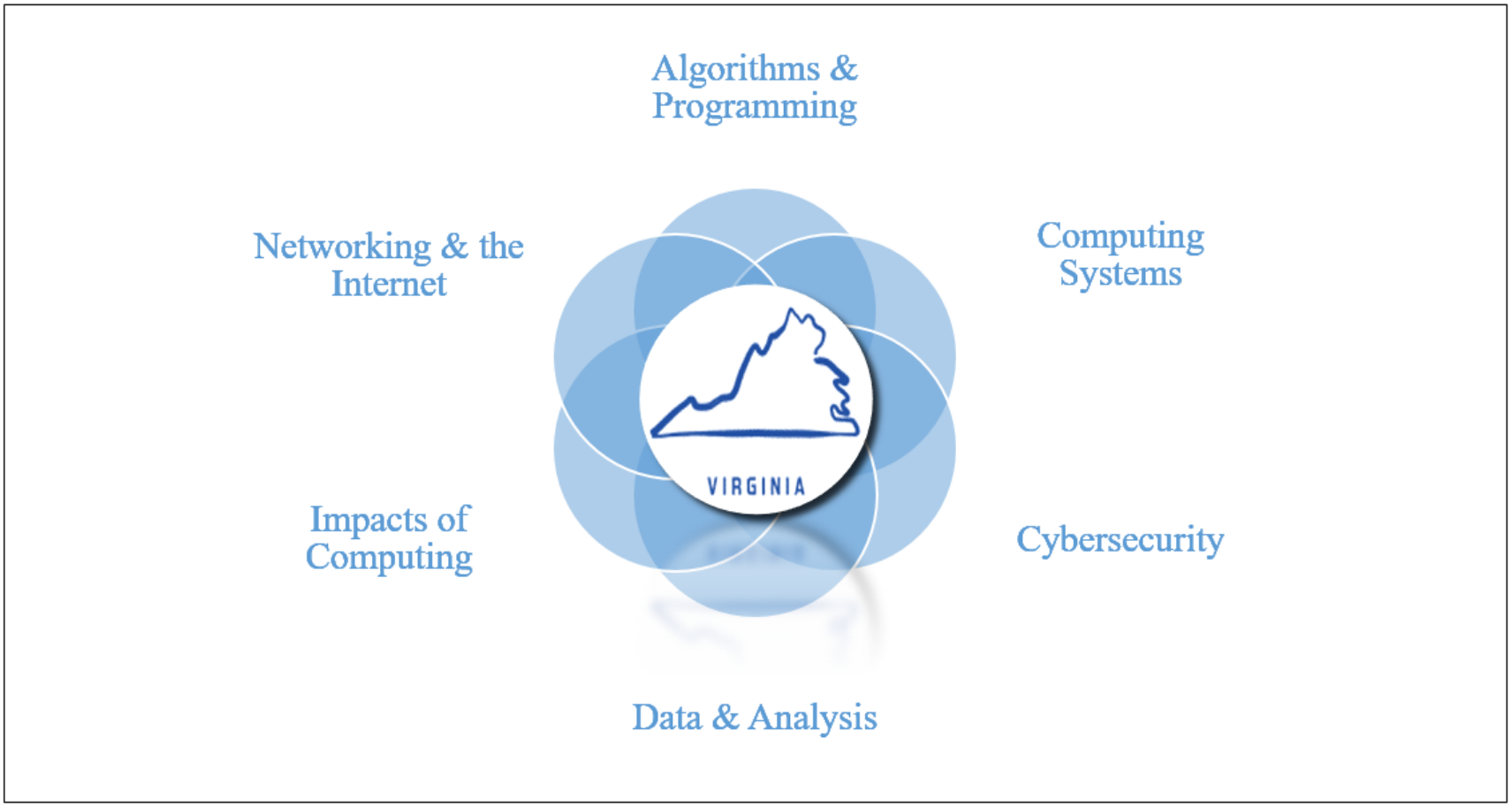
Computer Science Standards of Learning

Curriculum Framework



Board of Education
Commonwealth of Virginia

Superintendent of Public Instruction
James F. Lane, Ed.D.

Assistant Superintendent of Learning
Gena Keller

Office of Science, Technology, Engineering, and Mathematics
Tina Manglicmot, Ed.D., Director
Anne Petersen, Ph.D., Science Coordinator
Timothy Ellis, Computer Science Specialist
Joshua Bearman, Science Specialist

NOTICE
The Virginia Department of Education does not discriminate on the basis of race, sex, color, national origin, religion, age, political affiliation, veteran status, or against otherwise qualified persons with disabilities in its programs and activities.

The 2017 *Computer Science Curriculum Framework* can be found on the Virginia Department of Education's Web site.

Introduction

The *Computer Science Standards of Learning* Curriculum Framework amplifies the *Computer Science Standards of Learning for Virginia Public Schools* and defines the content knowledge, skills, and understandings that are measured by the Standards of Learning. The Computer Science Curriculum Framework provides additional guidance to school divisions and their teachers as they develop an instructional program appropriate for their students. It assists teachers as they plan their lessons by identifying essential questions and vocabulary to drive instruction and defining the essential skills students should demonstrate. This supplemental framework delineates in greater specificity the minimum content that all teachers should teach and all students should learn.

School divisions should use the *Computer Science Curriculum Framework* as a resource for developing sound curricular and instructional programs. This framework should not limit the scope of instructional programs. Additional knowledge and skills that can enrich instruction and enhance students' understanding of the content identified in the Standards of Learning should be included as part of quality learning experiences.

Each topic in the *Computer Science Standards of Learning* Curriculum Framework is developed around the Standards of Learning. The format of the Curriculum Framework facilitates teacher planning by broadening the context of the standards and identifying essential student skills that should be the focus of instruction for each standard.

Context of the Standard

The Context of the Standard provides educators an explanation of the standard, including a description and the vertical development of the concept. This context will support teachers in incorporating computer science content into discipline-specific lessons. The intention of the Computer Science standards in grades K-8 is that Computer Science principles be integrated throughout content area instruction.

Essential Skills

The Essential Skills define student performance expectations aligned to each standard. The intent of the K-8 computer science standards is that the concepts are integrated into existing disciplines and this will result in these skills being emphasized differently in each content area. The expectation is that these Essential Skills are partnered with content area performance expectations as appropriate in instruction. At the high school level, the expectations in the 2017 *Computer Science Standards of Learning Curriculum Framework* are to be used in the support of standalone computer courses; the essential skills outlined in the document are not intended to be integrated into other coursework unless a teacher chooses to use the content to support discipline practices.

Essential Questions
Each standard has identified key questions to drive classroom instruction. These questions lead teachers and students toward the big ideas of each concept and provide a more holistic viewpoint used to lead instruction relating to the context of each standard.

Essential Vocabulary
In order to effectively communicate Computer Science concepts, essential vocabulary terms are defined in grade-level appropriate terms. These definitions are found in the glossary (Appendix A).

Computer Science Programming

The Computer Science Programming standards outline the content for a one-year course with an emphasis on computer programming in a text-based language. The standards build on the concepts outlined in the Computer Science Foundations and Computer Science Principles standards.

This course continues the study of computer programming and prepares students to write programs of increasing complexity to solve problems of personal interest and professional relevance in a variety of technical fields. Additionally, this course provides the knowledge and experience to prepare students for further studies in computer science.

Teachers are encouraged to select text-based programming languages and environments, problems, challenges, and activities that are appropriate for their students to successfully meet the objectives of the standards. The majority of this course will address Algorithms and Programming. While the standards below do not include new content related to Computing Systems or Networks and the Internet, they may be used to provide context for additional exploration of these topics.

Cybersecurity

PRG.1 The student will describe and use best practices of program development that make some common flaws less likely and explain how this improves computer security.

Context of the Standard
Programming best practices exist so that programmers can create software that is efficient and safe. These best practices are built on decades of mistakes and oversights, and new best practices are being generated all the time. Some examples of best practices include: validating user input, avoiding direct command-line access, organizing program code into different files, and storing data in a different place than executable code. These best practices correct common vulnerabilities (e.g., command line injection, SQL injection, cross-site scripting) and help make code more readable.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Create software programs that meet basic requirements for security based on best practices.	Students should *investigate* these concepts: • What are some examples of times when programs were misused by users to exploit other users? • How should developers respond when it becomes clear that something they made is being misused?	Students should *apply* these terms in context: • Computer Security • Program Development • Validate • Vulnerability

Data and Analysis

PRG.2 The student will create programs that model the relationships among different elements in collections of real-world data.

Context of the Standard
One way computer programs are useful is that they automate repetitive tasks. One example of this use case is using a program to process and model (visualize) large amounts of data. Software programs and hardware devices are constantly creating data. Smartwatches for example, generate data about movement, location, sleep, heart rate, and many other kinds of user input. In order to interpret the raw information generated by software and hardware devices, programmers create data visualizations—visual representations of data. Some visualizations are traditional (e.g., bar graphs, pie charts, scatter plots), while others are more dynamic, including sounds, haptic feedback, or input which allow users to explore the data interactively. Some programmers will use data to create data-based art, which focuses less on communicating trends and more on creating evocative visual, sounds, or experiences. A visualization may even make use of data being gathered in real time, where sensors or programs update a database and the visualization program updates by pulling new data constantly. For example, a visualization might show domestic flights by pulling data from an air traffic control database. The software sends these requests using an application programming interface (API), a series of specialized commands which allow programs to access a database.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Create a program that models the relationship among different elements in a collection of data. • Create a visualization given a data set. • Use a data visualization to draw conclusions about the underlying raw data.	Students should *investigate* these concepts: • What types of data visualizations can you find on the Internet that is refreshed in real time (e.g., weather, traffic)? • What type of daily data could a school create, and how might you visualize that data?	Students should *apply* these terms in context: • API • Data Set • Data Visualization • Input • Interactive Data • Model • Raw Data

PRG.3 The student will translate numbers between machine representations and human-accessible representations.

Context of the Standard
Computers do not store data in human-readable formats. While people store data using written language, computers store data using a variety of systems that differ across use cases. At a fundamental level, these numbers are binary, consisting of zeros and ones. Sometimes, computers will store data in hexadecimal formats, or in other more verbose formats like Javascript Object Notation (JSON). Programmers are often tasked with the job of translating these machine representations into human-readable data by creating programs that take data as input and create language as output. For example, an online attendance database takes machine data and outputs a table or other visualization letting teachers and administrators know which students are in school and which ones are absent.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Identify different data formats (e.g., binary, hexadecimal, JSON) in context. • Translate given machine data into human language. • Store human input in a machine-readable format.	Students should *investigate* these concepts: • What are some reasons machines store data in a format other than language? • Why do you think there are so many different kinds of data?	Students should *apply* these terms in context: • Binary • Hexadecimal • Human-accessible Representation • Machine Representation

Algorithms and Programming

PRG.4 The student will design and implement a program working individually and in teams using a text-based language.

Context of the Standard
Programming is how people create new tools and experiences with computers. Programming languages are the creative medium of computer programming, allowing people to solve problems by generating new tools that run on computing devices. Many times, programmers work in teams by dividing the work among the team members. For example, each team member might work on one page of the application. This allows individuals to focus on making one aspect of the program as good as it can be, without worrying about the details of the rest of the application. After everyone has finished their delegated task, the team will come together and collaboratively combine their pieces into a cohesive whole.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills:	Students should *investigate* these concepts:	Students should *apply* these terms in context:

Essential Skills	Essential Questions	Essential Vocabulary
• Organize a plan to develop a program individually and as part of a team. • Create a program that solves a problem using text-based programming. • Reflect on a completed program to identify possible future opportunities for improvement.	• What are some of the benefits and drawbacks to working alone or in a team developing software? • What are some strategies for dividing labor when working on a software development team?	• Compile • Execute • Integration • Programming Language • Software Application

PRG.5 The student will explain the software life cycle and how it applies to iterative development processes.

Context of the Standard
Creating new software involves several steps, including designing, prototyping, testing, and debugging. Some programmers use the notion of a minimum viable product, or MVP, as a guide to help focus their creative efforts. An MVP is the simplest version of a tool that solves a problem. For example: if someone wants a tool to quickly move from place to place over long distances, a car might be a final solution. A car, however, is a very complex machine and may be difficult to design, prototype, and test in a reasonable amount of time. A minimum viable product for this problem might instead be a skateboard. It solves the problem, but is much easier to design, prototype, and test. After the skateboard is successful, the team might add on to it to solve the next problem (e.g., steering needs to be improved, so add a handle and make a scooter). This iterative process of designing, prototyping, testing, and debugging ensures that programmers and designers avoid wasting time and resources on a complex solution to a simple problem. Creating a minimum viable product is an excellent way of ensuring that programmers are frequently testing, debugging, and refining their software as they work toward short-term, iterative goals. After creating the initial MVP, programmers continue to build upon their work iteratively until the software meets its design specifications and is ready for release. After release, the software enters a maintenance stage, where programmers fix issues that come up as consumers use the product. Eventually, the software becomes

Context of the Standard
obsolete due to technological changes and the software enters its final stage of life, where programmers stop providing updates and support as the program is retired.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Explain the software life cycle. • Describe the process for designing and creating an original software program. • Given a problem, generate a minimum viable product.	Students should *investigate* these concepts: • What is the minimum viable product of a piece of software you use a lot? • Why might you test a product while building it rather than waiting until you finish the product? • What are some of the differences between programming alone versus programming with a team?	Students should *apply* these terms in context: • Iterative process • Minimum Viable Product • Program • Prototype

PRG.6 The student will design and implement an algorithm

a. with compound conditional execution, and analyze and evaluate complex Boolean conditions; and
b. using complex iteration, including nested loops.

Context of the Standard
An algorithm is a sequence of instructions that takes input, processes that input, and produces output. Compound conditional execution refers to a code snippet that checks input against a nested series of conditional (e.g., 'if') statements. These conditional statements evaluate Boolean statements, which return a true or false value depending on the input (e.g., $2 < 1$ returns a 'false' value) and run code based on the return value (one set of code for true, one set of code for false). Sometimes, conditional statements contain multiple Boolean statements which are evaluated against one another using logical operators (e.g., 'and', 'or'). Iteration

Context of the Standard
refers to a process that executes the same code given a sequence of inputs, like using a 'for' loop to iterate through a table or list (PRG.16). Complex iteration might involve using nested loops to iterate through a multi-dimensional table.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Create a program that makes use of compound conditional execution and complex iteration. • Predict the result of complex Boolean and/or logical expressions/conditions. • Trace the values of named variables over the course of a program that uses nested loops.	Students should *investigate* these concepts: • What is an example of a decision-making process that might be expressed using compound conditional statements (e.g., college entrance criteria, athletic team tryouts)? • Why do programmers use loops and iteration rather than writing out long lists of repetitive code?	Students should *apply* these terms in context: • Algorithm • Boolean • Conditional • Input • Iteration • Output

PRG.7 The student will implement programs that accept input from a variety of sources and produce output based on that input.

Context of the Standard
Input and output can take many forms in the digital realm. A smartphone is an example of a device that utilizes multiple inputs. These may include: location, touch, orientation, temperature, pressure, visual (through the cameras), aural (through the microphone), and many other kinds of input that create numerical data which is fed into software. The device takes these inputs and generates output to users, letting them know what processes are running on the device and prompting users to create more input data. The relationship between input and output is called a feedback loop. Users create input, and receive output which then prompts

Context of the Standard
more input in a cycle that eventually meets users goals. If the feedback loop is smooth, users might say the program is intuitive or fun to use. If the feedback loop is clumsy or choppy, users might choose to stop using the program.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Identify methods of input being processed in an example program (e.g., Instagram). • Create a program that takes user input and produces output based on that input. • Diagram the feedback loop of a software program.	Students should *investigate* these concepts: • Why do programmers want to create a feedback loop for users? • Are highly effective feedback loops good or bad for people? Give an example. • What are some of the visible and invisible inputs being processed by software you use in school?	Students should *apply* these terms in context: • Feedback loop • Input • Output

PRG.8 The student will trace the execution of iterative and recursive algorithms, illustrating output and changes in values of named variables.

Context of the Standard
Code tracing is the practice of analyzing a section of code and identifying the values of critical variables at certain points in the algorithmic process. Tracing is the first step toward debugging or otherwise decoding a computational process; it involves making hypotheses about how the code will change input values during execution, and ultimately involves predicting the output of an algorithmic process. As programmers trace code, they will identify how different variables change over time, and then compare that

Context of the Standard
hypothesis to outputs given by the program. If the expected output differs from the actual output, programmers will either update their hypothesis or change the code to reflect their intended design. Iterative algorithms are processes that are applied in the same way to a series of data (PRG.6 & 16). Recursive algorithms are processes that use themselves as part of their process. Recursion can seem complex; the classic example of a recursive algorithm is one that solves the factorial of a given input. When given a number as input, the program checks to see if the number is 1 or greater than 1. If the number is 1, the program returns the number '1' as its output. If the number is greater than one, it runs the factorial algorithm using the input number minus 1 and returns the output multiplied by the input number. For example, say the user inputs the number '2' to the factorial algorithm. The number is greater than one, so the program will run the factorial function again using 1 as input (1 less than the original input), and multiply the result by 2 (the input number). This effect will compound if larger numbers are used. factorial(2) = 2 * factorial(1) factorial(1) = 1 Therefore factorial(2) = 2*1 = 2

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Generate working recursive or iterative algorithms given pseudocode. • Create programs that makes use of iterative or recursive algorithms. • Debug iterative and recursive algorithms.	Students should *investigate* these concepts: • Why do programmers use recursion rather than just writing the whole thing out? • What is the difference between recursion and iteration?	Students should *apply* these terms in context: • Algorithm • Iteration • Input • Output • Recursion • Variable

PRG.9 The student will perform complex computations

a. on numbers, including modular division and random number generation; and
b. on strings, including substring manipulation and processing individual characters.

Context of the Standard
Programming often involves manipulating numbers and strings (data as text) to prepare it for processing or as part of an algorithmic process. Modular division involves using the modulo operator (%) to find numbers that divide evenly. The modulo returns the remainder after division of two numbers; 6 % 3 returns 0, while 6 % 4 returns 2. Given n, if n % 2 equals 0 then n is an even number. Random number generation is used to create normally distributed data sets, or to create procedural processes based on non-repeating sequences. Many random number-generating functions are pseudo-random, meaning they use a random seed that can be set or reset. Sequences of random numbers using the same seed will be identical (though still random and normally distributed) from run to run. This is to say that while pseudo-random number sets are comprised of random numbers, they are comprised of the same random numbers in the same order provided they are based on the same seed. String manipulation involves taking a string apart and using its constituent pieces as data in a program. String manipulation can be used to sanitize user input (e.g., removing all articles from a string of written text) or to find keywords in a given text. Strings can also be combined (called concatenated) with other strings or numbers to construct sentences. Programmers can also use regular expressions (RegExs) to find specific patterns of characters in strings. There are many different kinds of regular expressions, and many ways to use them which go beyond the scope of this document.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Create a program that makes use of random number generation, modular math, and/or substring manipulation.	Students should *investigate* these concepts: • How could modular math be used in encryption?	Students should *apply* these terms in context: • Modulus • Randomization

Essential Skills	Essential Questions	Essential Vocabulary
• Explain how and why an example program uses modular math, substring manipulation, and/or random number generation.	• What are some possible uses for substring manipulation? • Why might a program need to create random numbers?	• Regular Expression • String • Substring • Variable

PRG.10 The student will demonstrate an understanding of different data types by using appropriate constructs to convert between them when appropriate.

Context of the Standard
Data can be stored in many different forms in a computer program. These data types have different uses, and understanding the relationships between each form of data is very important. Some examples include referential text, strings, numerical values, and Boolean values. Referential text is any word in code that refers to some sort of data. Referential texts include variable names, function names, list names, etc. These words derive their meaning from the data they represent. Referential text is different from a string, which is text data. Strings can be stored in referential text without changing the value of the string; if a programmer types "print(name)," the program will output the value stored in the variable 'name'. If the program reads "print('name')," the program will output the word 'name' rather than the value stored in the variable 'name'. The quotation marks tell the computer that the data within the "print()" function is a string, and does not represent some other kind of data. Boolean values (i.e., true and false) are not strings (text data) or numerical data; they are their own separate type of data.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Predict the different types of data being used in a program.	Students should *investigate* these concepts: • What is the difference between strings and numerical values? Why do	Students should *apply* these terms in context: • Boolean • Function

Essential Skills	Essential Questions	Essential Vocabulary
• Validate different types of data using conditional statements and/or debugging methods. • Create a program that uses variables (Boolean, string, numerical), lists, and function definitions.	programming languages treat these data types differently? • What are some ways of keeping track of the data types for different variables in a program?	• Integers • Numerical Values • String • Variable

PRG.11 The student will analyze a large-scale computational problem, identify generalizable patterns, and implement a solution.

Context of the Standard
Computational problems are varied and may exist in any domain in or outside of computer science. Often, computational problems emerge out of real-world problems that programmers seek to solve using software. Solving computational problems involves finding out what steps the computer must execute in order, and then finding patterns in how the computer should process data. Once the programmer identifies patterns, they can abstract the computational function and allow the code to apply to multiple contexts or become more efficient as it processes large amounts of data. For example, imagine that a programmer tries to create software that will sort objects according to likeness. At first, the programmer needs to figure out a way for the computer to look at a single object for a single attribute (e.g., color) and apply a label. To prototype this early iteration of the program, the programmer might feed the program one object at a time to see if it reliably applies the appropriate label to it. Once this stage works, the programmer's goal is to apply this same code for sorting objects based on a single attribute (e.g., color) to the problem of sorting objects based on multiple attributes (e.g., color, size, and mass). The code should be generally the same for all three of these attributes; abstracting the process of analyzing objects from one use case (color) to multiple use cases (color, size, mass) solves a computational problem. The next stage might involve adding an option to analyze many objects rather than one at a time.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Analyze a computational problem to identify patterns and generate pseudocode. • Create a program that is useful across many use cases given a program that is useful in one context (i.e., abstract the implementation of a given program).	Students should *investigate* these concepts: • Why do programmers want to create abstractions of their solutions to computational problems? • How do you know when a given function or solution needs to be abstracted?	Students should *apply* these terms in context: • Abstraction • Computational Problem • Function • Iterative design • Real-world Problem

PRG.12 The student will implement an algorithm that uses existing functions and accesses existing libraries or APIs to satisfy its requirements.

Context of the Standard
Functions are like variables, except instead of storing data they store lines of code. Programmers use functions to help "chunk" large, self-contained sections of code to help make code more readable and to allow code to be used multiple times without having to re-write it. Most functions take input, and give some sort of output value. Functions almost always have parentheses after them; for example, a function that sums a list of values might look like this: "sum()." To use this function, you would put the values you would like to sum in the parentheses: "sum(2, 6, 7, 2, 1)." Given this input, the function would produce "18" as an output. Many programming languages come with predefined functions that programmers can use without having to worry about the underlying code that makes them work. For example, Math.random() returns a random number between 0 and 1 (in Java). Some programming languages have more predefined functions than others. In addition to the language-specific predefined functions, programmers may download packages that contain even more predefined functions, extending the library of functions made available by the language code base. An API is a library of functions that accesses a database stored outside of the program. Many databases have APIs that allow programmers to access data over the Internet to use in different programming applications.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Research, download, install, and use a third-party open-source library or framework. • Create a program that uses existing functions or libraries in a given programming language/framework.	Students should *investigate* these concepts: • Why do programmers use libraries rather than writing everything from scratch? • Can you think of a situation where you would not want to use an existing library or framework (e.g., how do you know a given API is secure)?	Students should *apply* these terms in context: • API • Function • Library

PRG.13 The student will write functions, both with and without parameters, and both with and without return values, that represent abstractions useful to the solution of a larger problem.

Context of the Standard
Functions are keywords that store chunks of code (PRG.12). Many times, as programmers write code that performs a specific function, it becomes necessary to abstract that portion of code such that it can be easily reused throughout a software program with different parameters. Abstraction allows programmers to hide all but the relevant data in order to reduce complexity and increase efficiency. For example, a function that produces the mean of a given set of 12 numbers could be abstracted so that it gives the mean of a given set of any number of values. In this case, the function would take a set of numbers as input, count the number of values, and divide the sum of the values by the quantity. In this way, the function becomes useful in many use cases rather than just when the given set of numbers is 12 values long. In the above example, the function returned a value equal to the mean of the given set of values. Functions do not always return values; sometimes, they will change the value of existing global or local variables already declared in the code. For example, one might write a function that sets the coordinates of a graphical object, or one that tells the computer to wait for a given length of time before continuing to execute the program.

Context of the Standard
Functions make it easier for programmers to create complex systems. Once a function has been abstracted, programmers can use it without thinking about the details of how the function works. By creating many different interacting functions, programmers can create modular systems that generate emergent complexity.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Create a program that makes use of defined functions with and without parameters/return values. • Debug a program that makes use of defined functions with and without parameters/return values.	Students should *investigate* these concepts: • Why do programmers write functions rather than writing everything out in order? • What are some strategies for making it easier to debug defined functions?	Students should *apply* these terms in context: • Abstraction • Function • Parameter • Return

PRG.14 The student will create programs demonstrating an understanding of the interactions between classes in object-oriented design, and by implementing classes with instance data and methods to satisfy a design specification.

Context of the Standard
Object-oriented programming (OOP) is a common programming paradigm that makes use of objects and methods to perform complex tasks. Objects are abstract, discrete units of data which have values called attributes. These attributes are persistent, and unique to the object. Methods are special functions written within a class to define actions that data type are capable of. Classes are templates that create objects. One example of an object in a programming context is a bank account. The bank account itself is abstract; you cannot hold or touch it. Instead, it is a data structure that has attributes: a balance, an overdraft limit, or perhaps an accruing interest rate. The class that was used to create this bank account object also created all the other bank accounts for all other bank customers. All bank accounts have the same attributes, though the values of these attributes are different for each customer. When you deposit money into the bank account, the computer program might execute a method that changes the balance attribute based on how much money you deposited. The deposit method only acts on the specified object. The program will use the same method when called on each individual object, but only affect one object at a time. Object-oriented programming is popular because it structures computer programs such that it matches human experiences of working with units of data (i.e., the objects and materials we work with in everyday life). It allows programmers to think in abstract terms and create programs that make use of similar logical patterns to those human experiences in everyday life. OOP is especially popular in videogame design, where many unique data structures must interact according to global rules (e.g., physics).

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Define object, class, and method. • Create a program that makes use of objects and methods given predefined classes.	Students should *investigate* these concepts: • Think of household pets; do fish, dogs, cats, and birds belong in the same class? Why or why not?	Students should *apply* these terms in context: • Attribute • Class • Instance • Method

Essential Skills	Essential Questions	Essential Vocabulary
• Create a class, and implement it in a program.	• What would the attributes of a "plant" class be? What are some methods that you might include?	• Object

PRG.15 The student will use code written by others by reading the documentation and incorporating it into their programs using proper citation of the reused code.

Context of the Standard
Programmers almost never create software programs without using examples from other programmers. These examples are often posted on online forums where programmers ask for help solving problems or in the form of open-source programs. It is incredibly important that programmers are able to navigate the social space of programming to expand their knowledge of different solutions to common problems. In order to facilitate knowledge sharing, programmers document their code by adding explanatory comments and documentation. Comments are small notes within code that explain portions of programs so that other programmers can learn how the software works. Documentation is often a longer text separate from the code, containing information about how to use the program and how to apply the code within the program to other contexts. Oftentimes, programmers will also include licensing information in documentation, letting other programmers know how they intend their program to be used. Some of these licenses require that programmers give credit to the creators of open-source software used in the application, while other licenses are more permissive.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Research, install, and use third-party libraries or frameworks in an original program. • Find and use examples from online resources to solve a problem in an original program. • Write documentation for an original program.	Students should *investigate* these concepts: • Why is creating detailed documentation so important? • What are some examples of bad or unhelpful documentation? What makes for good documentation?	Students should *apply* these terms in context: • Citation • Documentation • Forum • License

PRG.16 The student will read and store data in 1D and 2D collections, and design and implement algorithms to process and manipulate those collections.

Context of the Standard
Data is stored in many different forms. The simplest is a variable, storing only one value. Slightly more complex than a variable is an array or list. These terms (often synonymous, sometimes not depending on the programming language in question) refer to a variable that stores more than one value in an ordered list. For example, suppose a program is designed to store the age of each participant in a conference session. Rather than creating variables for each participant, the program might store the ages in a list. Lists are one-dimensional, storing only one value in each position. To access the values stored within a list, reference the position the value is stored within (i.e., the fifth value in the "ages" table can be found at "ages[5]" or "ages[4]," depending on if the programming languages starts counting table values at "0" or "1"). Two-dimensional data structures, like matrices, are created by storing lists at each position in a larger list. When data is stored in a list, it is very easy to access those values and manipulate the values stored therein using a programming structure called an "iterator." Iterators take an array or list as an input, and perform the same process using each value in the list one at a time. The most common iterator is the "for" loop, but there are many other techniques. In all cases, storing data in arrays rather

Context of the Standard
than in variables allows programmers to create abstractions, reusing code for each table value rather than writing code for each variable one at a time.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Create a program that makes use of 1D and/or 2D data structures. • Create a program that makes use of an iterator. • Generate a program that creates and processes 1D or 2D data structures given pseudocode examples.	Students should *investigate* these concepts: • Why do programmers store data in lists or tables instead of individual variables? • Why do programmers use iterators rather than writing everything out? • What are some challenges that arise when debugging iterators?	Students should *apply* these terms in context: • Data • Iterator • List • Array

PRG.17 The student will adapt classic algorithms for use in a particular context and analyze them for effectiveness and efficiency.

Context of the Standard
Algorithms, in addition to being a sequence of instructions in code, are also logical procedures for processing data according to rules. Because algorithms are not programming-language-specific or domain-specific, many algorithms are used across domains to solve different problems. Such algorithms include (but are not limited to): insertion sort, selection sort, bubble sort, merge sort, quick sort, binary search, A*, minimax, and many more. Each algorithm has different use cases across different domains. For example, minimax can be used to create an unbeatable tic-tac-toe game, and A* is used to find the shortest path between two given points in a field of obstacles.

Essential Skills	Essential Questions	Essential Vocabulary
Students should *demonstrate* these skills: • Generate a program designed around a commonly used algorithm given pseudocode. • Describe the effectiveness and efficiency of different methods from similar types of algorithmic solutions (e.g., sorting, searching).	Students should *investigate* these concepts: • Why are algorithms often represented in pseudocode? • How might you evaluate the effectiveness or efficiency of an algorithm?	Students should *apply* these terms in context: • Algorithm • Effectiveness • Efficiency • Pseudocode

PRG.18 The student will develop and use a series of test cases to verify that a program performs according to its design specifications, including edge cases and all branches.

Context of the Standard
When creating a computer program that performs a task for users, it is important that programmers consider all possible use cases. If a program performs well under narrow circumstances, but crashes if users misuse or misunderstand the program, it is likely the program will not perform according to its design specifications. When testing programs, designers will set up a number of scenarios that testers will work through, testing how the program performs in many different use cases. "Edge cases" refer to use cases where the program may not be designed to accommodate the case, or a use case at the minimum or maximum operating parameter. For example, a speaker distorts output audio at high volumes. Often times, the device will notify the user that the operating parameter (i.e., volume) is set too high and the user should correct the problem. Branches refer to the different paths users can take as they accomplish their goals: the series of pages or sections of the application they see as they go about looking for information or completing a task.

Context of the Standard
User experience design is the field that investigates how to create programs that are easy to use and intuitive for users. User experience designers conduct tests where users attempt to complete goals, and analyze data from those sessions to make recommendations to programmers as to how to improve the software in future iterations.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Generate a set of criteria by which to measure the success of a given program. • Create a set of criteria to measure the success of an original program. • Conduct user testing with student-created software.	Students should *investigate* these concepts: • Why do programmers and designers conduct user testing? • When in the development process should programmers start testing software with users? Why?	Students should *apply* these terms in context: • Branches • Edge case • Testing • Use case

PRG.19 The student will, through the process of code review, evaluate a program's correctness, readability, usability, and other factors.

Context of the Standard
Code review is the process of reading through a program to understand the processes it goes through as it completes tasks, and how efficiently it solves problems. It is important that programs are correct (performs according to design specifications), readable (code is logically written and easy to understand), and usable (it is easy to understand what the program does, and easy to make the program perform as expected). There are many different ways to evaluate a computer program, and the methods designers use to

Context of the Standard
evaluate programs will vary depending on the intended user profile, the complexity of the task the program should perform, and security or privacy issues embedded in the use case. Some techniques for improving code readability include creating debugging messages that prompt programmers to add required inputs, call functions with correct parameters, or otherwise correct misuse. Sometimes, improving code readability can be as simple as creating variable or function names that show the reader what the purpose of the variable or function is (a variable that stores the amount of time that has passed since the program began should be called ‘timer’ or ‘clock’).

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Trace the execution of student-created programs. • Generate criteria for readability and usability given good and bad examples. • Evaluate student-created programs based on original criteria.	Students should *investigate* these concepts: • What are some things code reviewers should look for when evaluating code? • Who should be responsible for software that violates users privacy, or that creates unjust or toxic social impact?	Students should *apply* these terms in context: • Code review • Design specifications • Readability • Usability

PRG.20 The student will use a systematic approach and debugging tools to independently debug a program.

Context of the Standard
Debugging is an essential part of the software development cycle. When programmers debug a software program, they look at the code and use a variety of techniques (e.g., adding print statements, code tracing) to validate their assumptions about how the program is operating. If they can confirm that the program is working as expected, then the programmer can move on to add new features. If the program produces and unexpected output, the programmer will correct the mistake and add new lines of code to help get more detailed debugging information until the program produces the expected output value. Debugging is, fundamentally, the process of making hypotheses and then validating those hypotheses using output from the software program. Code tracing (PRG.8) and using tracing tables are other important aspects of debugging that helps programmers focus in on potentially buggy or difficult to understand code snippets.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Debug a given software program. • Create a software program that includes mechanisms to assist in debugging. • Generate procedures for debugging software.	Students should *investigate* these concepts: • What are some ways to help avoid a situation where you don't know how to debug your software? • What are some ways to identify and validate the assumptions you make about how your program works as you work on debugging?	Students should *apply* these terms in context: • Code tracing • Debug • Software Development Cycle • Tracing Tables

Impacts of Computing

PRG.21 The student will identify some of the practical, business, and ethical impacts of open source and free software and the widespread access they provide.

Context of the Standard
Open source software is an important aspect of software development communities. Programmers create software, and then release their work with annotations and documentation so that other programmers can learn from their work and create new things based on the open source program. When software is "open source," it means that the developers have relinquished all or part of their exclusive intellectual property rights and have made their work publicly available for other programmers to use, alter, or reuse in different contexts. Software developers also contribute to open source projects as an act of community-building. Open source software is an excellent resource for people who are learning to code, and it makes it easy for people to create tools that solve problems for users or other software developers. When software developers release open-source code, they will often specify a license which explains the terms under which the open source software may be used. The MIT license is a popular example.

Essential Skills	**Essential Questions**	**Essential Vocabulary**
Students should *demonstrate* these skills: • Describe the meaning of a program being "open source." • Identify examples of open-source software for software development. • Explain the reasons for open-source software and its impact on software development.	Students should *investigate* these concepts: • Why do programmers create open-source software? • What are some of the reasons a programmer might decide to make software "closed-source"? • Is it ethical for programmers to create open-source alternatives to paid software (e.g., Inkscape and Corel)?	Students should *apply* these terms in context: • Intellectual Property Rights • License • Open-source • MIT License