290-3-3-.09 Computer Science Education (Grades 6-12).

(1) Rationale. Standards for teacher candidates in computer science education are informed by and aligned with the Alabama Course of Study for Digital Literacy and Computer Science (2018) and standards from the Computer Science Teachers Association (CSTA, 2017), and other reputable sources.

(2) Program Curriculum. In addition to meeting Rules 290-3-3-.02(6)(a)1.-4., 290-3-3-.02(6)(e)1. and 2.(i) and (iv), 290-3-3-.03, 290-3-3-.04, and 290-3-3-.30, the teaching field shall require an academic major of at least 32 semester hours of credit with at least 19 semester hours of upper-division credit. Additional information is provided in the definition for academic major in Rule 290-3-3-.01(2).

(a) Knowledge. Prior to program completion, prospective teachers of computer science shall demonstrate knowledge sufficient to teach content related to:

1. Impacts of Computing.

(i) Impact of, obstacles to, and effects of computing.

(ii) Issues regarding intellectual property, ethics (e.g., concerns related to artificial intelligence and machine learning capabilities that may affect society), privacy, and security in computing.

2. Algorithms and Computational Thinking.

(i) Abstraction; pattern recognition in data samples and computational processes; problem decomposition; and number base conversion.

(ii) Algorithm analysis, searching and sorting algorithms, recursive algorithms, randomization, and algorithm expression (e.g., pseudocode and flowcharts).

3. Programming.

(i) Programming control structures, standard operators (e.g., arithmetic, relational and logical), variables, correctness, extensibility, modifiability, and reusability.

(ii) Procedures, function, and methods; event-driven programs; usability; data structures (e.g., stacks, queues, lists); debugging; documenting and reviewing code; libraries and application programming interfaces (APIs), integrated development environments (IDEs); and programming language paradigms, including object-oriented concepts.

4. Data.

(i) Digitalization of information; data encryption and decryption; data compression, error detection and correction; and computational tools.

(ii) Simulation, modeling, and manipulation of data.

5. Computing Systems and Networks.

(i) Hardware and software (for designing systems, identifying and fixing problems, and troubleshooting issues), software life cycle, operating systems, computing systems, virtual machines, communication between devices, and cloud computing.

(ii) Networks, including protocols, encryption, and security issues and the Web.

(b) Abilities. Prior to program completion, prospective teachers of computer science shall demonstrate ability to teach students to:

1. Computing Systems.

(i) Recommend improvements to the design of computing devices, based on an analysis of how users interact with software and hardware devices.

(ii) Design projects that combine hardware and software components to collect, process, and output data.

(iii) Systematically identify and fix problems with computing devices and their components.

(iv) Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects.

(v) Compare levels of abstraction and interactions between application software, system software, and hardware layers.

(vi) Develop guidelines that convey systematic trouble-shooting strategies that others can use to identify and fix errors.

(vii) Categorize the roles of operating system software.

(viii) Illustrate ways computing systems implement logic, input, and output through hardware components.

2. Networks and the Internet.

(i) Model the role of protocols in transmitting data across networks and the Internet. (6-8)

(ii) Explain how physical and digital security measures protect electronic information, including public key cryptography.

(iii) Apply multiple methods of encryption to model the secure transmission of information.

(iv) Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.

(v) Give examples to illustrate how sensitive data can be affected by malware and other attacks.

(vi) Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.

(vii) Compare various security measures, considering tradeoffs between the usability and security of a computing system.

(viii) Explain tradeoffs when selecting and implementing cybersecurity recommendations.

(ix) Describe the issues that impact network functionality (e.g., bandwidth, load, delay, topology).

(x) Compare ways software developers protect devices and information from unauthorized access, considering different classifications of intrusion prevention systems and how each identifies malicious activity, logs information about the activity, reports it, and attempts to block or stop it.

3. Data and Analysis.

(i). Represent data using multiple encoding schemes.

(ii) Collect data using computational tools and transform the data to make the data more useful and reliable.

(iii) Refine computational models based on the data they have generated.

(iv) Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.

(v) Evaluate the tradeoffs in how data elements are organized and where data are stored.

(vi) Create interactive data visualizations using software tools to help others better understand real-world phenomena.

(vii) Create computational models that represent the relationships among different elements of data collected from a phenomena or process.

(viii) Use data analysis tools and techniques to identify patterns in data representing complex systems.

(ix) Select data collection tools and techniques to generate data sets that support a claim or communicate information.

(x) Evaluate the ability of models and simulations to test and support the refinement of hypotheses.

4. Algorithms and Programming.

(i) Use flowcharts and/or pseudocode to address complex problems as algorithms.

(ii) Create clearly named variables that represent different data types and perform operations (e.g., arithmetic, relational, and logical operations) on their values.

(iii) Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.

(iv) Decompose problems and subproblems into parts (functions) to facilitate the design, implementation, and review of programs.

(v) Create procedures with parameters to organize code and make it easier to reuse.

(vi) Seek and incorporate feedback from team members and users to refine a solution that meets user needs.

(vii) Incorporate existing code, media, and libraries into original programs, and give attribution.

(viii) Systematically test and refine programs using a range of test cases.

(ix) Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.

(x) Document programs in order to make them easier to follow, test, and debug.

(xi) Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.

(xii) Use lists or arrays to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.

(xiii) Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.

(xiv) Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.

(xv) Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and objects.

(xvi) Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.

(xvii) Systematically design and develop programs for broad audiences by incorporating feedback from users.

(xviii) Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries.

(xix) Evaluate and refine computational artifacts to make them more usable and accessible.

(xx) Design and develop computational artifacts working in team roles using collaborative tools and pair programming techniques.

(xxi) Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

(xxii) Demonstrate ways a given algorithm applies to problems across disciplines.

(xxiii) Describe how artificial intelligence drives many software and physical systems.

(xxiv) Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.

(xxv) Use and adapt classic algorithms (e.g., shortest path, sorting, and searching) to solve computational problems.

(xxvi) Evaluate algorithms in terms of their efficiency, correctness, and clarity.

(xxvii) Compare and contrast fundamental data structures and their uses.

(xxviii) Illustrate the flow of execution of a recursive algorithm.

(xxix) Construct solutions to problems using student-created components, such as procedures, nodules and objects.

(xxx) Analyze a large-scale computational problem and identify generalizable patterns that can be applied to a solution.

(xxxi) Demonstrate code reuse by creating programming solutions using libraries and APIs.

(xxxii) Plan and develop programs for broad audiences using a software life-cycle process.

(xxxiii) Explain security issues that might lead to compromised computer programs.

(xxxiv) Develop programs for multiple computing platforms.

(xxxv) Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.

(xxxvi) Develop and use a series of test cases to verify that a program performs according to its design specifications.

(xxxvii) Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).

(xxxviii) Evaluate key qualities of a program through a process such as a code review.

(xxxix). Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems.

5. Impacts of Computing.

(i) Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options.

(ii) Discuss issues of bias; accessibility for all users, including those with special needs; and usability in the design of existing technologies.

(iii) Collaborate with contributors through strategies such as crowdsourcing or surveys when creating a computational artifact.

(iv) Describe tradeoffs between allowing information to be public and keeping information private and secure, recognizing that nothing posted online is private.

(v) Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.

(vi) Test and refine computational artifacts to reduce bias and equity deficits.

(vii) Demonstrate ways a given algorithm applies to problems across disciplines.

(viii) Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields.

(ix) Explain the beneficial and harmful effects that intellectual property laws can have on innovation.

(x) Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.

(xi) Evaluate the social and economic implications of privacy in the context of safety, law, and ethics.

(xii) Evaluate computational artifacts to maximize their beneficial effects and minimize harmful effects on society.

(xiii) Evaluate the impact of equity, access, and influence on the distribution of computing resources in the global society.

(xiv) Predict how computational innovations that have revolutionized aspects of our culture might evolve.

(xv) Debate laws and regulations that impact the development and use of software.

(xvi) Consider the impact of professional societies (e.g., Association for Computing Machinery, Institute of Electrical and Electronics Engineers, Association of Information Technology Professionals) on decisions that affect society.

Author: Dr. Eric G. Mackey
Statutory Authority: Ala. Code §§16-3-16, 16-23-14 (1975).
History: New _____.