SAN LEANDRO UNIFIED SCHOOL DISTRICT
COURSE PROPOSAL: Course of Study
SECTION A. COVER PAGE

1. Course Title: AP Computer Science Principles	**7. Action** ☒ New Course ☐ Course Revision ☐ Title Change Only
2. Date Submitted: 11/16/16	**8. Grade Level**(s) ☐ 6 ☐ 9 ☒ 11 ☐ 7 ☒ 10 ☒ 12 ☐ 8
3. Transcript Title/Abbreviation(s)	**9. Prerequisites** (Please list.) ☐ Required **Algebra 1** ☐ Recommended **Algebra 2**
4. Transcript Course Code/Course Number :	**10. Seeking Program Distinction?** ☐ YES (Check one below.) ☐ NO ☐ Honors ☒ AP ☐ Other
5. Subject Area Computer Science	**11. Is this a CTE course?** ☐ YES ☐ NO **If YES, complete Section C**
6. Department Career Technical Education	**12. Previously approved by UC?** ☐ YES ☐ NO ☐ NA for 6th-8th Year approved Year removed
13. UC ELIGIBILITY **Already eligible?** ☐ NA for 6th-8th ☒ **YES** ☐ **NO** **a** ☐ **b** ☐ **c** ☐ **d** ☐ **e** ☐ **f** ☐ **g** ☐ **Proposing eligibility?** ☐ NA for 6th-8th ☒ **YES** ☐ **NO** **a** ☐ **b** ☐ **c** ☐ **d** ☐ **e** ☐ **f** ☐ **g** ☐ Please mark one for approved eligibility or proposed eligibility.	**14. Unit Value/Course Credit/Length of Course** ☐ 0.5 (semester or half-year ☒ 1.0 (one year equivalent) ☐ 2.0 (two-year equivalent) ☐ Other
15. School Contact and Course Developer Name: Anthony Keithley Phone:510-618-4600 x2236 Email: akeithley@slusd.us Position: Teacher	
16. Approval Signatures: Department Chair Principal:	**Approval Date:** 11/16/16 11/16/16
	DISTRICT OFFICE USE ONLY **Date approved by Board Curriculum Committee** 11/29/16 **Date approved by Board**

SECTION B. COURSE CONTENT

17. Course Description

This course covers the AP Computer Science Principles Framework, including the Seven Big Ideas of computing: Creativity, Abstraction, Data and Information, Algorithms, Programming, the Internet, and Global Impact. Students will explore these concepts with a primary emphasis on programming, using Snap!, where students learn programming logic and major concepts without the complexity of the syntax of a specific language (Snap is based on Javascript).

Students will use use their knowledge of mathematics, English language arts, and the arts to create their own applications, animations, and games.

18. Course Goals and/or Major Student Outcomes

This course has a heavy emphasis on critical thinking and problem solving. Students will learn to use abstraction to break large problems down into smaller problems and use algorithms to create self-contained programs that are immediately usable.

Students will use computing principles to strengthen their knowledge of mathematics and English, from using angles and rotation to create various graphics and animations, to exploring the rules of the English language to create interactive chatbots that students can interact with and talk to.

Furthermore, students will exit the course with a greater understanding of how computers and computing is changing our world, from the impacts of big data and ever-increasing interconnectivity with the world around them to human-computer interaction and the social implications of computing.

19. Course Objectives (standards)

https://drive.google.com/a/slusd.us/file/d/0ByV56Bd34MGPUVdCbjY1d09RREU/view?usp=sharing

CT: Computational Thinking

CT.L2-01 Use the basic steps in algorithmic problem-solving to design solutions (e.g., problem statement and exploration, examination of sample instances, design, implementing a solution, testing, evaluation).
CT.L2-02 Describe the process of parallelization as it relates to problem solving.
CT.L2-03 Define an algorithm as a sequence of instructions that can be processed by a computer.
CT.L2-04 Evaluate ways that different algorithms may be used to solve the same problem.
CT.L2-05 Act out searching and sorting algorithms.
CT.L2-06 Describe and analyze a sequence of instructions being followed (e.g., describe a character's behavior in a video game as driven by rules and algorithms).
CT.L2-07 Represent data in a variety of ways including text, sounds, pictures, and numbers.
CT.L2-08 Use visual representations of problem states, structures, and data (e.g., graphs, charts, network diagrams, flowcharts).

CT.L2-09 Interact with content-specific models and simulations (e.g., ecosystems, epidemics, molecular dynamics) to support learning and research.
CT.L2-10 Evaluate what kinds of problems can be solved using modeling and simulation.
CT.L2-11 Analyze the degree to which a computer model accurately represents the real world.
CT.L2-12 Use abstraction to decompose a problem into sub problems.
CT.L2-13 Understand the notion of hierarchy and abstraction in computing, including high level languages, translation, instruction set, and logic circuits.
CT.L2-14 Examine connections between elements of mathematics and computer science including binary numbers, logic, sets and functions.
CT.L2-15 Provide examples of interdisciplinary applications of computational thinking.
CT.L3A-01 Use predefined functions and parameters, classes and methods to divide a complex problem into simpler parts.
CT.L3A-02 Describe a software development process used to solve software problems (e.g., design, coding, testing, verification).
CT.L3A-03 Explain how sequence, selection, iteration, and recursion are building blocks of algorithms.
CT.L3A-04 Compare techniques for analyzing massive data collections.
CT.L3A-05 Describe the relationship between binary and hexadecimal representations.
CT.L3A-06 Analyze the representation and trade-offs among various forms of digital information.
CT.L3A-07 Describe how various types of data are stored in a computer system.
CT.L3A-08 Use modeling and simulation to represent and understand natural phenomena.
CT.L3A-09 Discuss the value of abstraction to manage problem complexity.
CT.L3A-10 Describe the concept of parallel processing as a strategy to solve large problems
CT.L3A-11 Describe how computation shares features with art and music by translating human intention into an artifact.
CT.L3B-01 Classify problems as tractable, intractable, or computationally unsolvable.
CT.L3B-02 Explain the value of heuristic algorithms to approximate solutions for intractable problems.
CT.L3B-03 Critically examine classical algorithms and implement an original algorithm.
CT.L3B-04 Evaluate algorithms by their efficiency, correctness, and clarity.
CT.L3B-05 Use data analysis to enhance understanding of complex natural and human systems.
CT.L3B-06 Compare and contrast simple data structures and their uses (e.g., arrays and lists).
CT.L3B-07 Discuss the interpretation of binary sequences in a variety of forms (e.g., instructions, numbers, text, sound, image).
CT.L3B-08 Use models and simulations to help formulate, refine, and test scientific hypotheses.
CT.L3B-09 Analyze data and identify patterns through modeling and simulation.
CT.L3B-10 Decompose a problem by defining new functions and classes.
CT.L3B-11 Demonstrate concurrency by separating processes into threads and dividing data into parallel streams.

CL: Collaboration

CL.L2-01 Apply productivity/ multimedia tools and peripherals to group collaboration and support learning throughout the curriculum.
CL.L2-02 Collaboratively design, develop, publish, and present products (e.g., videos, podcasts, websites) using technology resources that demonstrate and communicate curriculum concepts.
CL.L2-03 Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities.
CL.L2-04 Exhibit dispositions necessary for collaboration: providing useful feedback, integrating feedback, understanding and accepting multiple perspectives, socialization.
CL.L3A-01 Work in a team to design and develop a software artifact.
CL.L3A-02 Use collaborative tools to communicate with project team members (e.g., discussion threads, wikis, blogs, version control, etc.).

CL.L3A-03 Describe how computing enhances traditional forms and enables new forms of experience, expression, communication, and collaboration.
CL.L3A-04 Identify how collaboration influences the design and development of software products.
CL.L3B-01 Use project collaboration tools, version control systems, and Integrated Development Environments (IDEs) while working on a collaborative software project.
CL.L3B-02 Demonstrate the software life cycle process by participating on a software project team.
CL.L3B-03 Evaluate programs written by others for readability and usability.

CPP: Computing Practice and Programming
CPP.L2-01 Select appropriate tools and technology resources to accomplish a variety of tasks and solve problems.
CPP.L2-02 Use a variety of multimedia tools and peripherals to support personal productivity and learning throughout the curriculum.
CPP.L2-03 Design, develop, publish, and present products (e.g., webpages, mobile applications, animations) using technology resources that demonstrate and communicate curriculum concepts.
CPP.L2-04 Demonstrate an understanding of algorithms and their practical application.
CPP.L2-05 Implement problem solutions using a programming language, including: looping behavior, conditional statements, logic, expressions, variables, and functions.
CPP.L2-06 Demonstrate good practices in personal information security, using passwords, encryption, and secure transactions.
CPP.L2-07 Identify interdisciplinary careers that are enhanced by computer science.
CPP.L2-08 Demonstrate dispositions amenable to open-ended problem solving and programming (e.g., comfort with complexity, persistence, brainstorming, adaptability, patience, propensity to tinker, creativity, accepting challenge).
CPP.L2-09 Collect and analyze data that is output from multiple runs of a computer program.
CPP.L3A-01 Create and organize web pages through the use of a variety of web programming design tools.
CPP.L3A-02 Use mobile devices/ emulators to design, develop, and implement mobile computing applications.
CPP.L3A-03 Use various debugging and testing methods to ensure program correctness (e.g., test cases, unit testing, white box, black box, integration testing)
CPP.L3A-04 Apply analysis, design, and implementation techniques to solve problems (e.g., use one or more software lifecycle models).
CPP.L3A-05 Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.
CPP.L3A-06 Select appropriate file formats for various types and uses of data.
CPP.L3A-07 Describe a variety of programming languages available to solve problems and develop systems.
CPP.L3A-08 Explain the program execution process.
CPP.L3A-09 Explain the principles of security by examining encryption, cryptography
CPP.L3A-10 Explore a variety of careers to which computing is central.
CPP.L3A-11 Describe techniques for locating and collecting small and large-scale data sets.
CPP.L3A-12 Describe how mathematical and statistical functions, sets, and logic are used in computation.
CPP.L3B-01 Use advanced tools to create digital artifacts (e.g., web design, animation, video, multimedia).
CPP.L3B-02 Use tools of abstraction to decompose a large-scale computational problem (e.g., procedural abstraction, object-oriented design, functional design).
CPP.L3B-03 Classify programming languages based on their level and application domain.
CPP.L3B-04 Explore principles of system design in scaling, efficiency, and security.
CPP.L3B-05 Deploy principles of security by implementing encryption and authentication strategies.
CPP.L3B-06 Anticipate future careers and the technologies that will exist.
CPP.L3B-07 Use data analysis to enhance understanding of complex natural and human systems.

CPP.L3B-08 Deploy various data collection techniques for different types of problems.

CD: Computers and Communication Devices

CD.L2-01 Recognize that computers are devices that execute programs.
CD.L2-02 Identify a variety of electronic devices that contain computational processors.
CD.L2-03 Demonstrate an understanding of the relationship between hardware and software.
CD.L2-04 Use developmentally appropriate, accurate terminology when communicating about technology.
CD.L2-05 Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use.
CD.L2-06 Describe the major components and functions of computer systems and networks.
CD.L2-07 Describe what distinguishes humans from machines, focusing on human intelligence versus machine intelligence and ways we can communicate.
CD.L2-08 Describe ways in which computers use models of intelligent behavior (e.g., robot motion, speech and language understanding, and computer vision).
CD.L3A-01 Describe the unique features of computers embedded in mobile devices and vehicles (e.g., cell phones, automobiles, airplanes).
CD.L3A-02 Develop criteria for purchasing or upgrading computer system hardware.
CD.L3A-03 Describe the principal components of computer organization (e.g., input, output, processing, and storage).
CD.L3A-04 Compare various forms of input and output.
CD.L3A-05 Explain the multiple levels of hardware and software that support program execution (e.g., compilers, interpreters, operating systems, networks).
CD.L3A-06 Apply strategies for identifying and solving routine hardware and software problems that occur in everyday life.
CD.L3A-07 Compare and contrast client-server and peer-to-peer network strategies.
CD.L3A-08 Explain the basic components of computer networks (e.g., servers, file protection, routing, spoolers and queues, shared resources, and fault-tolerance).
CD.L3A-09 Describe how the Internet facilitates global communication.
CD.L3A-10 Describe the major applications of artificial intelligence and robotics.
CD.L3B-01 Discuss the impact of modifications on the functionality of application programs.
CD.L3B-02 Identify and describe hardware (e.g., physical layers, logic gates, chips, components).
CD.L3B-03 Identify and select the most appropriate file format based on trade-offs (e.g., accuracy, speed, ease of manipulation).
CD.L3B-04 Describe the issues that impact network functionality (e.g., latency, bandwidth, firewalls, server capability).
CD.L3B-05 Explain the notion of intelligent behavior through computer modeling and robotics.

CI: Community, Global, and Ethical Impacts

CI.L2-01 Exhibit legal and ethical behaviors when using information and technology and discuss the consequences of misuse.
CI.L2-02 Demonstrate knowledge of changes in information technologies over time and the effects those changes have on education, the workplace, and society.
CI.L2-03 Analyze the positive and negative impacts of computing on human culture.
CI.L2-04 Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems.
CI.L2-05 Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing).
CI.L2-06 Discuss how the unequal distribution of computing resources in a global economy raises issues of equity, access, and power.
CI.L3A-01 Compare appropriate and inappropriate social networking behaviors.
CI.L3A-02 Discuss the impact of computing technology on business and commerce (e.g., automated tracking of goods, automated financial transactions, e-commerce, cloud computing).
CI.L3A-03 Describe the role that adaptive technology can play in the lives of people with special needs.
CI.L3A-04 Compare the positive and negative impacts of technology on culture (e.g., social

networking, delivery of news and other public media, and intercultural communication).
CI.L3A-05 Describe strategies for determining the reliability of information found on the Internet.
CI.L3A-06 Differentiate between information access and information distribution rights.
CI.L3A-07 Describe how different kinds of software licenses can be used to share and protect intellectual property.
CI.L3A-08 Discuss the social and economic implications associated with hacking and software piracy.
CI.L3A-09 Describe different ways in which software is created and shared and their benefits and drawbacks (commercial software, public domain software, open source development).
CI.L3A-10 Describe security and privacy issues that relate to computer networks.
CI.L3A-11 Explain the impact of the digital divide on access to critical information.
CI.L3B-01 Demonstrate ethical use of modern communication media and devices.
CI.L3B-02 Analyze the beneficial and harmful effects of computing innovations.
CI.L3B-03 Summarize how financial markets, transactions, and predictions have been transformed by automation.
CI.L3B-04 Summarize how computation has revolutionized the way people build real and virtual organizations and infrastructures.
CI.L3B-05 Identify laws and regulations that impact the development and use of software.
CI.L3B-06 Analyze the impact of government regulation on privacy and security.
CI.L3B-07 Differentiate among open source, freeware, and proprietary software licenses and their applicability to different types of software.
CI.L3B-08 Relate issues of equity, access, and power to the distribution of computing resources in a global society.

20. Course Outline

Unit 1 - Intro to Snap! Programming

Students dive right into Snap! programming by designing games and visual projects using loops, randomness, and control blocks. They get to know the Snap! interface, coordinate system, sounds, mathematics, variables, and how to build their own blocks. Students are encouraged to experiment with Snap!, playing with inputs, trying out blocks before they have been formally introduced, and modifying and extending the assigned tasks.

Unit 2 - Conditionals, Abstraction, and Debugging

Starting with the challenging task of teaching the computer to generate the plurals of nouns (e.g., butterfly → butterflies, moth → moths, bush → bushes), students begin thinking about the structure of programs. They learn to test for special cases, and how to use, sequence, and optimize the conditionals that control a program based on those cases. They also begin to think about what makes correctly working code into "good" code—is it the brevity, the clarity, or some combination? They learn to think about debugging by deliberately looking for ways to make a program fail and then finding ways to avert failures.

Unit 3 - Lists

This unit introduces lists, a powerful data type for storing multiple items of any type, including numbers, strings, other lists, or even blocks (function/procedure as data). Just as functions can take numbers and strings as inputs, they can also take lists as input or produce lists as output. A list is an ordered sequence of items. Students are

introduced to several powerful higher-order list-processing functions that take functions, along with other data, as inputs. One virtue of the higher order list functions is that they generate new lists to report rather than mutate existing lists. This is in contrast with the imperative, looping, mutation-based programming that is more common, but more error-prone, in dealing with sequential data. Near the end of the course, students build these higher order functions for themselves.

Unit 4 - The Internet and Global Impact

Unit 4 addresses the structure of the Internet, the various protocols on which it runs, and the implications of this technology to society. Students learn to recognize HTML and learn that it is another computer language, and they learn to "scrape" an HTML page for data. They build on their work in Unit 3 to analyze lists containing data using Snap! procedures and strategies they have already learned.

Unit 5 - Algorithms and Data

This unit focuses on several types of analysis: analysis of problems to generate algorithms for their solution; analysis of the algorithms (especially of the time it takes to execute them) in order to optimize them; analysis of phenomena by generating models and simulations that give insight and help one generate and test hypotheses; and analysis of data, especially including visualization. Students have been generating algorithms to solve problems from the start of this course, but have not yet focused on analyzing them for efficiency. For small enough computational problems, such analysis isn't needed. But modeling complex phenomena and handling large data sets requires understanding that there are sometimes alternative algorithms that reduce the impact of the size of a model on the time it takes to execute. In-depth coverage of this broad domain (computational complexity, data analysis, modeling and simulation) is beyond the scope of an introductory course, but Unit 5's projects in each of these areas will give students a good firstapproximation understanding of these issues.

Note: At the end of Unit 5, all of the CSP curriculum framework has been addressed. Units 6 and 7 contain additional material that's important, but part of Unit 6 and all of Unit 7 will come after the AP exam in May.

Unit 6 - Trees and Other Fractals

Recursion and functional programming are two programming techniques that go beyond the Framework requirements, but are at the heart of what makes this course unique. Unit 6 is about recursive commands, mainly fractals. It starts with a teacher demonstration of the Vee project, in which a short program generates V shapes with randomly chosen decorations at the ends. This same program suddenly generates arbitrarily complex results if the Vee procedure itself is added to the list of possible decorations. This is a teacher demonstration rather than an independent lab activity because the collective gasp of the class is itself a valuable learning experience. After the demonstration, students develop their own fractal tree program by building up from small cases (a one-level tree is just a trunk; a two-level tree is a trunk with a vee above it, etc.) so that they are not confronted at first with the seeming "cheating" of a procedure calling itself. Only after they've written several almost identical procedures does the lab suggest combining them into one.

The students then discover why recursive procedures need a base case to terminate the recursion. Once the tree fractal is thoroughly understood, students go on to construct other fractals (Koch snowflake, Sierpinski gasket, etc.). The elegance of the programs themselves helps students see computer programs, and not just the effects they produce, as things of beauty. A key moment in developing that sense is when students understanding how a short recursive procedure can generate a deeply complex computational process. This unit also contains a Social Implications lab exploring the effects of computers on jobs (including both the displacement of old categories of work by new ones and the on-the-job experience of workers whose output is measured by computers) and on warfare (with an emphasis on drones and why they make a qualitative difference in the political cost of war).

Unit 7 - Recursive and Higher Order Functions

Unit 7 is about recursive functions, combining the ideas of recursion, from Unit 6, and functional programming, introduced in Unit 3 with the higher order functions on lists. One highlight of the course is the implementation by students of three key list operations: map, keep, and combine. After a brief introduction to the form of a recursive function (in which the recursive call is an input to a combining function, as opposed to a separate instruction as in recursive commands), we work through the example of Pascal's triangle. As part of that lab, we see how the naïve implementation takes exponential time, but techniques such as memoization can be used to create an efficient program that still maintains the essentially recursive definition of Pascal's triangle. Other examples in the unit include conversion of numbers to and from binary, which is then generalized to arbitrary bases (up to 36, using all the letters as digits); finding the subsets of a set (a simple example of a computation that's unavoidably exponential in time, because the desired output is exponentially large); and sorting lists (using mergesort to exemplify O(n log n) algorithms, because the algorithm is simpler than Quicksort and is guaranteed n log n rather than just probabalistically n log n). Finally, students build simple examples of recursive procedures that apply a function to every item of a list (square all the numbers, take the first letter of all the words, and so on), then generalize that pattern to write the map function, and similarly for keep and combine. This last programming lab is one highlight of the course, because it combines several central ideas: abstraction, functional programming, and recursion.

21. Instructional Materials:
Board approved required text

Abelson, H., Ledeen, K., & Lewis, H. R. (2008). *Blown to bits: Your life, liberty, and happiness after the digital explosion*. Upper Saddle River, NJ: Addison-Wesley.

Available to purchase in hardcover or free online download.
http://www.bitsbook.com/

Supplementary materials

UC Berkeley's Beauty and Joy of Computing curriculum or code.org Computer Science Principles

http://bjc.edc.org/bjc-r/course/bjc4nyc.html

http://code.org/csp

22. Instructional Methods and/or Strategies (Key Assignments)

Students explore concepts through hands-on interaction in collaboration with their peers, completing labs almost daily and through project based learning.

Two kinds of labs:

- Programming Labs, in which students practice pair programming in Snap, are included in each of the units.
- Social Implications Labs, in which students read, discuss, and write about issues of computing in society using the Blown to Bits book and various online articles. Social Implications Labs are heaviest in the first four units, to precede the AP Explore task.

23. Assessment Methods and/or Tools

Interunit Quizzes - Assessing for program interpretation, debugging, and loop tracing.

AP Explore and Create Tasks - assessed by the College Board

End of unit assessments:

- Unit 1 - Students create an interactive eCard
- Unit 2 - In preparation for the AP Create Task, students describe the purpose of each of their lab programs, how to use them, and how they developed their programs, and they also write a description of the abstractions in each of their programs.
- Unit 3 - Students complete and refine their work on the Tic Tac Toe project, describe the purpose of the program (in 100 words or less), and write a description of the *abstractions* they used (in preparation for the AP Create Task)
- Unit 4 - Students choose one computing innovation, do some additional research, and independently write a one page paper. In preparation for the AP Explore Task, students will produce a visual artifact and a reference list.
- Unit 5 - Students revisit two projects: List Searching from Unit 5, Lab 1 and Tic Tac Toe from Unit 3, Lab 4, and write a descriptions of the *algorithms* they used (in preparation for the AP Create Task).
- Unit 6 - Students create various fractals (Koch snowflake, Sierpinski gasket, etc.)

- Unit 7 - students build simple examples of recursive procedures that apply a function to every item of a list (square all the numbers, take the first letter of all the words, and so on), then *generalize that pattern* to write the map function, and similarly for keep and combine. This last programming lab combines several central ideas: abstraction, functional programming, and recursion.

24. Grading Policy

Grading will follow the San Leandro Unified School District's policy. It is expected that assignments will be turned in on time. Late assignments will be accepted if student has communicated to teacher the reasons and actions taken to complete the work in a timely manner.

SECTION C

25. Context for Course

About This Course

AP Computer Science Principles introduces students to the central ideas of computer science, inviting students to develop the computational thinking vital for success across multiple disciplines. The course is unique in its focus on fostering students to be creative and encouraging students to apply creative processes when developing computational artifacts. Students design and implement innovative solutions using an iterative process similar to what artists, writers, computer scientists, and engineers use to bring ideas to life.

To appeal to a broader audience, including those often underrepresented in computing, this course highlights the relevance of computer science by emphasizing the vital impact advances in computing have on people and society. By focusing the course beyond the study of machines and systems, students also have the opportunity to investigate the innovations in other fields that computing has made possible and examine the ethical implications of new computing technologies. In partnership with the National Science Foundation, the AP Program collaborated with secondary and postsecondary educators and members of computer science educational professional organizations to develop the AP Computer Science

Principles curriculum framework.
This new AP Computer Science Principles course is complementary to AP Computer Science A. Students can take these courses in any order or at the same time, as schedules permit. Both courses include rigorous computer science content and skills that can be built on to complete further science, technology, engineering, and mathematics (STEM) and computing studies. It is important to note that the AP Computer Science Principles course does not have a designated programming language. Teachers have the flexibility to choose a programming language(s) that is most appropriate for their students to use in the classroom.

Course Overview
The AP Computer Science Principles course is designed to be equivalent to a first-semester introductory college computing course. In this course, students will develop computational thinking skills vital for success across all disciplines, such as using computational tools to analyze and study data and working with large

data sets to analyze, visualize, and draw conclusions from trends. The course is unique in its focus on fostering student creativity. Students are encouraged to apply creative processes when developing computational artifacts and to think creatively while using computer software and other technology to explore questions that interest them. They will also develop effective communication and collaboration skills, working individually and collaboratively to solve problems, and discussing and writing about the importance of these problems and the impacts to their community, society, and the world.