisit these standard algorithms to determine what code needs to be added or modified and to interact with completed program code. Students apply these concepts to manipulate pixels and in The Theater to create image filters in addition to working with primitive values and various object references. Additionally, students use the programming knowledge and skills they have acquired to consider the impacts of programs on society, economies, and culture.

Unit 6: ArrayLists and String Methods

15 class periods

Big Ideas VAR-1, VAR-2, CON-2

Computational Thinking Practices

CED Units and Topics

1.5 Casting and Ranges of Variables
2.6 String Objects: Concatenation, Literals, and More
2.8 Wrapper Classes: Integer and Double
4.3 Developing Algorithms Using Strings
5.3 Documentation with Comments
7.1 Introduction to ArrayList
7.2 ArrayList Methods
7.3 Traversing ArrayLists
7.4 Developing Algorithms Using ArrayLists

This unit continues to expand on data structures to introduce students to creating lists using the **`ArrayList`** class. In the process, students learn about the **`Integer`** and **`Double`** classes and use their methods to parse data from text files and explore the limits of integer values. Students differentiate between when to use each type of data structure while learning about the structure and functionality of an **`ArrayList`**. Students apply standard algorithms to find and manipulate data in an **`ArrayList`** of numerical and object data. Throughout the unit, students learn to use the **`String`** class to analyze and process text obtained from a user and from file input while learning about basic natural language processing techniques and applications. Additionally, students further develop software engineering skills by writing Javadoc comments to create API documentation for their programs.

Unit 7: Method Decomposition and Recursion

15 class periods

Big Ideas MOD-1, MOD-2, MOD-3, CON-2, IOC-1

Computational Thinking Practices 2.C

CED Units and Topics

2.4 Calling a Void Method with Parameters
5.1 Anatomy of a Class
5.2 Constructors
5.6 Writing Methods
5.9 this Keyword
9.1 Creating Superclasses and Subclasses
10.1 Recursion

This unit allows students to practice software design and development using the skills they have learned throughout the curriculum while planning and developing a creative coding project to convey a personal interest or story using The Theater. Students use decomposition strategies and object-oriented principles to plan and implement their ideas while ensuring their projects meet specified requirements. In the process, students learn to write **`private`**, overloaded, and overridden methods and use the **`super`** keyword in a subclass method to call a superclass method while exploring the functionality of methods and their parameters. Throughout the unit, students practice tracing and writing recursive methods and comparing these methods to iterative solutions. With the knowledge and skills acquired throughout the year, students consider the need for maximizing system reliability as they explore bugs and issues in existing programs.

Unit 8: Searching and Sorting

15 class periods

Big Ideas CON-2, IOC-1

Computational Thinking Practices 2.D

CED Units and Topics

4.5 Informal Code Analysis
7.4 Developing Algorithms Using ArrayLists
7.5 Searching
7.6 Sorting
8.2 Traversing 2D Arrays
10.2 Recursive Searching and Sorting

This unit expands on algorithms students have learned to introduce common approaches to searching and sorting 1D and 2D arrays and **`ArrayList`**s. In the process, students analyze and compare the efficiencies of these algorithms using statement execution counts and further develop problem-solving skills to decompose complex problems. Throughout the unit, students apply their programming and software engineering skills to plan and develop a creative coding project using the console that incorporates object-oriented design, data structures, and algorithmic thinking. With the knowledge and skills acquired throughout the year, students consider the privacy and security of programs and users.

Unit 9: AP Exam Review and Practice

15 class periods

This unit prepares students for the AP CSA Exam by reviewing key concepts, practicing multiple-choice and free response questions, and strengthening test-taking strategies. Students identify strengths and areas of improvement to create individualized study plans to focus their practice and self-assess their progress.

Lab Requirement

Students in AP Computer Science A must engage in a minimum of 20 hours of hands-on, structured lab experiences to engage students in individual or group problem-solving. The Code.org CSA curriculum provides students opportunities to design solutions to problems, express their solutions precisely in the Java programming language, test their solutions, identify and correct errors, and compare possible solutions.

Using this curriculum, students will exceed the 20-hour in-class programming requirement. In addition to writing dozens of programs throughout the year, students will also complete a larger programming project at the end of each unit.

Unit 1: Asphalt Art Project

Big Ideas MOD-1, MOD-2, MOD-3, CON-2 Computational Thinking Practices 3.A, 3.B

Students create asphalt art in The Neighborhood using the **`Painter`** classes created throughout the unit and writing a new **`Painter`** class that they develop for the project. In the process, students continue to expand their hierarchy of **`Painter`** classes which share attributes and behaviors but have specific types of behaviors in each subclass. Students choose a theme or concept for their asphalt art representing something they are interested in or that is meaningful to them. After brainstorming and planning, students develop their programs to create their designs by creating one or more **`Painter`** objects and calling their methods. Students use conditional statements and **`while`** loops while incorporating the **`!`** (NOT) logical operator and Boolean expressions to manage navigation and painting for their design.

Unit 2: Store Management Project

Big Ideas MOD-1, MOD-2, MOD-3 Computational Thinking Practices 3.A, 3.B

Students create a program for a store or business that might exist in their community, similar to the management program created for the Joyful Pastries Food Truck throughout the unit. Students identify an object that the store or business would have that can be extended to create two subclasses, define instance variables to represent its attributes, and implement accessor, mutator, and **`toString()`** methods to work with the objects. Additionally, students use the **`Scanner`** class to obtain and use input for initializing objects and modifying attribute values and to write expressions to work with variables and object data. Students use peer feedback from code reviews to inform revisions and improvements to their projects. As part of the project development process, students create and manage priority lists and self-assess their work and progress in completing project requirements.

Unit 3: Data for Social Good Project

Big Ideas MOD-1, VAR-1, VAR-2

Computational Thinking Practices 3.D

Students create a program for a user to analyze data and find information based on their needs using the algorithms they learned throughout the unit. Students choose a user scenario or create their own and read data from a text file into a one-dimensional (1D) array to process and find information using loops and conditional statements. Students have the option of incorporating user input to interact with their program to request information from the data and execute algorithms they have implemented. Students use peer feedback from code reviews to inform revisions and improvements to their projects. As part of the project development process, students create and manage priority lists and self-assess their work and progress in completing project requirements.

Unit 4: Abstract Data Art Project

Big Ideas MOD-1, MOD-2, CON-1, CON-2

Computational Thinking Practices 3.C

Students use The Theater to create visuals and sound effects to portray meaning for a dataset they choose to analyze and visualize. Students work with image files, colors, shapes, text, and sounds by creating, traversing, and manipulating elements in one-dimensional (1D) arrays to create visuals and sound effects. In the process, students use selection statements, iteration, logical operators, and randomness to create interesting visuals and animations that portray the story behind their data. Students use peer feedback from code reviews to inform revisions and improvements to their projects. As part of the project development process, students create and manage priority lists and self-assess their work and progress in completing project requirements.

Unit 5: Personal Narrative Project

Big Ideas MOD-2, CON-2, VAR-2

Computational Thinking Practices 3.E

Students create a personal narrative using The Theater consisting of visuals and sound effects to communicate stories or experiences that have significant meaning to them. To create these effects, students plan their algorithms with pseudocode using data structures, expressions, conditional statements, and iterative statements to modify and write standard algorithms. Additionally, students define and use classes to represent components of their personal narrative, including their scenes and sounds, and use polymorphism to work with arrays of objects of superclass types and use subclass versions of methods. While working with the 2D arrays, students modify standard algorithms used with 1D arrays to find and manipulate elements in the 2D array. Students use peer feedback from code reviews to inform revisions and improvements to their projects. As part of the project development process, students create and manage priority lists and self-assess their work and progress in completing project requirements.

Unit 6: Natural Language Processing Project

Big Ideas VAR-2, CON-2

Computational Thinking Practices 3.D

Students use natural language processing (NLP) techniques to identify structure, patterns, and meaning in text, stories, poetry, songs, and other forms of communication to process, analyze, and/or generate new text. To extract data from literature, students read content from text files to store and manipulate the data using **`ArrayList`**s and **`String`** methods. Students plan and manage their projects using project management practices, including managing tasks using their Project Planning Board. Throughout the unit, students learn about and incorporate NLP techniques, such as keyword extraction, named entity recognition, part-of-speech tagging, sentence segmentation, and sentiment analysis and explore how these are used in real-world applications. Students self-assess their work and progress in completing project requirements. As students develop these algorithms, they incorporate standard algorithms for working with **`ArrayList`**s and **`String`** methods to find elements meeting specific criteria or to add and remove elements from a list. As part of the peer review process, students learn to write and test acceptance criteria and use the feedback they receive to inform revisions and improvements to their projects.

Unit 7: Creative Coding with The Theater

Big Ideas MOD-1, MOD-2, MOD-3

Computational Thinking Practices 4.A

Students create art and designs using images and drawings to create artwork or animations to convey a personal interest or story using The Theater. To further develop object-oriented design and programming skills, students incorporate overloaded and overridden methods, work with objects as parameters to methods and constructors to develop more efficient solutions more quickly and with a greater degree of confidence, and use the **`super`** keyword in subclass methods to call superclass methods. Students plan and manage their projects using project management practices, including managing tasks using their Project Planning Board. Students self-assess their work and progress in completing project requirements and learn to write criteria and perform acceptance testing to evaluate their user stories. As part of the peer review process, students write and test acceptance criteria and use the feedback they receive to inform revisions and improvements to their projects.

Unit 8: Creative Coding with the Console

Big Ideas CON-2

Computational Thinking Practices 3.C, 4.1

Students plan and develop a program using the console that expands on a previous project they developed, solves a new problem for a scenario, or explores a personal interest. Students apply the object-oriented design and programming skills they developed throughout the curriculum to incorporate object-oriented programming principles, data structures, and algorithms to demonstrate their knowledge and skills and affirm their software engineering identity. Students incorporate searching and sorting algorithms to find elements stored in 1D or 2D arrays or **`ArrayList`**s and self-assess their work and progress in completing project requirements. Additionally, students perform acceptance testing to evaluate their user stories. As part of the peer review process, students write and test acceptance criteria and use the feedback they receive to inform revisions and improvements to their projects.