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 A (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 advisers 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

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 students 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 course 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 course 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
- Unit 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
- **The Playground** – a package available in Java Lab to develop simple games using clickable images 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

Textbook

The curriculum is accessible online at studio.code.org/courses/csa-2022.

The following online textbooks can also be used as additional resources to support student learning:

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

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.

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

Some lessons make use of common classroom materials, such as:

- Poster paper
- Markers or colored pencils
- Sticky notes

Suggested substitutes can be found in individual lesson plans.

AP CSA Framework

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

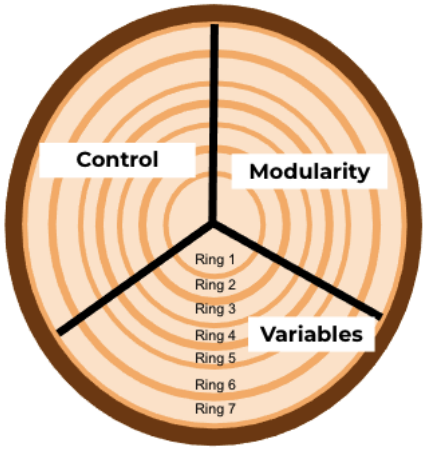
MOD	VAR	CON	IOC
Modularity	Variables	Control	Impacts of Computing

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

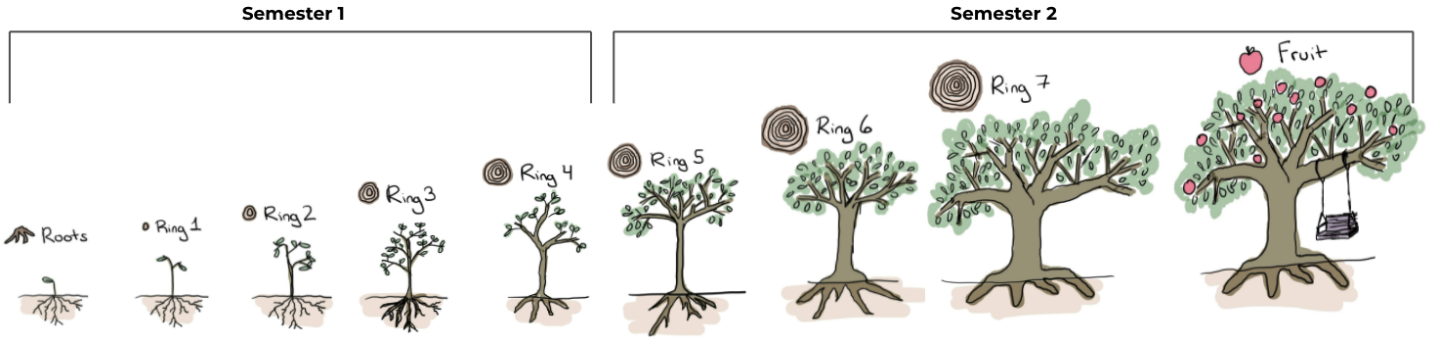
CTP1	CTP2	CTP3	CTP4	CTP5
Program Design and Algorithm Development	Code Logic	Code Implementation	Code Testing	Documentation

A Layered Approach

The Code.org CSA curriculum covers the content in a layered approach that is different than 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.



Root to Fruit: A Layered Approach



Roots: OOP ---- **Rings:** Programming Concepts ---- **Leaves:** Skills and Experience ---- **Fruit:** Software Engineering Identity

Curriculum Outline

The curriculum is divided into 9 units – 8 content units and 1 review unit. Each lesson is designed for a 45-minute class period. The last week of each unit makes up a Show What You Know week, which consists of a 3-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 45-minute lessons

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
2.5 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
5.2 Constructors
5.3 Documentation with Comments
5.4 Accessor Methods
5.5 Mutator Methods
9.1 Creating Superclasses and Subclasses
9.2 Writing Constructors for Subclasses
9.3 Overriding Methods
9.4 super Keyword
9.5 Creating References Using Inheritance Hierarchies

This unit exposes students to object-oriented programming principles as they explore The Neighborhood. Students learn fundamental Java concepts as they navigate and interact with The Neighborhood with Painter objects and create new types of Painters to expand the capabilities of their programs **(MOD-1, MOD-2, MOD-3)**. Students practice predicting the outcome of program code and use their Painter objects with conditional statements, while loops, and Boolean expressions to navigate mazes and create drawings **(CON-2)**. While students work with the Painter, they practice identifying syntax and logic errors to explain why code segments will not compile or work as intended **(Skill 4.B, Skill 5.B)**. Additionally, students learn to document program code using comments to describe the behavior of specific code segments **(Skill 5.A)**. Throughout this unit, students 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

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
2.6 String Objects: Concatenation, Literals, and More
2.7 String Methods
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
9.2 Writing Constructors for Subclasses
9.3 Overriding Methods
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 work with variables and user input as they develop and utilize classes and objects to represent desserts and customers for the Project Mercury Pastries Food Truck **(VAR-1, MOD-1, MOD-2)**. After designing a class to represent a dessert and a customer, students build on their knowledge of inheritance to design subclasses that represent specific types of desserts and customers **(MOD-3)**. While working with variables, students learn to create and evaluate expressions using assignment operators and trace code segments to determine the result of output **(CON-1, Skill 2.A, Skill 2.B)**. Throughout this unit, students continue to develop software engineering skills as they learn to use UML diagrams to represent classes and the DRY principle to refactor program code.

Unit 3: Arrays and Algorithms

15 45-minute lessons

CED Units and Topics

1.2 Variables and Data Types
1.4 Compound Assignment Operators
2.3 Calling a Void Method
3.1 Boolean Expressions
4.1 while Loops
4.2 for Loops
4.4 Nested Iteration
5.3 Documentation with Comments
5.6 Writing Methods
5.7 Static Variables and Methods
5.8 Scope and Access
6.1 Array Creation and Access
6.2 Traversing Arrays
6.3 Enhanced for Loops for Arrays
6.4 Developing Algorithms Using Arrays
9.3 Overriding Methods
9.5 Creating References Using Inheritance Hierarchies
9.6 Polymorphism

This unit introduces students to data structures to store primitive values and object references as they explore The Theater. Students use one-dimensional (1D) arrays to represent lists of data while expanding their knowledge of loops and conditionals to analyze and process data in a 1D array **(VAR-2)**. 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 **(CON-2, Skill 4.C)**. Throughout the unit, students develop and modify standard algorithms to find and manipulate elements in a 1D array while also discovering the concept of polymorphism when traversing arrays of objects **(MOD-3)**. As part of the planning process, students identify the preconditions and postconditions that an algorithm must satisfy **(Skill 5.D)**. Additionally, students learn to use the final keyword to create static variables to represent values shared with all classes and constants to represent values that cannot be changed **(MOD-2)**.

Unit 4: Conditions and Logic

15 45-minute lessons

CED Units and Topics

1.5 Casting and Ranges of Variables
2.9 Using the Math Class
3.1 Boolean Expressions
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
9.7 Object Superclass

This unit revisits the use of packages to access existing functionalities in their programs, such as The Neighborhood and The Theater, to expand their knowledge of APIs and libraries and explore The Playground. In The Playground, students learn to improve the decisions made in conditionals and loops with relational and logical operators to evaluate primitive values and object references **(CON-2)**. Students expand their knowledge of the static keyword to write static methods and explore their functionality **(MOD-1, MOD-2)**. Additionally, students use casting operators and Math class methods to evaluate expressions and perform calculations, including incorporating random in program decisions and behaviors. While working with compound Boolean expressions and logical operators, students develop an understanding of De Morgan's Laws and learn how to evaluate truth tables **(CON-1)**.

Unit 5: Two-Dimensional Arrays

15 45-minute lessons

CED Units and Topics

5.4 Accessor Methods
6.4 Developing Algorithms Using Arrays
8.1 2D Arrays
8.2 Traversing 2D Arrays
5.10 Ethical and Social Implications of Computing Systems

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 **(VAR-2)**. 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 **(CON-2, Skill 1.B, Skill 1.C)**. 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 **(IOC-1)**.

Unit 6: ArrayLists and String Methods

15 45-minute lessons

CED Units and Topics

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

This unit continues to expand on data structures to introduce students to creating lists using the ArrayList class **(VAR-2)**. 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 **(VAR-1)**. 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 **(CON-2)**. Throughout the unit, students learn to use the String class to analyze and process text obtained from a user and from file input **(VAR-1)**. Additionally, students further develop software engineering skills by writing Javadoc comments to create API documentation for their programs and notating the preconditions and postconditions of their code.

Unit 7: Method Decomposition and Recursion

15 45-minute lessons

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
5.10 Ethical and Social Implications of Computing Systems
9.4 super Keyword
10.1 Recursion

This unit allows students to practice software design and development using the skills they have learned to create a title sequence for their favorite content creator in 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 **(MOD-1, MOD-2, MOD-3)**. Throughout the unit, students practice tracing and writing recursive methods and comparing these methods to iterative solutions **(CON-2, Skill 2.C)**. 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 **(IOC-1)**.

Unit 8: Searching and Sorting

15 45-minute lessons

CED Units and Topics

4.5 Informal Code Analysis
7.4 Developing Algorithms Using ArrayLists
7.5 Searching
7.6 Sorting
7.7 Ethical Issues Around Data Collection
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 ArrayLists **(CON-2)**. 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 **(Skill 2.D)**. Throughout the unit, students apply their programming and software engineering skills to develop a game in The Playground that incorporates object-oriented design and efficiency. With the knowledge and skills acquired throughout the year, students consider the privacy and security of programs and users **(IOC-1)**.

Unit 9: AP Exam Review and Practice

15 45-minute lessons

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 (MOD, CON, CTP3)

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. (**MOD-1**, **MOD-2**, **Skill 3.A**, **Skill 3.B**). 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 (**MOD-3**). 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 (**Skill 3.A**). Students use conditional statements and while loops while incorporating the ! (not) logical operator and Boolean expressions to manage navigation and painting for their design (**CON-2**).

Unit 2: Store Management Project (MOD, VAR, CON, CTP3)

Students create a program for a store or business that might exist in their community similar to the management program created for the Project Mercury 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 (**MOD-1**, **MOD-2**, **MOD-3**, **Skill 3.B**). In the process, students use primitive and reference variables to store and manipulate primitive values, String literals, and object references and data using assignment and arithmetic operators (**VAR-1**, **CON-1**). 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 (**Skill 3.A**).

Unit 3: Personal Narrative Project (MOD, VAR, CON, CTP3)

Students create a personal narrative in The Theater using visuals and sound effects to communicate stories or experiences that have significant meaning to them. Students work with image files, colors, shapes, text, and sounds by creating, traversing, and manipulating elements in one-dimensional arrays to create visuals and sound effects (**Skill 3.D**). Students use one-dimensional (1D) arrays to store multiple image files, colors, shapes, and audio samples and use constants and static variables to represent values that cannot be changed and values to be shared between classes (**MOD-1**, **VAR-1**, **VAR-2**). To create these effects, students plan their algorithms with pseudocode using expressions, conditional statements, and iterative statements to modify and write standard algorithms (**CON-2**, **Skill 3.C**). 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 (**MOD-2**, **MOD-3**). Students use peer feedback from code reviews to inform revisions and improvements to their projects. As part of the project development progress, students create and manage priority lists and self-assess their work and progress in completing project requirements.

Unit 4: Block Party Project (MOD, CON, CTP2)

Students create a simple game in The Playground that might be played in their community similar to the games they worked with throughout the unit. In their games, students develop classes to represent sprites and game elements that can be displayed or used as clickable items (**MOD-1**, **MOD-2**). Students outline the mechanics of their game, write pseudocode for each component, and then use an iterative design approach to develop a game that responds to a player's selections and interaction. Students use selection statements, iteration, logical operators, and randomness to create the core logic of their game (**CON-1**, **CON-2**, **Skill 3.C**).

Unit 5: Sustainable Goal PSA Project (MOD, VAR, CON, CTP3)

Students create a public service announcement (PSA) in The Theater using image filters and effects to inform their community about the benefits and impacts of their chosen sustainable goal. Students define and use classes to represent types of images and their filters by creating, traversing, and manipulating image pixels in two-dimensional (2D) arrays to incorporate into their PSA using an iterative design process (**MOD-2**, **VAR-2**, **Skill 3.E**). While working with the 2D arrays, students modify standard algorithms used with 1D arrays to find and manipulate elements in the 2D array (**CON-2**). As part of the project development process, students create and manage priority lists and plan their algorithms with pseudocode.

Unit 6: Literature Festival Project (VAR, CON, CTP3)

Students use natural language processing (NLP) techniques to identify structure, patterns, and meaning in stories, poetry, songs, and other forms of literature to process, analyze, and/or generate new literature of their favorite author or artist. To extract data from literature, students read content from text files to store and manipulate the data using ArrayLists and String methods (**VAR-2**, **Skill 3.D**). 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. Students self-assess their work and progress in completing project requirements. As students develop these algorithms, they incorporate standard algorithms for working with ArrayLists and String methods to find elements meeting specific criteria or to add and remove elements from a list (**CON-2**). 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: Title Sequence Project (MOD, CTP4)

Students create a title sequence to introduce a TV show for their favorite content creator using images and drawings to create animations and effects in 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 (**MOD-1**, **MOD-2**, **MOD-3**). 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 computer-generated imagery (CGI) concepts and techniques to inform the design of their images and animations for their title sequence. 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 (**Skill 4.A**).

Unit 8: Inclusive Playground Project (CON, CTP3, CTP4)

Students create a game that contributes to creating an inclusive community environment in The Playground. 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 inclusive game design principles and techniques to make their game welcoming and enjoyable for the people in their community. Students incorporate searching and sorting algorithms in their games to find elements stored in 1D or 2D arrays or ArrayLists and self-assess their work and progress in completing project requirements and perform acceptance testing to evaluate their user stories (**CON-2**, **Skill 3.C**, **Skill 4.1**). 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.