Journal of Computer Science Integration

Volume 2 | Issue 1 Article 2

2019

All Advanced Placement (AP) Computer Science is Not Created Equal: A Comparison of AP Computer Science A and Computer Science Principles

Douglas D. Havard
Chapman University

Keith E. Howard
Chapman University

Recommended Citation

Havard, D. D., & Howard, K. E. (2019). All Advanced Placement (AP) Computer Science is Not Created Equal: A Comparison of AP Computer Science A and Computer Science Principles. *Journal of Computer Science Integration, 2* (1), 16-34. https://doi.org/10.26716/jcsi.2019.02.1.2

JCSI
Journal of Computer Science Integration

All Advanced Placement (AP) Computer Science is Not Created Equal: A Comparison of AP Computer Science A and Computer Science Principles

Abstract
This article compares the two most prominent courses of Advanced Placement (AP) computer science study offered throughout 9-12 grades in the U.S. The structure, guidelines, components, and exam formats of the traditional AP Computer Science A course and the relatively newer AP Computer Science Principles course were compared to examine differences in content and emphases. A depth-of-learning analysis was conducted employing Bloom's Revised Taxonomy to examine potential differences in rigor and challenge represented by the two options, particularly as it relates to acquiring computer programming proficiency. Analyses suggest structural differences in both course content and end-of-course exam components likely result in less depth and rigor in the new Computer Science Principles course as compared to the Computer Science A course. A lower minimum standard for learning programming skills in the Computer Science Principles course was observed, making it a less viable option for students looking to acquire skills transferable to future computer science study or employment. The potential implications for students choosing the new course over the traditional offering, as well as for schools opting for the new course as its sole or primary offering are discussed.

Keywords
computational thinking, advanced placement programs, computer science education, secondary education

DOI
10.26716/jcsi.2019.02.1.2

Corresponding Author
Douglas D. Havard
Chapman University
Attallah College of Educational Studies
One University Drive
Orange, CA 92866

Cover Page Footnote
Footnote 1: The Turtle Geometry project began as a physical computing device (a robot) before transitioning to a virtual (on-screen) turtle. LOGO programming was developed and used by children to manipulate the motion of a two-dimensional turtle to draw figures in a way different from Euclid (logical style) and Descartes (algebraic style). The method is described as a "computational style" – the basis of a new way of thinking and learning (Papert, 1980, p. 55).

All Advanced Placement (AP) Computer Science is Not Created Equal: A Comparison of AP Computer Science A and Computer Science Principles

In December 2014 the College Board made a seminal announcement, declaring their intention to launch a new Advanced Placement computer science course developed in collaboration with the National Science Foundation (NSF) and designed to be "rigorous, engaging and accessible for all students" (National Science Foundation, 2014). The official launch of AP Computer Science Principles prior to the 2016 academic year marked, for the first time since 2003, a decision by the College Board to either revise or modify their model of computer science preparation for higher education. The reasons for the revision included recent paradigmatic shifts in the methods for, and approaches to, teaching computer science (Cuny, 2015). Computer science as a discipline has a long history of national importance (i.e., as a grounded field for emergent ideas and technologies) and potential for engaging career opportunities. The field, however, has been marked recently by a growing discontinuity in connecting a large population of students with the future careers that are believed to materialize from learning both the foundational and creative aspects of computer science. According to the Bureau of Labor Statistics (2018), computer and information technology occupations are expected to grow by 13% from 2016-2026, 7% faster than the average growth rate of all other occupations.

Careers such as computer and information research science, network architecture, information security analysis, and software development require skills related to both applied programming fundamentals and creative design practices. In step with the Bureau of Labor Statistics, the National Economic Council, Council of Economic Advisors, and the Office of Science and Technology Policy (National Economic Council and Office of Science and Technology Policy, 2015) have suggested that high-quality STEM education and access to STEM programs are the "building blocks of the American innovation ecosystem." Providing access to computer science curriculum for traditionally underrepresented students engages a larger and more sustainable workforce who might not have otherwise had the opportunity for access to these careers. Although the participation rate of AP Computer Science course exams had steadily increased since 2003, including a rate of increase of 22.1% per year on average between 2009-2016 (Howard & Havard, 2019), an ongoing participation gap by race and gender became a concerning trend.

Following the introduction of AP Computer Science Principles in 2016, access to computer science appeared to improve considerably, addressing the intended design goal of accessibility for all students. Comparing the two-year periods before the launch of the new course (2014/15 – 2015/16) and after the launch (2016/17 – 2017/18), there was a 124% increase in the total numbers of students participating in AP computer science course exams. Over that same period, participation increased for females by 150.2%, Hispanics by 171.9%, Blacks by 109.3%, Whites by 99.7%, and Asians by 94.6% (College Board, 2018a, 2018b). Howard and Havard (2019) illustrate that females, Hispanics and Blacks participated in the new Computer

Science Principles exam in far greater numbers than they did in the traditional Computer Science A exam, whereas more White and Asian students opted for the traditional exam over the new offering. The comparison between AP Computer Science A and Computer Science Principles exams reveals not only differential levels of diversity in participation, but also an increase in passing scores (3 or above) amongst traditionally underrepresented participants. To fully understand the scope and depth to which these results represent a move forward in the computer science educational landscape at the secondary level, it is worth taking pause before labeling Computer Science Principles as a sweeping success and assessing the second design goal expressed by the NSF. With both courses identified as the equivalent to an introductory computer science course at the post-secondary level, there is value in examining the following question through a historical and structural lens: What is the extent to which both courses compare on a spectrum of "rigor"?

The Influence of "Computational Thinking"

The curriculum framework for the new Computer Science Principles course was built around "the concepts and computational thinking practices central to the discipline of computer science…" (College Board, 2017b, p. 6). This paradigmatic approach to computer science education – the practices of computational thinking - has been around for over 50 years, but given its heavy influence on current approaches to computer science instruction in K-12, a brief discussion of its origins will provide some historical context to its recent application. The disciplinary practices and interdisciplinary ways of thinking within the field of computer science first began to enter mainstream academic discourses in the late 1950s. Attributable to spawning a cognitive revolution in the following decade (Miller, 2003, pp. 142-143), computing pioneers such as Alan Perlis sold the wider academic community on the idea that computing could be applied uniquely as a tool in solving many different types of problems from multiple fields. Central to this perspective was viewing computing as a methodology rather than a physical tool (i.e., a practice or approach for performing many different tasks rather than a tool to accomplish one specific task). Perlis used the term *algorithmizing* to explain a larger "theory of computation" by which a problem is generalized into an ordered set of steps (a procedure) for finding its solution (Tedre & Denning, 2016, p. 121). As it began to evolve, this way of thinking was discussed and debated on its merit as a "general-purpose mental tool" and its potential ability to develop higher-order knowledge transfer skills within students (Minsky, 1974). It wasn't until Seymour Papert (1980) conducted a series of seminal studies examining the effects of computers and computer programming on the problem-solving practices of K-12 students that breakthroughs in computing and learning began to evolve into classroom instructional practices.

Papert (1980) bridged theoretical perspectives, educational research, cognitive science, and computer science. In so doing, Papert tapped into more than just a cursory understanding of how students interact with technology through the delivery of information and instruction as a tool. By synthesizing problem solving in mathematics, he sought to understand how students learn through computers. The idea that these students had much to gain, through *procedural*

DOI: 10.26716/jcsi.2019.02.1.2

thinking (i.e., logically in sequences) and by applying heuristic approaches thoughtfully to programming the computer, energized this promising subfield of educational research due to its far-reaching implications. Of the major ideas resulting from this line of research (Papert, 1980, 1996; Papert & Harel, 1991) and observations of student-computer interactions through the "Turtle Geometry" project[1], a reified set of practices emerged which have played a significant part in the current description of *computational thinking*.

More recently, the concept of computational thinking reemerged through the highly influential work of Jeanette Wing (2006), spurring a renewed attention to the potential benefits of computer science concepts across other disciplines. In her position paper, Wing explored the current state of the field of computer science and considered what the field could become, providing a retrospective on "what it is" versus "what it could achieve." Wing firmly planted a claim for a set of global practices used by computer scientists to solve problems fundamental to all other subject areas. Similar to the application of the Turtle Geometry project by Papert (1980) to cognition and learning through new perspectives of drawing, Wing envisioned computational thinking as an approach to designing problem solutions which transcended geometry and movement. In the process, Wing redirected the scope of the field to reconsider computational thinking as a "universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use." She posited that, as a field of study "[o]ne can major in computer science and go on to a career in medicine, law, business, politics, any type of science or engineering, and even the arts" (p. 35).

Since its re-emergence, computational thinking has become pervasively adopted and employed throughout K-12 education, though not without challenges. Riding alongside the large-scale push for prolific Science, Technology, Engineering, and Mathematics (STEM) initiatives, activities, and training opportunities, its popularity had seemingly overreached its operational understanding. Misuses and misunderstandings remain throughout K-12 curricula, particularly because of the loosely defined "habits of mind" stemming from an inconsistent operational definition (Denning, 2017). Since computational thinking was not explicitly defined by Wing (2006), its interpretation varied wildly until undergoing refinement years later (Aho, 2012; Royal Society, 2012; Wing, 2011). As an accepted operational definition, Wing (2011) later clarified, "Computational thinking is the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (p. 1). Although computational thinking remains far reaching, the limits on which this problem-solving approach can be applied to educational contexts is bounded by research in the cognitive sciences. There currently exists no evidence to support prior debates within the field which propose an ability of computational thinking to predict student transfer of learning to new content and between learning contexts (Guzdial, 2008). Denning (2017) posits that computational thinking's primary benefit is to those who "design computations," but asserts that claims of benefits to non-designers have yet to be substantiated (p. 38).

Grover and Pea (2013) mostly acknowledge an agreement between computer science educators and researchers on the following elements of computational thinking as supporting

student learning and understanding of computational thinking practices or habits of mind: (a) abstractions and pattern generalizations; (b) systematic processing of information; (c) symbol systems and representations; (d) algorithmic notations of flow of control; (e) structured problem decomposition (modularizing); (f) iterative, recursive, and parallel thinking; (g) conditional logic; (h) efficiency and performance constraints; and (i) debugging and systematic error detection (pp. 39-40). What is often confused today when designing curricula that address computational thinking practices is the dissolution of computer programming from computational thinking. Computer programming skills, although distinctive from the general computer science aims, are inseparable from any application of computational thinking. Grover and Pea (2013) challenge the notion that programming is simply a utility in support of computer science when they posit "Programming is not only a fundamental skill of [computer science] and a key tool for supporting the cognitive tasks involved in [computational thinking] but a demonstration of the computational competencies as well" (p. 40).

Since Wing's (2006) article reviving computational thinking, the National Science Foundation and the College Board partnered to develop a course built around a framework supporting new computing methodologies and computational thinking practices. Abstraction and algorithmic thinking, with roots in the seminal discoveries of Papert and Perlis, are central computational thinking practices within this new course – AP Computer Science Principles. Designed using a Universal Design for Learning framework, the course was created around seven "big ideas" in computing which the curriculum framers believed students should be able to articulate and apply to real-world scenarios. These big ideas are (a) creativity, (b) abstraction, (c) data and information, (d) algorithms, (e) programming, (f) the Internet, and (g) global impact. The release of the Computer Science Principles course in 2016 for general offering contrasted with the traditional AP Computer Science A course, which focused primarily on the interpretation and development of programs (logically-situated) using an object-oriented programming framework. The Computer Science A course had been the sole AP computer science course offering since the 2009-10 academic year.

Contrasting Computer Science A and Computer Science Principles

The AP Computer Science courses represent the only broadly adopted computer science framework or curriculum in U.S. high schools (Nager & Atkinson, 2016), which are offered throughout grades 9-12 for advanced study of post-secondary computer science concepts and principles. Although both courses are considered the equivalent of an introductory level college computer science course, they vary significantly in their design, scope, and sequence. The traditional Computer Science A course is structured around the paradigm of object-oriented programming in a subset of the Java programming language, teaching students how to solve problems through the development of computational solutions in and around multiple disciplines. This course requires all students to attain some level of proficiency in a designated, high level programming language (Java). Conversely, the Computer Science Principles course was designed to provide flexibility for the educator in choosing between several approaches (e.g., project-based, integrated, or inquiry-focused) for organizing instruction around a programming

DOI: 10.26716/jcsi.2019.02.1.2

language-agnostic set of computational thinking practices and major areas of study (seven big ideas). This course encourages teachers to "select a programming language(s) that is most appropriate for their classroom and that will provide students opportunities to successfully engage with the course content" (College Board, 2017b, p. 38). The Computer Science Principles framework provides a list of 13 different programming languages/platforms that can be considered for use in the course, which include some low-level block-based coding platforms often used in elementary and middle schools (e.g., Scratch, Snap!, and Alice) as well as the object-oriented Java programming language (p. 39).

The overarching goals of the two courses are described differently as well. Computer Science A is described as "intended to serve both as an introductory course for computer science majors and as a course for people who will major in other disciplines and want to be informed citizens in today's technological society" (College Board, 2014, p. 6). By contrast, the Computer Science Principles curriculum is designed such that "students will develop computational thinking skills vital for success across all disciplines…[and] will also develop effective communication and collaboration skills by working individually and collaboratively to solve problems, and will discuss and write about the impacts these solutions could have on their community, society, and the world" (College Board, 2017b, p. 4). These divergent philosophies, a problem-solving (pragmatic) versus human-computing (holistic) foci, have situated the Computer Science Principles course to become one which "aims to broaden participation in the study of computer science" (College Board, 2017a). An overview comparison of the two courses is provided in Table 1.

***AP Exam Components*.** Fundamental to both courses is their multi-dimensional approach to assessing student understanding of the curriculum. With a problem-solving focus, Computer Science A uses a more traditional AP assessment format containing multiple-choice and free response sections, an hour and a half dedicated to each, with each part representing 50% of the final assessment and the end-of-course score. These scores are summed and normalized to a value between 1 (no recommendation) and 5 (extremely well qualified), and recorded as an assessment of the individual students' ability to master the content material of the course. The multiple-choice section contains 40 questions based on the course learning objectives assessing the ability to understand, interpret (trace), and debug code segments. The free response section contains 4 questions focused on the application of the content material to a set of problem preconditions, propelling students to design, synthesize, and apply programming concepts to these problem spaces. Although student scores are determined exclusively through their performance on a three-hour proctored exam, a recently amended laboratory requirement provides students the opportunity to apply and synthesize programming concepts to real-world problem tasks, which is intended to prepare them for similar mental tasks on the free response section of the exam.

Table 1

Crosswalk of AP Computer Science Curriculum (Overview)

Course	Computer Language	Prerequisites	Lab Requirement [1]	Computational Thinking (CT) Practices[2]	Computing Principles	Assessments	Assessment (%/hrs)[3]	CS Program[4]
AP Computer Science Principles	**Agnostic**	**Completed Algebra** (algebraic functions & problem-solving strategies)	**None** (see assessments – 20 hrs of performance tasks)	• Abstraction • Algorithms • Analyze Data • Represent Data • Decomposition • Testing	• Creativity • Abstraction • Data and Information • Algorithms • Programming • The Internet • Global Impact	**Explore** - Impact of Computing Innovations **Create** – Application to Ideas **AP CSP Exam**	**16**/8 **24**/12 **60**/2 (MC only)	"…complements AP Computer Science A as it aims to broaden participation in computer science."
AP Computer Science A	**Java**	**Basic English and Algebra** (algebraic functions)	**20 hours** (hands-on, structured)	• Abstraction • Algorithms • Decomposition • Testing • Parallelization • Simulation	• Object oriented programming • Program Analysis • Data Structures • Operations and Algorithms • Computing in Context	**AP CSA Exam**	**100**/3 (MC and FR)	"…focus on computing skills related to programming in Java"

Note.

[1] Three labs as applications of the content material: Magpie (string methods), Picture (arrays), and Elevens (object-oriented programming)

[2] Computational Thinking practices are assessed using the ISTE Framework (collect data, analyze data, represent data, decomposition, abstraction, algorithms, automation, testing, parallelization, and simulation).

[3] Percentage of the overall CS course AP score (1-5) and the number of in-class/proctored hours to complete the assessment.

[4] AP Computer Science courses may be taken in any order, each course is stand-alone (College Board, 2014; College Board, 2017a).

By contrast, the Computer Science Principles scoring structure is determined through a combination of in-class performance assessments, totaling 40% of the final score, and a proctored multiple-choice exam (75 questions) representing the final 60% of the score. The multiple-choice exam is focused on the understanding, interpretation, and application of Computer Science Principles concepts. Attributing 40% of the exam's final AP score to a pair of extended in-classroom tasks represents a significant departure from the traditional exam, which bases its final AP score entirely upon performance on the proctored exam. The assessment of programming in the Computer Science Principles course occurs in one of the in-class performance assessments, completed over several days, creating an assessment environment that is less controlled in terms of potential external influences on assessment results. Allowing students to collaborate on the programming task also raises questions as to the level of individual programming proficiency acquired by students who rely too heavily on classmates.

The content assessed on the two AP exams also represents a major potential difference in how performance results may be interpreted. Computer Science A requires students to take an assessment on their understanding of a specific, high level, object-oriented programming language (Java), in a proctored setting. Figure 1 depicts a short snippet of Java code syntax, illustrating the format of the kind of syntax students would need to understand. The Java code represents exactly the same syntax that is used to create commercial software, providing the students with highly transferable technical knowledge should they decide to further pursue programming academically or professionally. Conversely, Computer Science Principles is programming language-agnostic, allowing teachers to decide which language is appropriate for their students. One of the acceptable options teachers may consider, Scratch, is depicted in Figure 2, displaying the same "programming" functionality shown in the Java snippet in Figure 1. A Scratch program can be created by dragging the colored blocks shown in the figure to a linear stack in the order the user wants the actions performed. Students are allowed to "create" a program using the selected platform over several days, and they are encouraged to collaborate on parts of this task.

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);

    System.out.print("How old are you?: ");
    int age = in.nextInt();

    if (age < 16) {
      System.out.println("Sorry, you are not
      quite old enough to drive!");
    }
    else {
      System.out.println("Yeah! Happy driving!");
    }
  }
}
```

Figure 1. Example of Java code developed using *repl.it*.

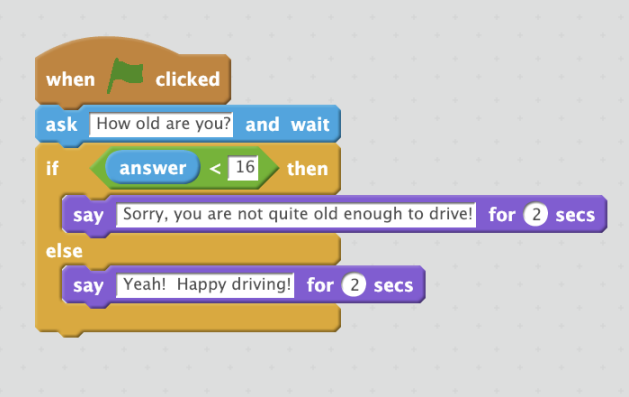

Figure 2. Example of coding in the block coding platform *Scratch*.

This block-coding platform, and others listed as acceptable in Computer Science Principles, are far less transferable due to their simplification and insulation of syntax to facilitate ease of use. Encouraging teachers to use platforms they believe appropriate for their students potentially introduces teacher bias into curriculum design in classrooms where teacher expectations are not high. It also allows minimal investment from teachers in becoming proficient in more complex (and more transferable) object-oriented programming options such as Java.

The in-class tasks scored as part of the AP Computer Science Principles exam are comprised of more than just the programming task. There are two in-class tasks: The "Explore" task (8 class hours) and the "Create" task (12 class hours). Overall, these performance tasks are designed to have students analyze an innovation, describe its impact on people and society, and create a computer program explaining the most "significant aspects" which allow it to run (College Board, 2017b). Through the "Explore" performance task, students choose an innovation (physical computing or non-physical computing) to evaluate by creating a "computational artifact" such as a digital poster and written responses to prompts. Students are "expected to complete the task with minimal assistance from anyone" (p. 108). Within the "Create" performance task, students are required to create a software program around a topic of interest. This program can be created using the language/platform selected as appropriate for the class by the instructor. The program guidelines indicate "You are strongly encouraged to work with another student in your class…It is strongly recommended that a portion of the program involve some form of collaboration with another student in your class, for example, in the planning, designing, or testing (debugging) part of the development process" (p. 113). At the end of the course, the tasks are submitted to the College Board for external scoring. Since the Computer Science Principles performance tasks are completed internally (within the classroom) and assessed for creativity (one of the seven big ideas), it affords a level of flexibility to the educator and student in selecting material that is relevant to the individual; such relevancy is perceived to have previously been a significant barrier to ensuring broad access to the curriculum.

Course-Specific Curriculum. A more detailed look at the differences between the two courses can be seen in Table 2, comparing the big ideas of Computer Science Principles with those of Computer Science A. This qualitative comparison reveals some side-by-side similarities in computational thinking topics such as abstraction, decomposition, and algorithmic thinking. There is a notable disparity in programming content, depth, and application in Computer Science Principles compared with Computer Science A across the big ideas. Much of the Computer Science Principles curriculum is observed to occur outside of the programming space and to a much shallower depth than that of Computer Science A. Computer Science Principles provides a more generalized, conceptual curriculum, situating the big ideas in context but with less programming application. Computer Science A provides an applied approach, with content material almost entirely devoted to its programming application to solving multi-disciplinary problems. It does not advance, nor in some cases cover, the more holistic components of the Computer Science Principles course (i.e., those learning objectives in and around the human-

DOI: 10.26716/jcsi.2019.02.1.2

Table 2

Comparison of Big Idea Applications to Programming (AP CSP and AP CSA)

Big Ideas[1]	AP CSP	In-Programming[2]	AP CSA	In-Programming
Creativity	Focus on the creative development process, tools, and techniques for the creation of digital artifacts (not limited to a program, image, audio, video, presentation, or Web page file).		Not assessed in the AP CSA curriculum.	
Abstraction	In-programming abstraction is limited in scope and depth, not to include a discussion of reference parameters. Multiple levels of abstractions are suggested including constants, expressions, statements, procedures, and libraries.	✓	In-programming abstraction is rigorously applied through an object-orientated programming approach. Students design a class, understand and implement inheritance and composition relationships in the creation of program.	✓
Data and Information	Methods of information processing and data visualization outside the programming space, extraction of information from data using software (conceptually limited, does not include specific formulas), and analyze the manipulation of data.		In-programming primarily situated within standard data structures seeking the understanding and application of Java class methods, and managing data with 1-D, 2-D arrays and the `ArrayList` class.	✓
Algorithms	Through the expression and development of an algorithm in a programming language, in-programming learning objectives support solutions to computational problems. Limitations to their uses are also discussed.	✓	Focused on operations on data structures, knowledge of the two-standard searching (sequential, binary) and three sorting algorithms (selection, insertion, merge) and how to implement them into a program.	✓
Programming	A focus on programming for creative expression (human-computer perspective) is mirrored through the "Create" performance assessment. Develop a program (through collaboration) to solve a problem, explain how programs implement algorithms, use abstraction to effectively manage complexity in programs, employ mathematical and logical concepts (basic arithmetic and logic operations), and evaluate program correctness.	✓	A focus on designing a program which can solve a problem (pragmatic perspective) given a set of preconditions or constraints. An extensive overview of object-oriented (and procedural) programming extending beyond basic algorithms and logical operations to their application in data (in multi-dimensional arrays), programming abstractions (inheritance and abstract classes), and evaluation (search and sort algorithms).	✓
The Internet	Characteristics of the internet, its systems, and analysis of concerns such as cybersecurity.		Not assessed in the AP CSA curriculum.	
Global Impact	The impact of computing on innovations in other fields, how people participate in the problem-solving process, and the benefits and harmful effects of computing.		The impacts of computing to the Internet, economic and legal impacts of viruses, life-critical applications, and intellectual property.	

Note.

[1] The seven big ideas from the AP Computer Science Principles curriculum is adopted and applied to AP Computer Science A.

[2] In-programming acknowledges the inclusion of programming tasks/instruction within a big idea.

computer interface). This dichotomy compels a deeper study into the overall depth of knowledge obtained by students embarking on either Computer Science A or Computer Science Principles.

Method

Depth of Learning

Exploring the course curricula in greater detail, Bloom's Revised Taxonomy was employed to evaluate the learning objectives of Computer Science Principles as compared to those of the Computer Science A course. The goal for employing Bloom's Revised Taxonomy was to further compare the courses in terms of student potential depth of knowledge through a well-established cognitive learning tool used prolifically by K-12 educators. As detailed earlier, this revised taxonomy was applied to the learning objectives in the course descriptions (textual in the case of Computer Science A and tabular for Computer Science Principles) producing a "depth of knowledge" score on a cognitive scale of 1 (remember) to 6 (create). For example, Computer Science Principles learning objective 2.2.3 states students will "[i]dentify multiple levels of abstractions that are used when writing programs" (College Board, 2017a). This learning objective, when evaluated using Bloom's Revised Taxonomy, would receive a depth of knowledge score of 1 as "identification" asks students to simply retrieve or recall information stored in long-term memory. Conversely, learning objective 4.2.4 states that students will "Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity" (College Board, 2017b). A learning objective which prompts students to "evaluate," or cognitively make judgements based on a predetermined set of criteria, would receive a score of 5, a higher cognitive task than recall.

Results

Following the coding and Bloom's taxonomic score determination process for each learning objective, a mean score was codified for each Computer Science course. Table 3 provides an example of this process through a textual analysis of keywords presented in each learning objective. Using the guidelines of the revised taxonomy to determine an average depth of knowledge score, Computer Science Principles curricular material was determined, on average, to fall within a value of 3-4, whereas Computer Science A revealed an average score between 4-5 (see Appendix A for complete results). These results highlight an emphasis of Computer Science Principles on applying knowledge and analyzing information, whereas Computer Science A places a stronger emphasis on analyzing and evaluating. This apparent shift in perspective (from analyzing to evaluating) may be realized through the distribution of scores presented in Figure 1. The differing distributions of the analyzed content along the Bloom continuum highlights a shift in the conceptualized depth of knowledge between the two courses.

Table 3
Comparison of "Selected" Learning Objectives (AP CSA and AP CSP)

Program	Topic Area	Learning Objective	Bloom's Revised Taxonomy Score
AP Computer Science Principles	Algorithms	**LO 4.2.4.** Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity.	5
		LO 4.2.3. Explain the existence of undecidable problems in computer science.	2
	Abstraction	**LO 2.2.1.** Develop an abstraction when writing a program or creating other computational artifacts.	6
		LO 2.2.3. Identify multiple level of abstractions that are used when writing programs.	1
AP Computer Science A	Program Analysis	"Examining and testing programs to determine whether they correctly meet their specifications."	6
		III.B. Debugging including error categories, error identification and correction, and evaluating code using techniques (e.g., debugger, output statements, or hand-tracing).	
		III.F. Interpret preconditions and postconditions when provided as pseudo code.	2
	Program Implementation	"The implementation of solutions in the Java programming language reinforces concepts, allows potential solutions to be tested, and encourages discussion of solutions and alternatives."	5
		II.A. Statement of solutions in a precise form for evaluation using the following techniques: top-down, bottom-up, object-oriented, encapsulation, and procedural abstraction.	
		II.C. Appropriate use of Java library classes and interfaces to solve a problem.	3

Note. Key words used in the coding of each learning objective (Bloom's revised taxonomy score of 1-6) is identified by an underline.

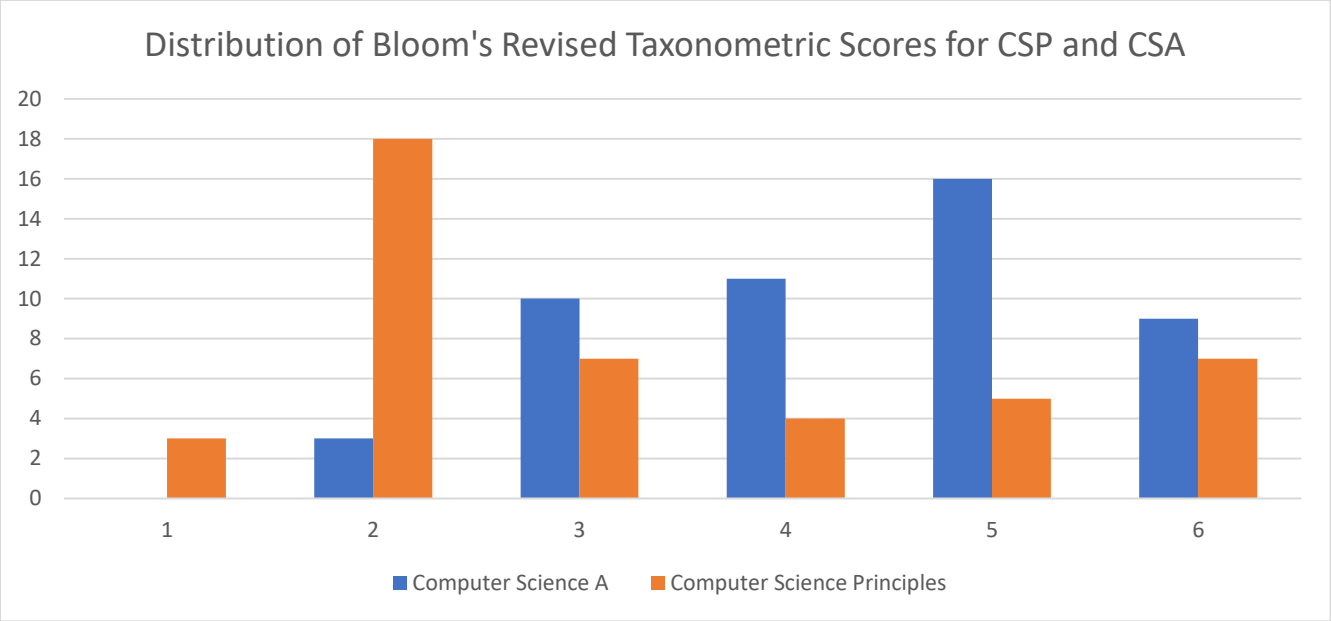

Figure 3

Discussion

The juxtaposition between both Computer Science Principles and Computer Science A through a depth of understanding analysis is important when considering how far students' exposure to computer science ultimately takes them, both academically and professionally. Given the importance of preparing the next generation of STEM professionals, of particular importance is the level of STEM content preparation being afforded to students in post-secondary education. Given the stark differences between the two AP computer science courses, especially as it relates to how each one approaches the level of depth afforded to learning programming, the results of our analyses reveal a discernible difference in both the depth and foci of the two courses, with Computer Science A being more focused on pragmatic aspects of programming, utilizing a context more easily transferable to more advanced study in computer programming. The Computer Science Principles course was found to be broader in its coverage of the field of computer science, while less focused on the specific skillsets and platforms that could provide the foundation for further and deeper study.

Conclusions

An in-depth analysis of the activities and assessments associated with the two AP computer science options provides support for the notion of two-tiered preparation, despite both courses being identified as equivalent to introductory college-level courses. Research on changes in participation reveal a significant increase in access to Advanced Placement computer science curricula by traditionally underrepresented groups of students. An in-depth content analysis of rigor (or depth of assessed knowledge), however, has indicated a much different picture.

Differences in the assessment methods, including the attribution of 40% of the Computer Science Principles score to two tasks completed in the classroom over several days, has resulted in marked differences in the distribution of scores between the two exams (as reported in Howard & Harvard, 2019). Although the in-class assignments are scored by the same subject-matter experts as the traditional exam, allowing students to complete them over several days relinquishes some control over whether the students seek external help between class meetings. The encouragement of collaboration on these tasks further distinguishes Computer Science Principles as computer science "light" in terms of its level of challenge and preparation for students. Furthermore, the content of the two exams is very different in emphases as well. Whereas the Computer Science A course assesses students' ability to design, write, and analyze programs using Java programming language, the Computer Science principles course only requires students to write a program in one of the two in-classroom tasks, completed collaboratively, using a teacher-chosen platform from among a wide range of options in terms of complexity. This raises questions as to how prepared students taking Computer Science Principles are to later succeed in postsecondary STEM majors that lean on programming proficiency, as well as to how much credence postsecondary institutions should place in passing scores on the Computer Science Principles exam.

Given the increasing importance of computer science, and in particular, computer programming as a high-demand and highly technical field, it is imperative that school counselors are aware of the substantive differences in the two AP Computer Science course offerings as they advise their students. For the increasing number of students with prior coding or computer science experience through elementary or middle school curricula, Computer Science A may be the most beneficial option. For students with minimal prior exposure to the field, perhaps both courses in sequence is advisable, provided that both are offered at their schools. It is likewise important for schools and school districts to carefully consider the potential limiting effect of selecting Computer Science Principles as their sole AP Computer Science course offering. In order to ensure equitable opportunities for students to excel in this important field in higher education and in the workplace, having the opportunity to choose the best option for their respective academic and professional paths is critical.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074

Bureau of Labor Statistics. (2018). Occupational Outlook Handbook: Computer and Information Technology Occupations. Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/home.htm

College Board. (2014). AP Computer Science A course description. Retrieved from https://apcentral.collegeboard.org/pdf/ap-computer-science-a-course-description.pdf?course=ap-computer-science-a

College Board. (2017a). AP® computer science principles. Retrieved from https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-overview.pdf?course=ap-computer-science-principles

College Board. (2017b). Course and exam description: AP computer science principles. Retrieved from https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf?course=ap-computer-science-principles

College Board. (2018a). AP Archived Data. Retrieved from https://research.collegeboard.org/programs/ap/data/archived

College Board. (2018b). *AP Program Participation and Performance Data 2018*. Retrieved from: https://research.collegeboard.org/programs/ap/data/participation/ap-2018

Cuny, J. (2015). Transforming K-12 computing education: AP® computer science principles. *ACM Inroads, 6*(4), 58-59. https://doi.org/10.1145/2832916

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33-39. https://doi.org/10.1145/2998438

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38-43. https://doi.org/10.3102/0013189X12463051

Guzdial, M. (2008). Education: Paving the Way for Computational Thinking. *Communications of the ACM, 51*(8), 25-27. https://doi.org/10.1145/1378704.1378713

Howard, K. E., & Havard, D. D. (2019). Advanced Placement (AP) Computer Science Principles: Searching for Equity in a Two-Tiered Solution to Underrepresentation. *Journal of Computer Science Integration, 2*(1), 1-15. https://doi.org/10.26716/jcsi.2019.02.1.1

Miller, G. A. (2003). The cognitive revolution: A historical perspective. *Trends in Cognitive Sciences, 7*(3), 141-144. https://doi.org/10.1016/S1364-6613(03)00029-9

DOI: 10.26716/jcsi.2019.02.1.2

Minsky, M. (1974). A Framework for Representing Knowledge. Retrieved from https://dspace.mit.edu/bitstream/handle/1721.1/6089/AIM-306.pdf?sequence=2

Nager, A., & Atkinson, D. R. (2016). The Case for Improving U.S. Computer Science Education. *Information Technology & Innovation Foundation*, 1-38.

National Economic Council and Office of Science and Technology Policy. (2015). *A strategy for American Innovation*. Retrieved from https://obamawhitehouse.archives.gov/sites/default/files/strategy_for_american_innovation_october_2015.pdf

National Science Foundation. (2014). College Board launches new AP Computer Science Principles course. Retrieved from https://www.nsf.gov/news/news_summ.jsp?cntn_id=133571

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc.

Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning, 1*(1), 95-123. https://doi.org/10.1007/BF00191473

Papert, S., & Harel, I. (Eds.). (1991). *Constructionism*. Norwood, NJ: Ablex Publishing Corp.

Royal Society. (2012). Shut down or restart: The way forward for computing in UK schools. Retrieved from http://royalsociety.org/education/policy/computing-in-schools/report/

Tedre, M., & Denning, P. J. (2016). *The long quest for computational thinking.* Paper presented at the Proceedings of the 16th Koli Calling International Conference on Computing Education Research, Koli, Finland. https://doi.org/10.1145/2999541.2999542

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2011). Research notebook: Computational thinking—What and why? Retrieved from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Appendix A

Depth of Knowledge Course Comparison

A Bloom's Revised Taxonomy was utilized to compare the cognitive depth of knowledge addressed by the course learning objectives for AP Computer Science Principles and AP Computer Science A. Tables A.1 and A.2 detail the course learning objectives with a cognitive score following the revised taxonomy between 1 (remember) and 6 (create).

Table A1. Computer Science Principles Framework and Depth of Knowledge

Big Idea	Learning Objective	Bloom's Revised Taxonomy Score
1. Creativity	LO 1.1.1. Apply a creative development process when creating computational artifacts.	3
	LO 1.2.1. Create a computational artifact for creative expression.	6
	LO 1.2.2. Create a computational artifact using computing tools and techniques to solve a problem.	6
	LO 1.2.3. Create a new computational artifact by combining or modifying existing artifacts.	6
	LO 1.2.4. Collaborate in the creation of computational artifacts.	5
	LO 1.2.5. Analyze the correctness, usability, functionality, and suitability of computational artifacts.	4
	LO 1.3.1. Use computing tools and techniques for creative expression.	3
2. Abstraction	LO 2.1.1. Describe the variety of abstractions used to represent data.	2
	LO 2.1.2. Explain how binary sequences are used to represent digital data.	2
	LO 2.2.1. Develop an abstraction when writing a program or creating other computational artifacts.	6
	LO 2.2.2. Use multiple levels of abstraction to write programs.	3
	LO 2.2.3. Identify multiple level of abstractions that are used when writing programs.	1
	LO 2.3.1. Use models and simulations to represent phenomena.	3
	LO 2.3.2. Use models and simulations to formulate, refine, and test hypotheses.	3
3. Data and Information	LO 3.1.1. Find patters and test hypothesis about digitally processed information to gain insight and knowledge.	4
	LO 3.1.2. Collaborate when processing information to gain insight and knowledge.	2
	LO 3.1.3. Explain the insight and knowledge gained from digitally processed data by using appropriate visualizations, notations, and precise language.	2
	LO 3.2.1. Extract information from data to discover and explain connections or trends.	2

DOI: 10.26716/jcsi.2019.02.1.2

		LO 3.2.2. Determine how large data sets impact the use of computational processes to discover information and knowledge.	2
		LO 3.3.1. Analyze how data representation, storage, security, and transmission of data involve computational manipulation of information.	4
4.	**Algorithms**	LO 4.1.1. Develop an algorithm for implementation in a program.	6
		LO 4.1.2. Express an algorithm in a language.	2
		LO 4.2.1. Explain the difference between algorithms that run in a reasonable time and those that do not run in a reasonable time.	2
		LO 4.2.2. Explain the difference between solvable and unsolvable problems in computer science.	2
		LO 4.2.3. Explain the existence of undecidable problems in computer science.	2
		LO 4.2.4. Evaluate algorithms analytically and empirically for efficiency, correctness, and clarity.	5
5.	**Programming**	LO 5.1.1. Develop a program for creative expression, to satisfy personal curiosity, or to create new knowledge.	6
		LO 5.1.2. Develop a correct program to solve problems.	6
		LO 5.1.3. Collaborate to develop program.	5
		LO 5.2.1. Explain how programs implement algorithms.	2
		LO 5.3.1. Use abstraction to manage complexity in programs.	3
		LO 5.4.1. Evaluate the correctness of a program.	2
		LO 5.5.1. Employ appropriate mathematical and logical concepts in programming.	3
6.	**The Internet**	LO 6.1.1. Explain the abstractions in the Internet and how the Internet functions.	2
		LO 6.2.1. Explain characteristics of the internet and the systems built on it.	2
		LO 6.2.2. Explain how the characteristics of the Internet influence the systems built on it.	2
		LO 6.3.1. Identify existing cybersecurity concerns and potential options to address these issues with Internet and the systems built on it.	1
7.	**Global Impact**	LO 7.1.1. Explain how computing innovations affect communication, interaction, and cognition.	2
		LO 7.1.2. Explain how people participate in a problem-solving process that 4scales.	2
		LO 7.2.1. Explain how computing has impacted innovation in other fields.	2
		LO 7.3.1. Analyze the beneficial and harmful effects of computing.	4
		LO 7.4.1. Explain connections between computing and real-world contexts, including economic, social, and cultural contexts.	2
		LO 7.5.1. Access, manage, and attribute information using effective strategies.	1

	LO 7.5.2. Evaluate outline and print sources for appropriateness and credibility.	5

Table A2. Computer Science A Framework and Depth of Knowledge.

Big Idea	Learning Objective	Bloom's Revised Taxonomy Score
1. Object-Oriented Program Design	Program and Class Design	6
	Problem analysis	4
	Data abstraction and encapsulation	6
	Class specifications, interface specifications, relationships ("is-a", "has-a"), and extension using inheritance	5
	Code reuse	6
	Data representation and algorithms	6
	Functional decomposition	5
2. Program Implementation	Implementation techniques	5
	Top-down	5
	Bottom-up	5
	Object-oriented	6
	Encapsulation and information hiding	5
	Procedural abstraction	6
	Programming constructs	2
	Primitive Types vs. Reference types	4
	Declaration (constants, variables, methods, classes, interfaces)	3
	Text output using System.out.print and System.out.println	4
	Control (method call, sequential and conditional execution, iteration, and recursion)	4
	Expression evaluation (numeric, String, Boolean expressions and DeMorgan's Law)	5
3. Program Analysis	Testing	4
	Development of appropriate test cases, boundary cases	4
	Unit testing	4
	Integration testing	4
	Debugging	5
	Error categories: compile-time, run-time, logic	5
	Error Identification and correction	5
	Techniques such as using a debugger, hand tracing code	5
	Runtime exceptions	2
	Program correctness (pre- and post-conditions, assertions)	2
	Algorithm analysis (execution counts and run time comparisons)	4

		Numerical representations of integers	4
4.	**Standard Data Structures**	Primitive data types (int, boolean, double)	5
		Strings	5
		Classes	6
		Lists	6
		Arrays (1-dimensional and 2-dimensional)	6
5.	**Standard Operations and Algorithms**	Operations on data structures	3
		Traversals	3
		Insertions	3
		Deletions	3
		Searching (sequential and binary)	3
		Sorting	3
		Selection	3
		Insertion	3
		Mergesort	3
6.	**Computing in Context**	System reliability	4
		Privacy	5
		Legal issues and intellectual property	5
		Social and ethical ramifications of computer use	5