Part 1: Inequitable Learning Outcomes for AP Computer Science Principles

Consider the following mean AP Computer Science Principles Exam Scores disaggregated by ethnicity/race. The mean scores in all three columns are virtually identical. Note: 3 or higher is a passing score.

	Mean AP CSP Exam Score		
Race/ Ethnicity	**[1]Chicago Public Schools 2018**	**[2]LAUSD 2018**	**[2]LAUSD 2019**
Asian	3.2	3.21	3.24
White	3.4	3.31	3.36
Black	2.1	2.13	2.10
Hispanic	2.3	2.07	2.11

Table 1. Mean AP CSP Exam Scores for CPS (2018) and LAUSD (2018 & 2019). Disaggregated by Ethnicity/Race.

Sources:

1. Steven McGee, Randi McGee-Tekula, Jennifer Duck, Lucia Dettori, Andrew M. Rasmussen, Erica Wheeler, and Ronald I. Greenberg. "Study of access and outcomes from advanced computer science coursework in the Chicago Public Schools" poster in Structured Poster Session CS for All: An intersectional approach to unpacking equity in computer science education. In American Educational Research Association (AERA) Annual Meeting, April 2019.

2. Portnoff, Scott R. A New Pedagogy to Address the Unacknowledged Failure of American Secondary CS Education. ACM Inroads 11, 2 (June 2020), Pages 22-45. ACM New York, NY, USA. doi: 10.1145/3381026

Exam scores correspond to a narrow range of passing rates, as seen in the more detailed LAUSD data below.
Approximately 69-85% of Asian and White students received passing exam scores.
Approximately 30-36% of Black and Hispanic students received passing exam scores.

Ethnicity	**Exams Taken**	**Exams Passed**	**Exam Pass Rate**	**% A, B or C Letter Grade**	**Mean Score**	**Mean Pass Score**
African American	70	21	**30.0%**	**84.3%**	**2.13**	3.57
Asian	117	81	69.2%	96.6%	3.21	3.81
Filipino	52	32	61.5%	100.0%	2.75	3.47
Hispanic	649	203	**31.3%**	**84.7%**	**2.07**	3.31
White	138	107	77.5%	95.7%	3.31	3.78
Other	10	7	70.0%	—	3.20	3.43
Total LAUSD	1036	451	43.5%	—	2.41	3.53
National	72187	51383	71.2%	—	3.11	3.69

Table 2. LAUSD 2018 AP CSP Exams by Ethnicity.

Ethnicity	Exams Taken	Exams Passed	Exam Pass Rate	% A, B or C Letter Grade	Mean Score	Mean Pass Score
African American	80	25	**31.3%**	**83.8%**	**2.10**	3.36
Asian	180	129	71.7%	97.8%	3.24	3.82
Filipino	68	47	69.1%	92.6%	2.93	3.47
Hispanic	946	341	**36.0%**	**89.3%**	**2.11**	3.23
White	159	134	84.3%	97.5%	3.36	3.66
Other	30	26	86.7%	—	3.30	3.62
Total LAUSD	1463	702	48.0%	—	2.45	3.46
National	95105	69059	71.9%	—	3.11	3.68

Table 3. LAUSD 2019 AP CSP Exams by Ethnicity.

The CPS investigators found a correlation of exam scores with both GPA and school quality. This finding is unremarkable. De facto segregated schools are inherently unequal, due primarily to economic/income factors – disproportionate numbers of Black and Hispanic families are low-income. Alexandria Ocasio-Cortez has borne witness in numerous interviews that changing zip codes, i.e. moving from the Bronx to Yorktown, a small city 30 minutes north, where she was one of the few Latinx students in her high school, made all the difference in terms of her educational opportunities. The primary objective of AP Computer Science Principles (AP CSP) – indeed its raison d'être – was to broaden participation of historically underrepresented minorities (URMs), particularly those in underserved communities. Part and parcel of that goal was to produce equitable learning outcomes.

The teacher model for AP CSP is to provide one or two weeks of professional development to teachers with no particular expertise in Computer Science. A widely used metric to gauge poor school quality is the number of teaching faculty who are not fully credentialed in the core subjects (math, science, ELA, social studies, art) – e.g. interns or credentialed teachers holding emergency credentials for a subject they are not credentialed to teach. Yet the teacher model for AP CSP relies on exactly these kinds of teachers – notwithstanding the increasing numbers of state boards of education offering supplementary authorizations in Computer Science. Although poor school quality is an overwhelming and constraining determinant for learning outcomes – and future upward mobility – it is not absolute destiny. Certainly, Jaime Escalante's successful development of the AP Calculus program at Garfield High School demonstrated that a dedicated, qualified and talented teacher at a failing school can create a pocket of excellence in a poor school district. Building from the ground up.

I am hardly the first to argue that AP CSP does little to further the goal of mitigating inequities in computing participation rates in college or employment. The reason: it does not prioritize teaching students how to program, the core skill of the discipline and one which, by the way, is not easy to learn. One critic has argued – even after seeing the data cited above – that the value of the course lies in getting students "in the door". However, even were one to concede this point – i.e. even on the course's own terms – the failure data and the inequitable learning outcomes show that 65-70% of URMs are not even getting in that door.

The course was under development for approximately a decade, garnering the support of millions of grant dollars from NSF. It's hard, therefore, to understand – assuming the course was field-tested in majority-minority urban public schools – why these glaringly inequitable learning outcomes weren't detected and dealt with prior to a full national rollout of the program by the College Board. In hindsight, though, the situation was not entirely unpredictable, even to those with a passing knowledge of the history of American education. Offering basic high school classes in underserved communities staffed with instructors lacking full subject matter competence is a decades-old familiar formula of America's urban public school system.

As recently as this year, researchers have articulated a view that questions the value of the course:

> … our findings also indicate that students historically underrepresented in computing are disproportionately enrolled in a course that is less likely to prepare them for college-level computing majors, as other researchers have found that the two courses are differentially accepted for college credit and as prerequisites for college-level computing courses [9]. This raises important questions about what happens when students do not have the opportunity to also take CSA, particularly since those in our sample who took only CSP reported less experience with actual coding and much less interest in pursuing computing majors or technology careers.[3]

Source:

3. Linda J. Sax, Kaitlin N.S. Newhouse, Joanna Goode, Max Skorodinsky, Tomoko M. Nakajima, and Michelle Sendowski. 2020. Does AP CS Principles Broaden Participation in Computing? An Analysis of APCSA and APCSP Participants. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20). Association for Computing Machinery, New York, NY, USA, 542–548. DOI:https://doi.org/10.1145/3328778.3366826

Were a modern-day de Tocqueville to tour America's public high schools and take an inventory of CS programs, she would find a long-familiar pattern in large metropolitan areas. Inner city schools might offer ECS and/or AP CSP, but rarely AP CSA; while schools with robust AP CSA programs are situated in wealthier suburbs and isolated high-performing academic pockets within large urban school districts. Note that this two-tier system is a recapitulation of the situation described well over a decade ago in Jane Margolis book "Stuck in the Shallow End", which detailed the structural inequities in secondary CS education. The analogy, by the way is exact: the book concerned itself entirely with unequal access to the AP CSA course. Plus ça change, plus c'est la même chose.

As a side note, these same authors also speculated:

> Further, given a recent College Board report stating that students taking CSP are substantially more likely than other AP STEM students (and students in general) to subsequently take CSA [5], one might view CSP as a critical gateway course to more advanced computing curricula.[3]

There's no argument that there are substantial numbers of students who first take CSP and then CSA. First though, if we are talking in the same breath about successful CSA and CSP programs, then we are referring to high schools in academically desirable zip codes or isolated high performing academic pockets. That is, we are not talking about broadening participation to students historically underrepresented, i.e. Black, Hispanic and first-generation or immigrant students in underserved communities.

Second, CSP is decidedly not a "critical gateway" course – rather it is often the only other CS course that's available. Quite the opposite in fact, the word in discussions among CSA teachers is that students can perform brilliantly in CSP and fail miserably in CSA. This past spring, in fact, I remotely tutored a very talented CSA student in this exact situation who, because of health reasons, had missed substantial in-person class sessions, and so had crucial gaps and misunderstandings and was failing the class (she earned a 5 on the exam, another happy client). She told me point blank that CSP had been no help to her whatsoever.

The reason that CSP is neither critical, useful nor effective is that it does not lay the requisite groundwork for subsequent success in a rigorous programming course (CSA) because it does not address the substantial problems and obstacles that most novice programming students encounter. This is why I have argued for its replacement with a pre-AP CSA programming course that will identify and effectively tackle those obstacles.

Such a course would be a much more authentic way to get students “in the door”, as it both more accurately reflects the discipline, and gives students better preparation for a subsequent college prep CS course, which will inevitably require them to master fundamental programming concepts and skills.

As an anecdotal example of the relationship of the two courses, the CS chair at one of the consistently highest-performing AP CSA programs still left in LAUSD* has told me that CSP takers that subsequently do well in CSA are all “highly academic” and tend to be “into math and tech and want to take both”. That is, taking CSA was not contingent on those few students first taking CSP. Note that most students taking CSA at this school are from its highly gifted magnet. CSP, on the other hand, has no better gender parity at this school than CSA, about 25%. CSP is more diverse, the CS chair reported, but CSP minority students who subsequently took CSA did so in substantially lower numbers.

* The 3 largest high-performing public high schools in the district (read: majority-majority schools) seceded to become charter schools, taking their large CSA programs with them.

Part 2: Learning Failure Data for Exploring Computer Science (ECS)

Data regarding learning outcomes for Exploring Computer Science are harder to come by. Nonetheless, the same group of dedicated Chicago educators cited earlier have produced several research studies.

Learning Growth: from a D- to a D+

Source:

4. Steven McGee, Randi McGee-Tekula, Jennifer Duck, Catherine McGee, Lucia Dettori, Ronald I. Greenberg, Eric Snow, Daisy Rutstein, Dale Reed, Brenda Wilkerson, Don Yanek, Andrew M. Rasmussen, and Dennis Brylow. 2018. Equal Outcomes 4 All: A Study of Student Learning in ECS. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). Association for Computing Machinery, New York, NY, USA, 50–55. DOI: https://doi.org/10.1145/3159450.3159529

This study measured "the growth of computational thinking practices from pretest to posttest". The average pretest score was 15.2 out of 25 (60.8%). The average posttest score was 17 out of 25 (68%). The growth of 1.8 points (7.2%) was found to be statistically significant. The paper states, without providing scores disaggregated by ethnicity, that among all demographic groups except Hispanic (whose learning appears to have regressed somewhat, perhaps an artifact), there were no statistically significant differences in learning.

The sample comprised students from both Chicago and Wisconsin, but absolute numbers for either group were not listed. Notwithstanding, it appears that the sample comprised approximately equal numbers of students from each geographic group. Relative to their numbers in Chicago Public Schools (CPS), white students were about 4 times as likely to be overrepresented in the CPS portion of the sample (43%), and Black students about 4 times less likely (6%). The sample included 49% Hispanic and 10% Asian students. Some students selected multiple demographic categories.

The average 7.2% learning growth in the course increased the pretest grade from a D- to a posttest grade of D+. The paper unconvincingly characterized this as a "large learning gain". Most evaluators, however, would label this level of achievement neither advanced, proficient nor basic, but "below basic". Moreover, any teacher worth their salt would be taken aback at what most would regard as a huge learning failure, and immediately start scrambling to work out what might be done to improve teaching in the next cycle. Many parents would be beating down the instructor's door as well, desperate for ways to help their children do better.

One might also question the value of a course 60% of whose content is material that students already know prior to taking the course.

52% could not identify a Conditional Statement; 79% could not identify a Loop

Source:

5. Steven McGee, Ronald I. Greenberg, Randi McGee-Tekula, Jennifer Duck, Andrew M. Rasmussen, Lucia Dettori, and Dale F. Reed. 2019. An Examination of the Correlation of Exploring Computer Science Course Performance and the Development of Programming Expertise. In SIGCSE '19: 50th ACM Technical Symposium on Computer Science Education, February 27–March 2, 2019, Minneapolis, MN, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287415

This study measured competencies for 7 tasks, described below. The demographic breakdown for the Chicago Public Schools sample was 20% African-American, 58% Hispanic, 15% White and 5% Asian. White students were twice as likely to be overrepresented, and Black students half as likely.

In the Results section, the paper listed only the numbers in the Pretest and Posttest columns in Table 5 and claimed substantial learning. This is a curious interpretation, because, even ignoring baseline/pretest levels,

posttest scores were still less than 50% for 4 of the 7 tasks. Actual learning can be more accurately determined, however, by taking pretest scores as baseline – knowledge/skills that students already possess pre-course. The last 2 columns show that, on average, at least half of the students did not learn what was not already self-evident about 5 of the 7 tasks.

Note that these tasks were extremely simplified. Consider: the **conditional** and **loop** tasks merely asked students to identify which step in an algorithm was a conditional or loop statement – a far cry from correctly implementing these control structures in a Scratch program or writing code snippets or pseudocode. This identification task, then, was not a big ask for an introductory high school CS course – one might even characterize it as embodying low expectations. Yet ignoring baseline, only 48% could identify the conditional and only 21% the loop posttest. Actual learning for these 2 tasks was even worse when taking into account pretest/baseline scores: 33% and 15% respectively. Similarly, less than half of students were successful in the **Meet Requirements** and **Missing Inputs** tasks, again ignoring baseline.

Conversely, it's also worth noting that for the 3 tasks where posttest scores were above 50%, pretest baseline scores were markedly higher than for the other 4 tasks, a red flag for how much more self-evident these particular tasks already were. The baseline scores were 38%, 46% and 62% for the **General Requirements**, **Available Inputs** and **Determine Output** tasks, respectively.

CONTRAST these outcomes with a study of students of the same age taking a 30-hour Scratch **programming** curriculum – similar to the time ECS allots to its Scratch unit. Tasks were of a substantially higher order of thinking and understanding than the Chicago Public Schools study above, including defining CS concepts and writing actual functioning programs. For **Loops**, the study found: "… in the pretest, we saw that – as expected – the students did not have preexisting knowledge of CS concepts … In the posttest 57.5% of the students correctly defined bounded loops and 75% correctly defined conditional loops."

Source:

6. Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2010. Learning computer science concepts with scratch. In Proceedings of the Sixth international workshop on Computing education research (ICER '10). Association for Computing Machinery, New York, NY, USA, 69–76. DOI:https://doi.org/10.1145/1839594.1839607

It's hard to see how a credible case, therefore, can be made that ECS authentically broadens participation, disrupts inequities or adequately prepares students for college when – even on the course's own terms – a majority of students failed to learn most of the very simple concepts tested. Moreover, I have to take issue with the last two words of the paper's title – *Programming Expertise* – because the tasks are too simple to measure anything resembling "*expertise*" and are a far cry from actual "*programming*". Perhaps a more accurate phrase would be "Recognizing Basic Introductory Programming Concepts".

Like AP CSP, the ECS Teacher Model often relies upon teachers with considerably less than a full grasp of the discipline, even more so than with AP CSP. This certainly doesn't help. It's worth repeating that America's class- and geographically-based public educational system tightly couples school quality with zip codes: students at high schools in underserved communities have few choices beyond basic classes more often staffed with teachers lacking full subject matter competence, both of which leave them unprepared for college. This is just as true now of secondary CS education as when Margolis first documented the phenomenon, despite the substitution of computer application classes with the new non-programming survey courses.

TASK	Reported Results		% students below baseline	Learning for students below baseline			
	Pretest (baseline)	Post test		Success	Failure	Success Rates	Failure Rates
Determine Output	62	87	38	25	13	**65.8%**	**34.2%**
Meet Requirements	14	**34**	86	20	66	**23.3%**	**76.7%**
Available Inputs	46	78	54	32	22	**59.3%**	**40.7%**
Generate Requirements	38	69	62	31	31	**50.0%**	**50.0%**
Missing Inputs	20	**45**	80	25	55	**31.3%**	**68.8%**
Conditional	22	**48**	78	26	52	**33.3%**	**66.7%**
Loop	7	**21**	93	14	79	**15.1%**	**84.9%**

Table 5. Seven Learning Tasks.

TASK DESCRIPTIONS

(**Determine Output**) Program comprehension: Students were provided with a written algorithm. They were given an input value and asked to determine what the program **output** would be.

(**Meet Requirements**) Program comprehension: Students were given a scenario in which three drivers with cars needed to pick up 12 passengers for a concert and no more than five people could be in one car. Given an approach to efficiently picking up the passengers, students need to determine whether the algorithm will **meet the criteria**.

(**Available Inputs**) (**Missing Inputs**) Planning: Students used the same driver and passenger scenario. They were asked to determine which **inputs were provided** in the scenario and which important **inputs were not provided** in the scenario.

(**General Requirements**) Planning: Students were asked to provide **requirements** for a program that would help a teacher track student information.

(**Conditional**) **(Loop)** Program generation: Students were provided with an algorithm that is abstracted from any given language. They are then asked to decide which step in the algorithm would use a Scratch **conditional** block and which step would use a Repeat-Until **Loop**.

NOTE: In 2009-10, I took the 2-week ECS UCLA summer training and once-a-month PD. In 2015, I took the summer 3-week PLTW CSP training at Cal Poly Pomona. I mention this to underscore that I am not exactly approaching this as an outsider. That said, I believe the inequitable learning outcomes for AP CSP and the learning failures for ECS presented and discussed in this article were not unforeseeable, due to both the nature of survey courses – which prioritize breadth over depth – and the pedagogic/instructional approach employed in both courses.

The problem, I believe, stems from a failure to acknowledge that learning and retention of fundamental concepts and skills requires lots of practice, repetition and ongoing usage.

Generally, each survey course moves from one fairly unrelated unit to another, with few dependencies or connections, if any, between them. Although it's possible to go deep at any point, both AP CSP and ECS emphasize breadth. Neither course incrementally builds skills the way one does in math, science, foreign language or ELA, i.e. like a house: laying a foundation, building the first floor, then the second, and so on. Each unit in these courses instead focuses on a single final project as the primary point of learning. This pedagogic strategy is as ineffective as pure direct instruction. What both instructional modes share is that they expect students to learn material at their first and only encounter, something that rarely happens. Long-term retention

of learning requires not only repetitive practice, but revisiting material, often at a more complex and advanced level. Use it or lose it. The trouble is that spending more time on depth requires one to sacrifice breadth.

The main selling point for ECS and CSP is that they would stimulate interest in CS. Interest, though key, is never enough to retain students: they need to also feel an authentic sense of self-efficacy vis-à-vis programming concepts and skills, something which these courses are not designed to deliver. Consider the related passage:

> *A student who consistently struggles with schoolwork will understandably begin to lose confidence in her academic abilities.* ***The most effective way to lift her confidence is to help her improve her skills. Success is the best antidote to low self-esteem, and academic mastery is the best remedy for low academic self-confidence*** *(p. 205).*
>
> *Shore, K. (1998). Special Kids Problem Solver. San Francisco: Jossey-Bass.*

All said, I'm not suggesting that CSP change its pedagogical approach, because that would only address the symptom and not the problem. Instead:

1. I would like to see CSP replaced by an introductory programming course that aligns vertically with AP CSA. The version of this course that I teach gets students "in the door", but I believe in a more authentic way. First, they learn and develop self-efficacy and self-confidence vis-à-vis foundational aspects of programming / the beginnings of programming literacy (what I believe those are is the stuff of past and future articles). Second, the course can illustrate, through real-world programming examples, the ways that CS already operates in the world and the impacts it can have, or already has, on a myriad of other disciplines from art to apps to biology.
2. I would also like to see NSF stipulate that the millions of dollars it hands out to CSP and ECS be allocated to support tuition for teachers of those courses, allowing them to earn state-approved supplementary authorizations in CS (or the equivalent), so that they can effectively transition to teach both an introductory pre-AP CSA programming course and AP CSA.
3. A long-standing assumption for endeavors at secondary CS reform has been that academia should be the source of new curricula – handed down to secondary instructors like Moses delivering the Law at Mt. Sinai.[1] In fact, though, such courses already exist, their authors being a slew of talented CS high school teachers with years of experience testing and refining them under their belts. Surely a working group of such educators might compare notes and arrive at a consensus on a national introductory programming curriculum that can deliver equitable learning outcomes within a relatively short space of time.

One argument that proponents of CSP have used to rationalize the course amounts to what I see as a false either/or choice: a survey approach or a programming focus. An introductory programming course can actually be designed to do both, and I believe, much more effectively. It can provide an introduction to CS/programming in modern society, as well as teach students the basic skills and programming concepts that will prepare those who want to pursue subsequent opportunities in high school, college and beyond.

Although survey course proponents have continued to hawk an argument advanced two decades ago that students in programming courses must "master abstract programming for programming's sake" – something that hasn't been even remotely true for a decade – current high school introductory programming courses have featured engaging and accessible cross-curricular projects/units in graphics; steganography and encryption; 2D and 3D dynamic art; programmatic construction of gene maps from human genome data; molecular modeling; astronomy; geography; analysis of health-related data, crime data, etc.; cybersecurity and on and on.

---

[1] This is not to say that academics have not created amazing curricula: U of Washington's Larry Snyder's *Steganography* lesson has become one of the AP CSA labs. UC Berkeley's *Beauty and Joy of Computing* course, a variant of CSP, is a well-designed programming curriculum. I'd be curious to know, though, how it performs in schools in underserved communities, and how well it helps underrepresented struggling students overcome common novice programming misunderstandings and obstacles.