Justice-Centered Educational Programming Language Design

AMY J. KO, University of Washington, Seattle, USA
R. BENJAMIN SHAPIRO, University of Washington, Seattle, USA
JAYNE EVERSON, University of Washington, Seattle, USA
F. MEGUMI KIVUVA, University of Washington, Seattle, USA

Educational programming languages (EPL), for all their success in enabling computing education at scale, regularly exclude learners by embedding assumptions about ability, class, culture, language fluency, and identity. Further, most EPL designs are not governed in ways that are responsive to the needs of learners and their communities on the margins of computing, raising questions about how the design processes behind EPL could be organized to ensure they serve everyone equitably. Building upon discourse on diversity, educational justice, and design justice, we propose seven justice-centered design requirements for EPL, arguing that they should be *accessible*, *liberatory*, *transparent*, *cultural*, *obtainable*, *democratic*, and *enduring*. For each, we examine why these requirements are necessary and offer examples of languages that do and do not meet them. Throughout, we surface constraints that EPL impose on being justice-centered and grand challenges for research to be able to overcome them.

CCS Concepts: • **Social and professional topics → Computing education**; • **Software and its engineering → Development frameworks and environments**; **Compilers**.

Additional Key Words and Phrases: programming languages, equity, justice

ACM Reference Format:
Amy J. Ko, R. Benjamin Shapiro, Jayne Everson, and F. Megumi Kivuva. 2026. Justice-Centered Educational Programming Language Design. *J. ACM* 37, 4, Article 111 (August 2026), 29 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Broadening participation in computing is a profoundly complex endeavor [134], requiring massive, sustainable investments in teacher learning [78], curricula [119], communities [6], policy [38], and a rich ecosystem of informal learning [12]. It requires practice-centered research to ensure that change is effective, sustainable, and spreadable [22, 97]. Doing this equitably is an even grander challenge, given the broad lack of literacy about what equity means among teachers, school leaders, not-for-profits, policy makers, and computer scientists [135]. This requires reckoning with systems of power we need to dismantle in order to achieve equity [124] and routine resistance to that change felt daily by students, teachers, and advocates on the margins of computing.[1]

[1]The first author served as this journal's Editor-in-Chief at the time of publication. To avoid conflicts of interest, the article was independently and confidentially edited by acting Editor-in-Chief, Andrew Petersen; the first author had no visibility into its review and no authority in its evaluation.

Authors' Contact Information: Amy J. Ko, ajko@uw.edu, University of Washington, Seattle, Seattle, Washington, USA; R. Benjamin Shapiro, rbs@cs.washington.edu, University of Washington, Seattle, Seattle, Washington, USA; Jayne Everson, everjay@uw.edu, University of Washington, Seattle, Seattle, Washington, USA; F. Megumi Kivuva, megumik@uw.edu, University of Washington, Seattle, Seattle, Washington, USA.

Manuscript submitted to ACM

While all of these efforts are essential to broadening participation in computing education, there is one element, however, that receives comparatively little attention in reform efforts: *educational programming languages* (EPL). In computer science, "PL" often refers only to the academic field of programming languages, or the syntax and semantics of languages. Here we use the phrase as teachers and learners often refer to it, including the programming language design, but also its implementations in compilers, interpreters, and translators, associated tools (e.g. editors and debuggers), libraries, documentation, and standards governance that shape the language.[2] By *educational* PL, we mean any PL that either was designed explicitly for learning (e.g., Pyret, Scratch), or is used in pedagogical contexts (e.g., Java in an AP CS A course). [48, 144].

Of course, EPL do receive some attention in research and advocacy. Some of us design, build, and maintain them [39, 60, 73, 136, 147], some of us create learning technologies for them (e.g., [93]), many of us study how people learn them (e.g., [42, 109, 153]. But the majority of educators, students, and informal learning leaders do not participate in the design or implementation of PL, nor is PL design included in advocacy efforts [23, 115]. Instead, most efforts take EPL as givens, choosing from what is available to meet teaching or learning goals, and working around their constraints, rather than considering them something that can be redesigned to be more equitable.

Perhaps that is inevitable: creating PL, especially educational ones, requires immense expertise, time, and resources [98]. Nor is it necessarily desirable to have subtly different EPL for every classroom, school, district, or country: standardizing them has benefits, given the educator learning and curriculum development required to teach computing through them [146]. However, by concentrating the design of PL amongst a small group in academia and industry, there is a great power imbalance between designers and the vastly larger group of teacher and student stakeholders.

While addressing this power imbalance alone is definitely not sufficient for broadening participation, EPL are a necessary part of broadening participation: they are foundational media, notation, and learning technologies for computing education, and shape all of these other efforts in crucial, and often subtle ways. Consider, for example, the choice in Python to evaluate empty lists to *False*: this single choice has led to countless learners trying to debug this subtle type juggling behavior, teachers seeking professional development to improve their debugging pedagogy, and even curriculum design to address these design choices. Or, consider the early choice by Alice 2.0 and Scratch to only support drag-and-drop code editing: while this had significant positive impacts on learners who could use a mouse, it profoundly excluded any learner who could not. If such specific programming language and tool design choices can structure participation so forcefully, and without the voice of those excluded, the design of EPLs must play a necessary but insufficient role to equitably broaden participation.

This leads to a fundamental research question of EPL design governance: *How can the work of EPL design be equitably organized to meet the diverse learning needs of all learners?* One historical answer to these questions is that we should design *many* EPL. Indeed, over past decades, there have been hundreds, each with different design goals [71]. Some prioritize teaching and instruction [60]. Others prioritize creativity and youth identity [92]. Others still prioritize particular media to support practice-linked identity [108] and skill development in computing, e.g., by concentrating on media manipulation [52], data analysis [20], or storytelling [72]. This multiplicity of designs *has* managed to serve many different distinct communities of teachers and learners [48, 144]. Recent calls have even argued for intentionally creating a multiplicity of small languages that meet a diversity of cross-disciplinary instructional needs [53].

[2]Prior work has sometimes referred to this broader scope as a programming "system" [80]; we use "language" to situate our arguments in terms more familiar to communities of learning, at the expense of some precision.

While this heterogeneous approach has decentralized power in some ways, it does not serve everyone. In fact, most EPL fall into the familiar trap of utilitarian capitalism, designing for the majority (to achieve the "greatest good" [133]), at the expense of those at the margins, whose needs are perceived to be in tension with the majority's. For example:

- Few EPL serve learners with *disabilities* (e.g, who are blind and rely on screen readers, have motor-physical impairments and rely on speech input, or who have dyslexia or color blindness and may need fine-grained control over font or color choices [127, 136]).
- Few EPL attend to *neurodiversity*, offering learners with ADHD[3] control over the design of interfaces to regulate their attention, autistic learners agency over how information is communicated, or risk averse learners the ability to learn without tinkering [17].
- Most EPL ignore *language fluency*, typically using English keywords, and the few that do have localizations in other languages may only localize syntax and some documentation, but not deeper cultural assumptions [54].
- EPL often overlook *economic inequality*: learners who do not have personal computers or access to the Internet cannot access computing education [158]. Other EPLs require specific, expensive hardware to be usable.

Learners at the intersection of any of these forms of marginalization may find that they face unique compounding challenges with EPL that make participating in computing education exhausting, infeasible, or impossible, in or out of school [8, 54]. Their teachers and families may find themselves searching desperately for *some* solution to include them in learning, only to find a fragile ecosystem of poorly supported workarounds.

Addressing these problems in EPL design is what we will call *justice-centered educational programming language design* (JCEPL). We intend this phrase as a provocation for EPL designers to earnestly engage questions of justice as they design and maintain EPL, both in the design choices, and governance over those choices. In this paper, we aspire to deconstruct, define, and examine this provocation in order to provide concrete guidance and insight about what it means to do JCEPL, technically, sociotechnically, and sociopolitically. We build upon other recent efforts to deconstruct PL design from a feminist lens [61], questioning whose voices are engaged in shaping PL and what gender stereotypes are embedded in PL design. We extend this discourse to the many other systems of injustice in which PL are designed, while focusing on education.

We focus on three questions:

- *RQ1*: How should we conceptualize JCEPL design?
- *RQ2*: What requirements does justice-centered impose on EPL design?
- *RQ3*: What constraints do current PL design and implementation practices and contexts impose on being justice-centered and what grand challenges for research do those constraints imply?

To answer these questions, we begin with a survey of conceptions of justice, educational justice, justice-centered computing education, and EPL design. We then offer one conception of JCEPL design, deriving EPL-specific design guidance from the broader project of *design justice* [25], which itself builds upon an discourse of equity and justice. We then present an analysis of inspiring and weak cases of JCEPL in popular use, examining the extent to which they meet these requirements, and the challenges of improving. We end with a discussion of the implications of our conception for research and practice. Our contribution is not definitive answers to these questions, but catalyzing answers, helping our community deepen its commitments to educational justice by examining the technologies at the heart of computing education.

[3]Attention-deficit/hyperactivity disorder

2 Background

Here we set the stage for a set of proposed requirements for JCEPL, defining terms and concepts and reviewing prior work on justice-centered computing education and JCEPL design.

2.1 Diversity, Inclusion, Equity, Justice, and Design

It is not possible to adequately define justice in a short background section. Instead, we provide a basic overview of key ideas, grounding our argument in particular theories of justice in general, in relation to design, and in relation to education.

Our work draws upon Rawls' theory of justice [125], because of his focus how design choices in society directly impact whether society is just. We begin with the term *diversity*, which is central to how Rawls conceptualizes inequality. In this paper, we define *diversity* as a fact of variation in human bodies, minds, experiences, cultures, and values. It is a fact that is often ignored, dismissed, and neglected in society in favor of hegemonic, normative views of human experiences, but a fact nonetheless (e.g., some people are tall, some are short, but built environments often privilege particular heights). *Inclusion* is any effort to include people as social or technical systems as they are, by acknowledging diversity, and by creating pathways into systems for people being excluded. While inclusion welcomes those excluded, it does not necessarily change systems to account for diversity (e.g., welcoming short people into a space designed for tall people). *Equity* is a property of social systems in which everyone has the opportunities, resources, and rights they need to achieve equal outcomes, even if that means some people receive more resources than others (e.g., stools for short people). *Justice* is a property of social systems in which the need for targeted resources to account for diversity is minimal because the impact of diversity on access and outcomes has been eliminated (e.g., installing automatic height adjusting shelving).

While theories of justice from philosophers like Rawls' are important conceptual foundations, they are generally ahistorical, missing crucial history, context, and nuance about identity and power [133]. Theories of justice centered on identity — for example, disability justice (e.g., [57]), racial justice (e.g., [166]), gender justice (e.g., [139]) — offer this crucial social context, deconstructing the status quo and offering visions of more just futures. All generally stem from observations about intersecting forms of oppression and social hierarchy in the design of social systems and culture (including PLs, which are cultural artifacts). But they also reinforce that justice is not a set, unchanging, unambiguous goal. Rather, it is a socially situated, negotiated, and contested one that evolves as society changes, but also changes individuals, as we begin to better understand it and ourselves with greater clarity [63]. We view EPLs as just one of the social systems in which the oppressive hierarchies of the past and present are encoded, privileging particular groups over others, reifying inequities and injustices at play in broader society. We view changes in EPLs as a necessary part of understanding what computing is in the world and what our relationship to it could be.

We also build upon the broader theories of *educational justice*. The ideas in this paper are partly inspired by thinkers and activists like Freire [47], who rejected banking models of learning (depositing facts into students' minds but not seeing them as bringing valuable resources with them into the classroom), and advocated for critical consciousness through liberatory, community-centered discourse. Freire's ideas suggest an inversion of EPL design, centering learners' "limiting situations", rather than the discipline's authoritative judgment of relevance or meaning. We also build upon bell hooks' critiques of teaching [63], which connected more abstract notions of learning as liberation to the more concrete realities of racial and patriarchal capitalism that play out in classrooms and broader culture. These critiques suggest a need for EPL design to contend with the broader systems in which they are situated. Our ideas were also shaped by

Ladson-Billings' seminal work on critical race theory in education [84], which identifies racial social constructions as a structural force, suggesting PL as a form of racialized "property." We also build upon her work on culturally responsive pedagogy [83] and Gay's culturally responsive teaching [49], which conceptualize teaching and learning that centers care, cultural competency, and student agency. These core values, building upon Collins' *Black Feminist Thought* [24], suggest a vision of EPL design that not only attends to learners' differences, but puts that difference at the center of EPL design and use. We also learn from literature in education sociology and policy [4, 100, 162], which examines racial and gender justice in formal systems of education, but also economic rights, immigrant rights, mass incarceration, environmental justice, disability rights, and other forms of inequity in society. This literature argues that systems of education built upon inequitable systems in society inevitably produce unequal outcomes in learning, reproducing inequalities in society, rather than rectifying them.

Finally, we look to *design justice*, conceptualized by Costanza-Chock [25], which builds on these many notions of justice, and directly addresses design, unlike these other conceptions of justice. Its framework explains how the *activity of design* supports some groups while burdening others by reifying white supremacy, heteropatriarchy, capitalism, ableism, settler colonialism, and other kinds of inequalities built into our social structures. It offers principles by which design can be practiced in community-led ways to overcome these inequities, creating opportunities for meaningful participation in design decisions. Design justice's focus on community is in contrast to other approaches to design. This contrast includes *human-centered design*, which tends to reserve power over a design for designers, and frame stakeholders strictly as sources of design insight, and *participatory design*, which despite the shift in power, still tends to frame community participants in extractive terms. Design justice also problematizes *universal design* [65] by questioning whether universality is possible, let alone desirable: grounding designs in commonalities between communities' needs overlooks sometimes inherent tensions between groups (e.g., captions include deaf and hard of hearing attendees, but may distract audience members with ADHD).

2.2 Justice-centered computing education

Recently, computing education scholars have begun to examine equity and justice directly. There have been calls to engage computing critically [81, 82, 87, 99, 129], examining intersections between computing education and culture, identity, and power. For example, Rankin et al. examined three sites of violence in CS (K-12 schools, predominantly white institutions, and internships), deconstructing how power operated in each to produce racial inequality [124]. Scott et al. revisited culturally responsive pedagogy through the lens of collaboration, mentorship, and empowerment through identity [137]. Kirdani-Ryan and Ko [76] examined how departmental norms constrain and shape students' career goals, reproducing power structures molded by market forces, and displacing students' values, identities, and communities. Jacob et al. examined how English-primacy shaped multilingual students' perceptions of computing, themselves, and the support of their families and communities [69]. These each illustrate how structural forces in computing education such as power, norms, and language can shape student learning. These studies, and recent critiques in the PL research community [61], raise questions about how EPL design might inadvertently embed the same power structures in their design, reifying and reinforcing those structures in their use in classrooms.

Some works have focused on visions of more justice-centered computing literacy. For example, Vakil called for making space in education for sociopolitical aspects of computing systems [155, 156]. Yadav et al. called for computing education that critically examines computing in the context of community and citizenship [170]. Morales-Navarro and Kafai offer three concrete strategies for bringing criticality into CS classrooms through inquiry, design, and re-imagination [104]. Eglash et al. called for explicitly counter-hegemonic framings of education through valuing individual

and community assets [30]. Ko et al. described efforts to prepare computing educators to examine tensions between technical, sociotechnical, and sociopolitical framings of CS in the classroom [78]. Shea et al. examined the unjust dominant narratives and invisible labor in STEAM infrastructure, including how even micro-controllers marketed as "low cost" can be unaffordable to or unusable in settings with minimal access to computing and shoestring budgets [143]. Other works disrupt traditional forms of classroom and school authority and focusing students on envisioning futures [18, 32, 35, 130, 149], centering youth ingenuity, counter-narratives, design partnership, and implicit power structures. Some examine the challenges of building relationships across culture and language [5, 159]. Others examine the broader family systems that enable learner-centered creative empowerment with code [131]. Each of these works find that shifting students' mindsets from one of power*less* to power*ful* can create conflict, requires time, and depends on careful framing of the computing technologies brought into the classroom. These works might predict much the same in justice-centered EPL design, where changing what EPL and and who they are designed for may surface new tensions in identity and learning.

While these works are innovative on numerous dimensions – pedagogy, assessment, programs, and power – few challenge the role of EPL in shaping and constraining what learning is possible. Rather, EPL are framed as curated selections by educators, sometimes as a subject of critique, but not as tools of, and sites for, resistance or change. Papert's positioning of Logo in *Mindstorms* [117] is a possible exception to this. Although he did not use the word justice, there were clear elements of justice in his arguments, advocating for learner agency, advocating against education that devalued youth creativity, and arguing for the unique role of programmable media in helping learners to construct knowledge and identity. Papert distilled these ideas into three principles: 1) the *continuity* principle, which argues that learning must be aligned with learners' personal knowledge, 2) the *power* principle, which argues that the concepts in a programmable system must empower learners to create things of personal meaning, and 3) the *cultural resonance* principle, which argues that learning with programmable media must make sense in the larger social context in which a learner is situated. These principles, and Papert's application of them to EPL, point to a notion of justice-centered computing education that is responsive to learners and their contexts, and offers them tools that are not in and of themselves new ways of thinking, but enable learners to construct new ways of thinking and relating to others [171].

Wilensky's[4] theory of "restructurations" offers a similar learner-centered lens as Papert, but about the representation of disciplinary ideas [165]. The theory centers on five properties of how learning is structured: 1) the expressive *power* of a representation, 2) the *cognitive* affordances of the representations for learning, 3) the *affective* influence of engagement and interest of a representation, 4) the behavior of representations in *social* contexts, 5) and how representations support the *diversity* of human identity, values, culture, and thinking. While none of these explicitly address justice or injustice, they are a complementary "thinking tool" for analyzing the learning affordances of idea representations, and how they are affected by human difference.

2.3 Inclusive programming language design

Many programming languages, including many EPL, have elements that attend to inclusion, such as accessibility and cultural diversity. Such focus, however, is often the exception, and not the rule, and often elevates human-centered methods within existing frameworks of power, rather than attempting to disrupt those frameworks.

For example, the broader PL research community has been primarily concerned with correctness, performance, security, and other technical qualities [138], but not justice. In research, there are some efforts to use human-centered

[4]We note the irony of two successive paragraphs highlighting the ideas of abusive men in a paper about justice. We include these ideas for completeness, but not as endorsement of their behavior.

design methods to align PL with productivity needs [107]. This includes design methods such as *PLIERS*, [21], which offers a structured technique for need-finding, risk analysis, and refinement, and the more analytic framework of *Cognitive Dimensions of Notations*, which provides conceptual tools for reasoning about notation design [51]. These approaches are notable in that, unlike most PL research, they involve explicit reasoning about, and sometimes engagement with, stakeholders in PL design. Practitioner communities that govern PL through community processes are probably the closest to examining questions of social power, though scholarly work to examine power in these communities is nascent [67].

Some PL design has embraced concerns of equitable teaching. The efforts of the *How to Design Programs* (HtDP) team, including the Bootstrap project, offer several examples [39]. *Racket*'s collection of language levels offers a sequenced design of PLs, aligned with a learning trajectory crafted to manage cognitive load. Bootstrap's subject-specific courses, such as *Bootstrap Physics*, are designed in close collaboration with physics teachers. Guzdial's teaspoon languages [53] similarly center teacher voices in shaping EPL design.

Some PL design focuses on blind and vision impaired (BVI) creators [91, 141]. For example, *Quorum* has focused on being more screen-readable [147], and innovations such as tangible media for computation [122], programmable audio games [70], tweaks to PL that enable mixed-ability families to code together [128] offer visions of more accessible programming. Other works explore how to make existing PL and tools more screen readable [9, 31, 102, 106, 106, 123, 136]. Some work finds that programmable media needs to build upon youth interests and needs [150, 151, 157], comprehensively support screen reading [56], and that achieving these benefits requires new languages and editors, rather than retrofitted existing ones. Despite progress on BVI learners' needs, prior work has given little focus to neurodiversity, reading disabilities, color perception, motor abilities, cognitive impairments, and attention. Moreover, these innovations are often about making existing PL more accessible, rather than designing more intrinsically accessible PL. One exception to this is recent work by Hadwein et al., which reframes computing education around embodied cognition and disability studies, and elevates the importance of sense, sensemaking, and materiality in the design of accessible programming tools and pedagogy [55].

There is also a growing literature on multilingual learners in computing education. Recent EPL, for example, have been designed to be explicitly multilingual [60, 77]. Most work on linguistic justice, however, has focused on instruction, and not tools. For example, there is evidence that English primacy harms learning [3, 54], and that learning in students' first languages promotes learning, engagement, and belonging [2, 27, 50, 75, 95, 112, 145, 160], but that English primacy in programming and learning materials limits their impact [86]. While these benefits are clear, many learners legitimately see learning English as a goal [154] and many approaches to including learners' first languages inequitably impose translation labor onto youth [120], most of which cannot be automated without compromises to content quality [121]. None of this literature engages the intersection between language and accessibility or teacher facilitation needs.

Finally, Hermans and Schlesinger's essay on feminism in PL design is of particular note [60]. They make the case for engaging feminist ideas to reimagine PL design methods, posing questions about linguistic inclusion, the ways PL embed stereotypes about gender and masculinity, the bias in what research questions are asked about PL design, and the overarching epistemic bias of PL design toward mathematical ways of knowing. They also discuss the many ways that domination features prominently in PL design communities, as structural, disciplinary, and hegemonic norms. These critiques reflect the ways that attending to diversity of human experiences in PL design continues to be an act of resistance to a particular masculinized vision of what PL are and who they are for.

Aside from this handful of works, nearly all EPL have focused on innovations in new media, graphical editors, and runtime capabilities [71], and not on ensuring that our ecosystem of EPLs offers options that work for every learner. The

1.	*Design to heal and empower communities* → Design EPL as a tool of liberation in computing
2.	*Center direct stakeholders voices* → Center learner, teacher, and family voices in EPL design
3.	*Prioritize community impact over design intent* → Prioritize learners' identities, needs, and goals over PL innovation
4.	*View partnership as ongoing accountable collaboration* → Sustain ongoing partnerships between PL designers, learners and their communities
5.	*Frame designers as facilitators, not leaders* → Frame PL designers as in service of learners and their communities
6.	*Value stakeholders' lived experiences* → Value learner and community knowledge as assets for PL design
7.	*Share design knowledge with communities* → Empower learners and teachers to contribute to PL design
8.	*Work toward community-led, sustainable outcomes* → Work toward community-led EPL maintenance and funding
9.	*Solutions should reconnect communities rather than exploit* → EPL design should grow communities rather than extract from them
10.	*Designers should understand a community's existing solutions before building new ones* → PL designers should understand community needs before building

Table 1. Paraphrased design justice principles and corresponding JCEPL principles.

focus of these works on equity and inclusion alone, then might be viewed as upholding broader systems of domination, in that they are not being designed to advance educational justice, but rather help more people participate in unjust systems of learning. In this way, many EPLs might be viewed as a "master's tool" [89]: one designed to encourage and require conformity, rather than dismantle the matrix of oppression that shaped its design. A call for JCEPL, within this frame, is also a call to reimagine EPL design in more liberatory terms.

3 *ALTCODE*: Justice-Centered Educational Programming Language Design

Prior work on justice [125], educational justice [47, 49, 63, 83, 84], and PL design [61] teaches us many things: 1) power is unequally distributed and that the systems we create – including EPL – reflect those hierarchies of power; 2) efforts to redistribute power to youth are needed and possible, but broader social systems outside classrooms – including EPL – are barriers to that change; and 3) design methods for engaging stakeholders in the design of PL are only just emerging. Just as re-imagining CS teaching and pedagogy in justice-centered ways has revealed a wide gap between current practice and the more joyful dreams that students have for CS learning [37], there is a big gap between current EPL and justice-centered EPL designs.

How, then, can we organize EPL design to overcome these issues? Just as how reforming CS teaching [124, 137] requires a reconsideration of whose voices shape the classroom, we argue that the gap between current EPL design practice and educational justice is *access to power*. At the heart of this is a fundamental tension: few people have the knowledge to design and build EPL that work for all youth. And yet, for EPL to serve all youth, a diversity of youth and teacher voices must hold power through meaningful influence over the design, implementation, evolution, and selection of EPL. That might mean *directly* participating in shaping EPL, or it might mean students, teachers, and their communities, organizing and shaping design goals and constraints that reflect their needs, and then having those needs be realized in collaboration with EPL experts. And that means addressing *how* to share power, as youth often feel as though they do not have or deserve power [35], and teachers often feel they do not have the PCK to teach CS, let alone design EPL [169]. Similar questions arise in calls for teaching to be culturally relevant [83] and sustaining [118]: we know we need to restructure access to power, but how?

Costanza-Chock's *Design Justice* [25] offers one lens for how to redistribute power, bringing together communities of stakeholders to shape design. This mirrors justice-centered computing education work, which shifts power towards students and teachers, and away from curricula and standards [137]. One manifestation of design justice is the *Design Justice Network*'s cultivation of ten principles for design practice that center power in communities. Table 1 paraphrases them and shares possible analogs for JCEPL design. These principles help answer *RQ1*: we can conceptualize JCEPL design as *community-led*, de-centering EPL designers' expertise, intentions, and timelines in favor of learners' and their communities', reframing PL designers as partners *in service of* community needs, values, and goals. This is far from how most EPL designers organize governance: most centralize decision-making in ways that are responsive to community feedback, but are not necessarily accountable to it.

While this answer to *RQ1* offers value as a "north star", it raises several questions about how to apply them to EPL design, including how to apply these principles; which communities should be served and how big they can be; how EPL teams might collaborate with communities; what knowledge communities might need to shape EPL; and what a "community" even is, given the often global scope that many EPL attempt to serve. These questions are of immediate relevance to maintainers of EPL, even if they are not immediately answerable. And they have direct implications for what role EPL might play in enabling educational justice in CS more broadly, by centering student and teacher voices in shaping EPL designs.

While we cannot answer these questions here, but we can offer a prescriptive *bridge* between this north star and these concrete design questions. To this end, we answer *RQ2* by synthesizing seven justice-centered "requirements"[5] for EPL, grounded in design justice principles, prior work, and our own lived experiences as EPL designers and users. The requirements together constitute, and spell out, a framework we call *ALTCODE*, suggesting a minimum bar for achieving educational justice. For each, we define the property that must be met and give grounded rationale for it, and then offer examples of EPL that do and do not meet each requirement. To answer *RQ3*, we surface grand challenges for achieving each.

In presenting these seven requirements, we do not claim they are the *only* manifestation that qualifies as "justice-centered," or that it is even the best. Our focus was on language, race, ethnicity, culture, gender, ability, neurodiversity, class, and their interactions, but there are other facets to justice we do not consider here that may be relevant. We also do not claim that a single EPL could meet all of them or that doing so would be desirable. Third, our proposal is not a fixed target — conceptions of justice evolve, and are socially negotiated, and so this represents a moment in our discourse, not an endpoint. Our central claim is that these requirements are worthy of investigation in our discourse, as a research challenge, a design priority, and an evaluative framework. Finally, throughout, we conceptualize *community* abstractly. For some EPL, community might be global, raising the immense challenges of achieving global representation in youth and teacher voices. For other EPL, community might be hyper-local, focused on custom adaptations of EPL that serve a particular teacher, school, district, or region. Even these varying conceptions of "community", and the call to be explicit about what communities EPL are trying to serve, raise questions about the different design strategies required for these different scales.

We offer one final note about our process before we present the requirements. We begin with some notes about positionality, as our identities and experiences are fundamental to our synthesis of prior work into a design framework:

[5]We use the word "requirement" intentionally, borrowing terminology from software engineering to structure a *design space* for JCEPL to guide design and implementation as an alternative to "principles" or "guidelines", like those in Table 1, which can be viewed more as suggestions, rather than non-negotiable properties.

- The first author is a trans and queer, non-disabled, mixed-race person of color who brings rich lived experience with broader systems of oppression. She is also, however, an educational programming language designer: she made modest contributions to the *Alice 2.0* programming environment [72] as a graduate student, major contributions to the educational programming game *Gidget* [85], and led the community-engaged design and implementation of *Wordplay* [77], in partnership with teachers and youth. Jointly, this lifetime of experiences have informed her knowledge of the limitations and opportunities of EPL design. They have also shaped her beliefs on the necessity, but also near impossibility, of advancing educational justice solely through the narrow lens of CS and EPL design.
- The second author is a professor of computer science and learning sciences whose disabilities limit his mobility and sometimes necessitate use of different input devices. He has over twenty years of experience creating or contributing to programming and machine learning tools intended for educational use, including ALPACA ML [172], BlockyTalky [73], Co-ML [150], and NetLogo [1]. He conducts his research through community partnerships, and seeks ways to dissolve disciplinary boundaries while expanding people's agency to use computation for inquiry and creative expression.
- The third author is a graduate student with significant secondary teaching experience. Her experiences as a woman in computing inform her understanding of justice. Although less experienced with EPL design, she is very familiar with EPL use and pedagogy. She works to understand and disassemble underlying assumptions and norms in computing [36].
- The fourth author is a graduate student in computing education with years of experience teaching with EPLs, particularly with immigrants and refugees who often struggle culturally and linguistically when using EPLs. They have seen how EPL design can mediate what learning is possible and came to this work with curiosity about how to create inclusive EPLs.

Building upon these experiences, we used a collaborative approach to developing the proposed design framework. The paper began with the first author generating many possible requirements from prior work, and the other three authors critiquing and offering counter-proposals. We iterated on this until we converged as a group and felt the set captured by prior work, the four author's lived experiences with CS teaching and EPL design, and all felt the final set had coherence and completeness. We captured the final rationale for the requirements by citing the literature and principles surveyed in the prior section, building a bridge between ideas from prior work to the new ideas we are proposing. Our goal was to ensure that the set covered all requirements in Table 1, which are themselves a synthesis of prior perspectives on design and justice.

3.1 Requirement *Accessible*

> *Learners and teachers using an EPL must be able to use its full functionality with whatever input they can provide and whatever output they can perceive and comprehend.*

Prioritizing impact over intent is a central design justice principle Table 1.3; that means centering the impact of EPL design choices, rather than the intentions of EPL designers. One aspect of EPLs where this has been acutely overlooked is *accessibility*: most software and most of the web is not accessible to everyone. Baseline standards like WCAG 2.2 remain aspirational, even when they are used to enforce law [19], and do not even require *equitably pleasant* experiences [74]. EPL are part of this broader inaccessible world, but also help reproduce it.

This requirement, therefore, argues that all capabilities of an EPL, including the use of it and its program outputs, must be usable by all stakeholders, independent of what physical, perceptual, and learning abilities they have. This is not only central for students to be able to use EPL, but for teachers as well, and for teachers to navigate the diversity of access needs that can often constrain pedagogy and EPL choice.

It is important to note one nuance in the requirement. It is not saying that every EPL must be *universally* accessible without modification, or that such universality is even possible. After all, not every classroom has access needs that span the full diversity of human abilities. It is saying, however, that if there is a learner who wants to use, or is required to use an EPL, it should be possible to make the EPL accessible to them. Consider, for example, an EPL designed for rich computational imaging transformation. It *could* be made accessible to blind students, but it is not immediately clear how, or even whether that would be a compelling platform for blind youth to use. But if there was hypothetically a blind learner who wanted to use it, or was required to use it by a teacher in a classroom, it is the obligation of the designers to investigate and support that need, ideally in partnership with blind learners. This orientation toward accessibility is subtly different from a blanket universal accessibility requirement, in the same way that universal design for learning [65] is not about pedagogies that work for every ability without modification, but rather about prioritizing those modifications in partnership with learners.

One area of accessibility that is particularly important is in code editors and the constraints that PL designs impose on them. In the past decade, block-based editors have come to dominate EPL, and for pedagogically good reasons: by preventing syntax errors, they allow most learners to focus on semantics and expression [164]. But not everyone is sighted, can use a mouse, or use a keyboard; some rely on speech input, sip-and-puff input, or eye gaze. Efforts to create alternatives to text editors, including blocks, may inadvertently produce a segregated world where some learners live in different tools with only a subset of functionality found in more widely used editors. For instance, block-only editors can prohibit use (and thus learning) by people with low vision or dexterity. A design meant to include actually excludes when it is the only interface available.

Code editors, and editors for other aspects of programs (e.g., assets used in programs such as images, sounds, and other media), must be manipulable via *any* input modality and device. Anything less excludes students and teachers who might only read code via screen or braille reader, or who control a computer through low bandwidth eye or mouth input. This might also include tangible forms of programming, where programs take material form, rather than abstract form [55]. This requires creating editors and even physical computing media that support a diversity of input and output modalities, all with feature parity, and all without creating undue complexity from flexibility.

This requirement also applies to aspects of EPL use that are not typically supported by accessible technologies at all, such as what happens when stepping through a program's execution. This means avoiding program visualizations that require visual inference, and instead depicting (through whatever modality best serves a learner) multimodal symbolic forms to explain that a variable has a new value, that a conditional expression evaluated to false, or that a function is returning a value.

3.1.1 Analysis. The current ecosystem of EPL varies widely in this requirement. One EPL that does not meet it is *Scratch* [92]. Its exclusive reliance on mouse and touch-based drag and drop, while benefiting teachers and learners who *can* use a mouse or touch screen, excludes those who cannot, and forces teachers to settle for segregation or use other platforms. Its success at serving dominant groups has impacted many other platforms such as *Snap!* [58], and led to block interface building toolkits such as *Blockly* [46], that propagate an ableist interaction paradigm to countless other EPL. One sign of this inaccessibility are community advocacy efforts to either create accessible versions of code

editors outside of dominant platforms [9, 31, 102, 105], or to build coalitions to demand the Scratch team redesign the editor to be accessible to other inputs.

One EPL that has taken on the long-term project of building an accessible EPL from the ground up is *Quorum* [147]. Its focus has been screen readability; achieving this has required not only designing tools around a diversity of inconsistent operating systems accessibility frameworks to meet WCAG standards but also designing the syntax of the language itself around limitations of screen readers. For example, PL syntax often uses punctuation extensively, which most screen readers will not speak; instead, Quorum's grammar better positions syntax to facilitate screen reading. This early focus on blind learners has led to significant adoption in schools for the blind [148].

Of course, accessibility is not independent of other justice concerns. Quorum's focus on mirroring English, for example, privileges English. Quorum has also not yet engaged the wide spectrum of ability diversity. Still, the contrast between Scratch and Quorum's accessibility support is stark, and is especially striking given Scratch's two-decade history, including multiple complete rewrites (each of which could have prioritized accessibility, but did not) and substantial enduring funding. This is compared to Quorum's shorter history and smaller financial resources.

3.1.2 Challenges. Prior work has only begun to explore what communities of youth with disabilities even want or need to make with programmable media. Meeting this requirement is not merely about making existing EPL accessible, but potentially inventing new EPL for new media. For example, what would a gaze, sound and movement-based IDE for making purely gaze, sound, and movement-based apps be like? Early explorations with accessible programming of games for the blind [70] show that partnering with communities to co-design such experiences can reveal powerful new media that not only serves communities with disabilities, but also empowers everyone. Future work must examine ways that PL syntax, tools, and output can be represented in symbolic forms that can be transduced into learning-serving modalities by all access technologies.

Accessible EPL tools, of course, are still essential for any of this. One grand challenge is how to seamlessly integrate code editing interactions via speech and audio feedback. Techniques have been explored [11], but not with youth with sight or motor impairments. More recent advances in speech recognition and LLM-based program synthesis may enable new paradigms of interaction that might blend more seamlessly with text- and block-based editing paradigms. Other forms of input have received even less attention. For example, many people with motor impairments rely on low-bandwidth switches and eye gaze for input; many people with speech impairments rely on digital augmentative and alternative communication (AAC) devices; many people without sight use braille readers to read text. There is little research on how to make code readable or writable for any of these input and output devices, let alone pleasant and efficient.

3.2 Requirement *Liberatory*

> *EPL must empower learners with new conceptions of the natural, social, and artificial worlds, enabling them to imagine futures of computing that dismantle racial, patriarchal capitalism, and colonialism.*

Design justice principles call for the healing and empowerment of communities (Table 1.1) and designs that reconnect communities rather than exploit them (Table 1.9). Most EPL, do not reshape learners' conceptions of themselves or the world toward justice. Some EPL are platforms for creativity in which such learning might occur. Some are simply new media, marketed to educators as a potential tool for learning, but that primarily entertain and engage. Others still are instructive, but do not necessarily change a learner's conception of themselves or what futures might be possible in the world. Moreover, many aspects of PL remain so deeply linked to the original rationale for computers –

efficiency, productivity, profit, and domination – that most of these elements are "built-in" and viewed as "foundational" in computing. So many of the ideas inherent to computing culture are deeply linked to systems of capitalism that exploit women and racial minorities for profit [101] that computing cannot be understood as a social phenomenon without understanding its role in upholding these exploitative systems.

We conceptualize *liberatory* as EPL that explicitly offer youth the opportunity to deeply question the purpose of computing and computing skills as they learn them. Such questioning might enable learners to navigate power structures as they are, but also harness those skills to create a more just future of computing and society. This conception argues for a particular kind of learning that evokes hook's arguments [63] about teaching, which focus learning on how students change, particularly in their conceptions of themselves in relation to the world. It also relies on Papert's arguments [117] about powerful ideas that center learner agency in constructing knowledge. Liberation suggests that EPL should be media in which concepts of data, computation, and algorithms are learned, but also critically examined, deployed — and resisted — for broader projects of justice. This calls back to design justice ideas of sharing knowledge with communities, enabling youth to be critical designers themselves [25], to create their own futures. We contrast this requirement with the similar ways that teachers might use liberatory pedagogy, or that curriculum content might be liberatory in nature: EPL themselves, as media, have a role in enabling liberatory thinking about CS, to supplement these other liberatory pedagogical moves.

Some EPL designers might view this requirement as a politicization of inherently apolitical artifacts, especially in light of legislative bans on educators stating that racial, gender, ability, and class oppression exists in the United States. This position is, of course, a dismissal of rigorous examinations of the politics of technology (e.g., [13, 15, 34, 167]). These works make plain that behind every technical idea, PL included, are unexamined assumptions and priorities [113] that privilege social groups with more power over those with less. We leave it to skeptics to demonstrate that design choices that create inequality, intentional or not, are apolitical.

3.2.1 Analysis. It is easy to point to EPL that do not meet this requirement. Consider, for example, *CodeCombat*[6], a for-profit platform that describes its goal as giving learners "*the feeling of wizardly power at their fingertips by using typed code*." Rather than emphasizing how youth change, it emphasizes how much youth produce (e.g., "*1 billion lines of code*"). It emphasizes not how it empowers teachers to change how youth see themselves in the world, but how it can offer a "turnkey" solution for educators that centers students' desires to create games and have fun, freeing them from having to teach, or understand computing themselves. The focus on engagement, while not inherently problematic, masks its lack of focus on students' identities and their deep understanding of the tradeoffs of computing as a medium for expression. And, of course, it is frames learning around ideas of war and domination.

We do not know of an EPL that is liberatory in the way we have conceptualized it here. PL in general, and some EPL might be applied towards liberatory ends — using Python, for example, to do data science that surfaces racist decisions in a public data set — but that does not necessarily make Python itself justice-centered. We can imagine, however, what EPL *might* do. One glimpse, for example, is the narrative framing of *Gidget*[7], a debugging puzzle game, is that computers are inherently incapable of understanding intent or solving problems and that only people can, so they must be in charge of how computing is used [85]. This framing device is deployed to place boundaries around what the central characters' capabilities are and position the learner, as a human being, as the only one capable of identifying what is wrong with a problem and what to do about it.

[6]https://codecombat.com
[7]https://helpgidget.org

Gidget's narrative, while far from liberatory, suggests that future JCEPL might frame PL in liberatory rhetoric. Imagine, for example, a "self-critical" EPL that uses program analysis to provide critical judgments of a program's limitations on the data it is processing or the phenomena it is automating. Such rhetoric is visible in the documentation of *Wordplay* [77], for example, where each language construct is personified and given voice, as a character in a community. Each character, in articulating their functionality and purpose, surfaces existential questions about computing, such as whether representing truth as binary is sufficient to model the world, or how conditional logic could possibly handle all of the exceptions in complex decisions.

3.2.2 Challenges. Some liberatory challenges are technical and reach deep into computing foundations. For example, most EPL build upon Boolean logic and its limitations in representing truth and decision-making. How might EPL be built on different frameworks of reason (e.g., delegating high-stakes conditional branches in code back to communities, for critical, collective examination)? Engaging youth in epistemic playfulness, especially through the medium of computing, and in the ways it is applied in the world to structure rights and access to resources, is a broadly unexamined space.

Another challenge is inventing ways of weaving critical pedagogy into EPL designs, while still promoting mastery. We know computing can be taught in ways that foster self-efficacy and growth mindsets [88], but are there ways to design EPL to help youth critically examine the world through a computational lens, while reinforcing this growth, or might challenging the purpose of computing subvert youth motivation to learn? What new pedagogies might position learners as "philosophers of technology" [156], rather than vessels for production, while still valuing engineering skills? What new kinds of integrations between EPL, assessment, and pedagogy might be necessary to realize this vision? Our community has only recently begun to examine how to weave counter-hegemonic narratives into computing education (e.g., [30, 35]), and doing so, even for skilled facilitators is hard, as youth are so often told that they do not have or deserve power. How might EPL contribute to such pedagogies, and do so in ways that enable youth, families, and teachers to fluidly engage with the narratives it presents?

Other challenges of liberation are political. In a time where misrepresentations of "critical race theory" or "woke" have turned into legislative bans on free speech, how might EPL reframe liberatory discourse about computing, or resist these bans through youth-empowered governance?

3.3 Requirement *Transparent*

> *To foster youth agency over program behavior, program meaning must be learnable and program behavior must be observable, at multiple levels of granularity.*

Design justice principles call for sharing design knowledge with communities (Table 1.7) and viewing design as a partnership with ongoing accountability (Table 1.4). These principles are only possible in an EPL if learners and teachers are able to understand how EPLs work, and that requires transparency.

This is not the status quo. In most PL, programs do a large number of things very quickly and typically without explaining themselves. This is both the great power of computer programs, but also a great source of confusion, and a significant source of low self-efficacy and sense of agency in CS learning [10, 43]. In fact, most EPLs are quite opaque about their behavior, hiding explanations in implementations, and offering limited visibility of program execution. Not only does the lack of transparency complicate student learning, but it complicates teaching [59, 96], introducing comprehension friction, adding uncertainty in teachers' debugging pedagogies, and even deterring teaching CS itself.

There are many things that can help learners understand program behavior, including teachers and community. For example, pedagogical scaffolding of state changes can help [26, 79, 109, 168]. JCEPL, however, might have particular

supports built-in. For example, transparency about behavior might include clear documentation about every language feature, and links to that documentation from code. It might include visibility of runtime state during program execution, and more generally, the ability to observe program execution to track control and data flow. For example, prior work suggestions that that precise, causal, reversible explanations of how programs execute are central to learning a PL's evaluation rules [28]. Therefore, transparency might even include being able to reverse time to inspect a conditional branch's decision, or demonstrate some input, replay it, and even change an input and replay from that changed point. EPL might allow learners to request explanations of why a program did or did not do something (à la, [79]). Learners should be able to slow down time, to better manage their attention, make their tools align better with their motor-physical abilities, or get time to reason about causality. These capabilities, while seemingly distant from justice, are particularly central to comprehension, self-efficacy, and agency over computing.

3.3.1 Analysis. Transparency of behavior in EPLs is still poor. Consider *Snap!*, for example. It provides the barest of controls, including a limited "pause all" statement that pauses execution like a breakpoint, a "say" block that acts like a print statement during program execution, and a slider for controlling the speed of execution. This meets some of the goals of this requirement, but the tools are quite low-level, do not make clear the correspondence between code, state, and output, and do not offer reversibility. And, of course, since *Snap!* only works for sighted learners who can use a mouse, making program execution invisible to learners without sight.

While no EPL comprehensively meet this requirement, some do better. *Racket* and the *Dr. Racket* [44] tool, for example, offers basic stepping functionality, which allows learners to step forwards and backwards through an expression's evaluation, using a "rewriting" metaphor, which shows the result of each function application. It also can generate images depicting graphs of function calls over time. Programs can be stopped and restarted at any time. The result of executing a program is deterministic and so each replay is a chance to build comprehension. This approach orients learners towards building knowledge over how programs work, fostering agency. Other platforms, such as *Wordplay*, [77], offer random-access, instant time travel debugging to any point in a program's history of execution. Such features offer learners and teachers opportunities to inspect program behavior at a low level, though many open questions remain about the efficacy of such supports in fostering student and teacher agency over code.

3.3.2 Challenges. Instant, random access to any part of a program's execution should be a grand challenge of EPL design and implementation. Prior work has examined some of these [77, 79], exploring interfaces for interrogating program output and simulating reversibility for simple procedural languages. But how might EPL be architected to allow for fine-grained control over, and insight into, temporality, especially for multi-threaded programs, real-time applications where there may be a large volume of runtime state, or in distributed settings, where understanding causality depends on also understanding network topologies and is intrinsically non-deterministic (e.g. [73])? How might EPL semantics and programming interfaces need to change to make such inspection possible? Justice means wrestling with the possible tensions between how quickly something is computed and how we are able to understand *how* it is computed, whether for learning, program comprehension, or debugging. In education, in particular, we might trade performance for comprehensibility, particularly in an era where LLM-generated code has put an even greater distance between people and code [103].

Other challenges are pedagogical. If the point of transparency is agency, how can comprehension features help youth examine the limitations of computing? How might transparency features be used for advocacy, such as algorithmic audits of surveillance software in schools and in public? All of these possibilities rely on EPL that are explicitly designed

to be probed, disassembled, and critiqued from the outside in and inside out, and that requires seeing not only code, but execution.

3.4 Requirement *Cultural*

EPL must be culturally relevant, responsive, and sustaining in how they are designed, explained, and framed, enabling identity-inclusive pedagogy.

Design justice principles call for design processes to center direct stakeholder voices (Table 1) and value stakeholders' lived experiences (Table 1). This means, among other things, responding to and sustaining culture. As Scott et al. delineate [137], it means embracing the notion that all youth can innovate with computing, that learning should be transformative to youth and their communities, that learning fosters sociocultural understanding and insight about youths' intersectional selves, and that success should be framed around who creates and to what end. It means co-constructing anti-racist learning experiences [35] and assessments [64] by sharing power with youth. These tenets reposition computing education as about learners, their values, communities, and identities, rather than about a prescribed curriculum, a particular technology platform, or a set of industrial priorities. Whereas the *liberatory* requirement looks to the broader ways that computing is situated in society and how to change them, *cultural* looks to the specific cultural worlds in which youth live and EPL's relationship to them.

For EPL, this means examining the "unmarked" cultural ideas embedded in their designs [45]. Some of these ideas are conceptual. The notion of a "queue" data structure, for example, common in British English, and embedded in British culture, is not a universal one. Boole's binary truth values, by his own admission, reduce complex social ideas such as gender into fixed categories and can erase culture and identity [16]. These cultural ideas may serve some learners more than others, either by being more or less legible, or by signaling inclusion and exclusion.

But being culturally sustaining also means resisting linguistic hegemony [33]. Humanity has thousands of languages and is broadly multilingual, and yet nearly all EPL are English-centric, or monolingual. And language is more than linguistics; it is also *languaging*, the activity of speaking, hearing, listening, writing, reading, and communicating, which may or may not involve written scripts [90]. Justice here means supporting *all* of the world's languages and languaging, and allowing every person to communicate about code in the languages and languaging they use in their everyday life with families, peers, and communities. This affords the many benefits of translanguaging in language aware asset-based pedagogies [159], but also opens new possibilities for educating youth about how to create multilingual programs. It may also mean embracing threatened languages, and even creating new languages for colonized ideas in mathematics and computing, perhaps even connecting EPL to projects of cultural revitalization and healing [68]. Numerous recent works have begun to explore these possibilities, finding, for example, that students are far more comfortable learning CS in their first languages [152], and that they engage more advanced PL concepts faster [27].

3.4.1 Analysis. Most EPL are far from meeting this requirement. Consider *Python*, frequently used in introductory programming courses. Its syntax is English-only; Python 2 had weak Unicode support, privileging Latin characters in ASCII, and took years to embrace Unicode support. Its website is only in English; its annual gathering, PyCon, is only in the U.S. Although it has community-contributed localized documentation, there are only a handful of languages supported. And deeply embedded throughout the language and its libraries are culturally-bound metaphors (e.g., "pickle", "nanny", "abc"). And many of the community's "Zen of Python" mantras are questionable in relation to diversity (e.g., *"There should be one– and preferably only one –obvious way to do it."*, *"Special cases aren't special enough to break*

the rules."). None of this is to say that Python is *bad*: the Python Standards Foundation's governance is impressively global. But it is a PL deeply rooted in Western culture.

In contrast, some platforms like Blockly have put extensive effort into localization, and many other-block based platforms support rich visual and auditory media that enable youth to express their cultural identities (albeit in ways that are often inaccessible). One particularly extreme example of this linguistic responsiveness is *Hedy*, a multilingual EPL that has grammars for 47 of the world's languages [60]. It is unique in its explicit focus on embracing variation in language and culture through adaptable PL grammars, and extending that localization to the instruction and examples it provides as scaffolding. Recent work on *Wordplay* [77] advances multilingual programming further, supporting multilingual strings, documentation, and identifier names, machine translation of programs into other natural languages, and localized program output. These two projects point to possible EPL futures where language diversity is at the center of computing education, and another resource for learning and teaching.

3.4.2 Challenges. Future work should examine EPL that radically sustain culture. For example, radical might mean examining EPL that privilege no particular natural language, but embrace all natural languages, being designed for our multilingual, global world. They might enable programs to be viewed in any natural language, or even multiple natural languages, to allow for multilingual classrooms to fluidly collaborate across languages. How might PL syntax, code editors, and documentation be designed to allow for such fluid engagement with the world's languages? The same technical capabilities that would be required to support accessibility (see Section 3.1), may go a long way toward making this possible as well (though work on *Wordplay* [77] has found that they are often in tension, by introducing complexity).

Another challenge is how to communicate meaning about the purpose and semantics of computation, given the inherent reliance of PL on culturally-situated metaphors. How might, for example, a data structure like a queue, be described with a multiplicity of metaphors from across cultures, or even enable communities to devise their own metaphors to explain computational ideas? When such cultural touchstones are unknown or missing (not all cultures include the same ideas), what alternative representations will work?

Finally, could youth create their own PL, with their own ideas about computation, their own explanations for those ideas, and in ways fully situated in their cultures, communities, and values? Rather than perseverating over what notional machines to use to best explain computing to students [41], our machines could operate based on students' notions of how they should work [116]. Such futures would require a reexamination of the entire toolchain of PL design and implementation to support novice youth PL designers to collectively envision programmable media of their own.

3.5 Requirement *Obtainable*

Learners must be able to access an EPL and its tools and resources independent of their financial means.

Design justice emphasizes understanding a community's existing solutions before building new ones (Table 1.10). Part of achieving that is understanding what resources a community has access to, how they are using them, and what a new design might mean in light of current and future resources. This is a particularly central consideration in teaching in schools, where funding for learning technologies and their indirect technology requirements, may be insufficient or fickle [126], further exacerbating inequities.

One major aspect of this in computing education is understanding the hardware and infrastructure requirements of an EPL. This consideration, along with the broader arguments around educational justice presented earlier, suggest a

requirement that PL must be free, not imposing any monetary cost on learners or teachers to access, including access to the Internet or a device on which to use an EPL.

Obviously, this is exceptionally difficult to achieve, and not necessarily reasonable to achieve for all EPL. But in terms of justice, the rationale is clear: educational justice, broadly construed, requires educational equity [162], and not all students have the money to own their own devices, or even access devices for extended periods of time. Even low-cost hardware (e.g. the $15 micro:bit), can be too expensive for some settings without subsidy [143]. In many low-income homes, a family might time-share a smartphone, meaning that access to the app would be limited, as might storage on the device to download the app and create content. In many low-resource schools, students might share one computer across an entire classroom or school, severely limiting time on devices. Some learners and families might not have a smartphone at all, and instead rely on computers at school and public libraries; in these cases, the application would not be obtainable at all to youth or their families. These situations are not rare: 1 in 7 globally do not have reliable access to electricity; most country's smartphone penetration rates range from 30% to 70%, with Japan and the U.S. at around 80%. This is a broader reality of global digital divides [158] with which EPL must contend.

This requirement sets an operating context standard for which EPL must be designed. For example, EPL should work on slower, public devices that may be shared resources, and where Internet connections may be unreliable or slow, acknowledging the ongoing global digital divide. This requirement might be achieved by bridging digital divides, or by designing platforms that have minimal performance requirements and maximal compatibility. It might be achieved by designing for public institutions that focus on information access, such as public libraries and schools.

3.5.1 Analysis. Any EPL that assumes possession or ownership of a device cannot meet this requirement. Consider, for example, the recently released *OctoStudio*[8]. It is free, and does not require the internet after it is downloaded, increasing obtainability. But using the app requires possession of an Android 8 compatible device or an iOS 15 compatible device, the ability to install applications on it, and time to use the device. OctoStudio is, unavoidably, an EPL for economically privileged youth.

In the absence of universal rights to computing and the internet, the current floor to be designed for is one that imposes no device ownership requirements, no device installation requirements, no sustained access to the Internet, and the ability to save work on local, non-cloud devices, or at least privately on shared devices. While there are many examples of IDEs that are free to use online and offline, device access and cost is a central challenge. One example that partially overcomes this is the Texas Instruments line of graphing calculators: they are low-cost, portable, battery-powered, require no Internet access, allow for the creation of a diversity of programs in TI-BASIC, including inputs from sensors, output to speakers, LEDs, and more. And crucially, they are partially or fully subsidized by schools. Of course, there are numerous caveats to the platform in relation to other JECPL requirements. For example, paradoxically, TI's calculators sell at massive profit margins, and so exploit the same educational systems they assist. This is due to a combination of near-monopoly (schools and testing companies sometimes prohibit using other devices), infrequent technical improvement, and massive economies of scale [29].

3.5.2 Challenges. Future work might pursue open hardware platforms and communities to work together to create tools that have the beneficial affordances of TI's calculators, but with less rent-seeking and more responsiveness to teachers' and learners' desires. Some of the challenges that surface in realizing this vision include: 1) how to finance and sustain such an ecosystem of hardware and software, 2) how to design its governance structures to center communities,

[8] https://octostudio.com/

3) how to reconcile the diverse needs of communities with the world's limited capacity for a multiplicity of such platforms and the complexity of bringing multiple platforms into learning spaces.

3.6 Requirement *Democratic*

> *EPLs must be governed by and accountable to learners and their communities of support, especially those marginalized in computing and society more broadly.*

Design justice calls for designers to play the role of facilitators, not as leaders, instead distributing power to communities (Table 1.5). In computing education, this suggests a requirement that learners and their supporting communities should be in control of EPL governance, not EPL designers. This might mirror the calls in culturally relevant, responsive, and sustaining pedagogy frameworks [49, 83, 118], anti-racist assessment frameworks [64], and liberatory pedagogy frameworks [47, 63], to put learners in charge of shaping curricula, assessments, and classroom norms.

In this requirement, one form of accountability is openness. We define "open" as source code being available, there being a community process by which anyone can submit change requests to the design or its governance processes, and there existing a process by which students and teachers can gain power, particularly if they are not being served by an EPL's design. This implies some degree of power sharing, but does not imply majority rule, as that would inherently be counter to equity. Doing this in alignment with design justice principles means openness that is collaborative, facilitating, asset-based, outcome-focused, and non-exploitative [25].

There are many strategies for power sharing. One is governance, finding processes and policies for compromise on design choices. Other strategies may be designed into the platforms themselves to transfer power to individuals and groups. For example, EPLs might be designed to be *extensible*, enabling learners, teachers, and communities that support them, to change, customize, extend, or personalize a platform, without requiring agreement or support from others.

Whatever the strategy, it means that EPL designers will have to give up power. This raises challenging questions about whom should be given that power: having youth and teachers at wealthy private schools with comprehensive access to computing education in schools lead the design of EPL would lead to very different designs than engaging youth at low-resource public schools with no access to computing education. This requirement prioritizes the latter, centering voices on the margins of computing in order to ensure that the programmable media we create for learning serves everyone, no matter how powerless they are. Openness alone is therefore inadequate; it must be an openness that courageously partners with communities who may have the fewest resources with which to partner, and finding ways to sustain those partnerships.

3.6.1 Analysis. Many EPL are open, but not democratic. One worth highlighting here is Code.org and it's *Code Studio* platform. The platform is open source, and offers contributor guidelines. Teachers also provide feedback to Code.org, during professional development and as part of teaching. Code.org also organizes advisory boards to help inform its curriculum revisions and has a long history of offering free professional development and engaging educators in offering that professional development to each other. These efforts all move towards the democratic vision delineated above. But the key element of the requirement — that EPL be community-led and centered on the margins — is more questionable. Design authority resides in Code.org's designers and engineers, and not youth or teachers excluded from computing education.

A contrasting case is *Processing*[9], and the *Processing Foundation* that governs its work. It is also open source, but has numerous community contributions and many active pull requests. The foundation runs public events that solicit advocacy; it funds fellowships for teachers to explore and shape the platform; it partners with advocacy organizations at the margins of computing; it mentors and supports new contributors to the platform; and it directly engages communities and community leaders to shape priorities. While these activities may appear similar to Code.org's, they are different in how power is distributed, and thus more aligned with the democratic principles.

3.6.2 Challenges. This requirement raises many questions about how to sustain democratic governance. Fostering communities takes time and resources, and so future work might examine how to structure partnerships and funding, how to leverage emerging insights about student, teacher, and family advisory councils [7], and how to organize product management strategies that are community-led, rather than designer-led. Partnership work often involves conflict — individuals within communities, and distinct communities, often disagree about what an EPL is for and where it should go — and so there are many questions about how to manage and resolve that conflict, while accounting for the many technical constraints that software and EPL implementations impose.

Another challenge is technical. PL implementations tend to be deeply entangled across many layers of language design, compilers, toolchains, IDEs, and documentation. Even a small challenge at the bottom of this stack can have profound consequences for everything built atop it, including all of the programs ever written in the language. There remain open questions about how to enable EPL to be *redesigned* in response to change, and how to manage all of the work that those redesigns might impose on communities who have invested in prior designs.

3.7 Requirement *Enduring*

> *EPL must be sustainable for as long as a community needs them to be, respecting a community's capacity for change and planet's capacity for computation.*

Design justice calls for sustainable outcomes (Table 1.8). In computing education, means recognizing that EPL are critical infrastructure for learning and that they must be sustainable in order to sustain learning, teaching, and teacher education that depends on them.

Many things are required to sustain EPL, including funding to pay for maintenance, but also a community's capacity to maintain an EPL implementation. Software maintenance and funding closely interact: building and maintaining EPLs takes time and skill, and in capitalist systems, both cost money. JCEPL should attend carefully to where such money comes from, what constraints are applied to it, and how it is used to equitably subsidize maintenance labor. All of these factors are also impacted by how EPLs are implemented: software architectures vary in maintainability, and JCEPLs may require architectures and evolution practices that are more resilient to contributions from community members with widely varying skill. Research on online communities can inform challenges to community-led maintenance of computing infrastructure [40], including the unique burdens placed on experts that create single points of burnout and abandonment.

Enduring does not necessarily mean that an EPL lasts forever. EPL projects being canceled, or becoming incompatible or unsupported, can be a threat to investments in curriculum, teacher knowledge, and youth identity, and so sometimes they should last longer than they do. However, EPL lasting indefinitely is not necessarily in service of justice either: older EPL, which actively complicate learning or subvert justice, can do great harm when they endure. Newer EPL, which rely on unsustainable energy consumption (e.g., to fuel generative AI [14, 161]), or on material resource mining,

[9] processingfoundation.org

should not necessarily endure, especially when they physically harm the communities they serve [94, 142]. The value of endurance, in justice-centered framings, is something that a community should judge itself, sustaining a project for as long as it serves community values and goals, but perhaps for no longer. Youth and their teachers may also need to learn about sustainable computing, in local and global terms, in order to make these judgements. And teachers may need to interrogate the power they hold over students and incentives they have to keep using an EPL, even when it is doing harm.

3.7.1 Analysis. Whereas other requirements have relatively clear examples of success and failure, scholarship on sustainability of this kind is nascent enough that it is hard to critique systems. Instead, we offer contrasting cases that make challenges salient.

Consider Apple's *Swift Playgrounds*, a puzzle-based learning environment for learning Swift. Released in 2016 for iPad, it has a large base of curriculum and pedagogical support from Apple, one of the most valuable companies in the world. There is, however, no statement on how long Apple will support the platform, and no visibility into how product evolution decisions are made. Teachers must decide whether to invest in the platform that could disappear unexpectedly. And they have no voice in whether it does or does not; that is up to Apple's executives, their board, and their stakeholders.

Scratch, in contrast, is notable for its large base of funding, now centralized in the Scratch Foundation, and a history of more than 20 years of support, including multiple re-implementations. Despite this endurance, however, its limited openness means that community's capacity to maintain the platform may be limited if the foundation were to stop supporting the project. And endurance alone has not guaranteed equity: each time Scratch was re-implemented was an opportunity to act upon crucial feedback (e.g., about accessibility) that the team chose to deprioritize.

Finally, consider a research-based platform like *LaPlaya* [62], an EPL for primary computing education that sought significantly lowered "floors" and age-appropriate content. While there is a strong base of evidence around the platform from 2015, it was only available for less than a decade, likely due to lack of funding and human resources for maintenance. This is not a critique of its creators, but rather a sign of the deep complexities of sustainability in a world that broadly devalues educational justice.

3.7.2 Challenges. These cases demonstrate the core tensions around funding, community resources such as skills, time, and money, and indirect impacts of EPL on the environment and public health that can indirectly harm learners. On one side, EPL endurance requires funding and labor, and so research must examine justice-centered models for sustaining these resources both technically, socially, and politically, to promote resilience. On the other side, EPL endurance when learners or teachers no longer want them can become a barrier to change: future work should examine when and how to purposefully retire EPL that are doing more harm than good. This will be particularly important as EPL increasingly contribute to global energy consumption by embracing generative AI. And these sustainability risks, in turn, contribute to unpredictable impacts on when, how, whether and who computes.

4 Discussion

Our core argument is simple: EPL play an instrumental role in structuring what kinds of computing education are possible, who education serves, what kinds of digital worlds are possible, and whether those worlds are just. Being justice-centered means redistributing the power to design EPL to learners' and their communities, to more intentionally center and support their needs, values, cultures, and abilities (*RQ1*). The *ALTCODE* requirements (*RQ2*), as interpretations

of design justice principles for EPLs, are a proposal for conceptualizing the role of EPL in educational justice, and they raise many technical, social, and political grand challenges for future work (*RQ3*).

The implications of ALTCODE requirements, however, are complex. For example, we might argue that not meeting these requirements at the EPL ecosystem level is *hegemonic*. After all, anything less privileges dominant groups, whether a PL expert, a particular group of learners, or particular learning contexts. Such differences in privilege exist outside education, but being justice-centered in EPL design, at least in Rawlsian terms, means accounting for difference in how we distribute power. If we do not, we risk designing EPL that actively uphold and perpetuate injustices. And many EPL — all, perhaps — are complicit in these risks, even if unintentional. [10]

Another complexity is the implied universality and non-negotiability in the requirements. For example, it is tempting to imagine a single EPL that might be designed to meet all requirements, in the same way that a superficial interpretation of *Universal Design for Learning* [66, 132] might seek a single pedagogy that works for all students. The ALTCODE requirements do not require or imply that such an EPL should exist or is even possible to exist. Moreover, they do not imply that having only one EPL that meets these requirements would be desirable. After all, being culturally relevant and sustaining [83, 118] likely demands the opposite: that there exist *many* EPL that strive to meet all seven requirements as robustly as possible, to respond to different interests, needs, and contexts. A better conception of the requirements is as EPL design properties that ideally might all be met, but may also be in tension with each other. Future work should examine where these tensions lie, whether they can be overcome, and deepening our understanding of how feasible they are to meet in combination.

Our community has begun to make progress on this work. Many platforms have invested in localization, which has changed the capacity of teachers to engage learners not fluent in English. Many EPL are open source, though they vary in how power is organized and governed. Many EPL have explored accessibility for some kinds of disability. Most EPL have an intention of being enduring in some way, even if we have yet to question how or whether they endure. Recent works like Hadwin et al.'s deconstruction of tensions between abstract programmable media, tangible forms of computing, and our elevation of abstract, symbolic forms of computation, point to the kinds of analysis between the EPL we have and the EPL we might need to engage everyone. The computing education community is not necessarily "failing" to be justice-*centered* – every project of justice is a slow march [110] – but we are not done and there is much work to do.

Part of progress is examining whether existing EPL can be adapted to meet ALTCODE requirements. These are partly technical questions – *transparency*, for example, has significant implications for EPL runtimes that are no simple matter of engineering. These are partly interaction design questions — can code editors seamlessly support keyboards, mice, speech, gaze, and other forms of input, while being legible via sight, sound, and touch? Many of these are questions of governance — can control over PL design be distributed when projects have so long centralized power among a few leaders? And many of these are questions about the broader sustainability of the planet, and of the platforms we create — should we embrace LLM-based generative AI in EPL, with its massive demands for energy consumption? All of these questions are in direct tension with power, resources, culture, and values. It is not obvious whether changing existing EPL or creating new ones is easier; both likely pose their own challenges, as we have learned from the broader history of activism, which regularly must choose between changing existing systems [114] or creating new ones [111].

All of this will require research. Scholarly venues have historically been hesitant to publish analytical critiques of EPL, favoring post-positivist evaluations of learning outcomes. For example, this paper itself was originally rejected

[10]The word "hegemonic" poses challenges, as many EPL designers may not want to accept complicity. The identity work necessary for accepting it may be an important resource for progress [163].

from ACM's *International Computing Education Research* conference for not being "reproducible" and being a "position paper," reinforcing the epistemic barriers to advancing novel arguments about justice in computing education. We hope that *ALTCODE* can provide one basis for rigorous analyses of EPL features and tools, as an alternative to empirical studies. We also need venues where EPL analyses and designs are welcomed and fairly evaluated within their epistemic goals. Some venues have signaled such interest [140].

We end by recognizing that our arguments are cold comfort. Every day, learners encounter EPL they cannot use, teachers choose between lesser evils, and computing education, for all of its profound possibilities, gets reduced to skill development in service of corporate profit. A primary school teacher we talked to recently, for example, expressed her frustration that while most of her students diligently used Scratch, Jr. on laptops, one of her students, with a sensory aversion to touching screens, could not engage. "*Is there an alternative, like speech, or another platform that doesn't require a mouse or keyboard?*"; we shared that there was not. If there is anything we hope our argument offers to learners and teachers today, it is the idea that you deserve better, that better is possible, and that we will build better, together. We hope this argument is a catalyst for deeper discourse on how.

Acknowledgments

This work was supported in part by the University of Washington Center for Research and Education on Accessible Technology and Experiences (CREATE), the Paul G. Allen School of Computer Science & Engineering, the Population Health Initiative, and the Race & Equity Initiative at the University of Washington.

References

[1] Dor Abrahamson, Matthew W Berland, R Benjamin Shapiro, Joshua W. Unterman, and Uriel J. Wilensky. 2006. Collaborative interpretive argumentation as a phenomenological-mathematical negotiation: A case of statistical analysis of a computer simulation of complex probability. *For the Learning of Mathematics* 26, 3 (2006).

[2] Suad Alaofi and Seán Russell. 2022. A validated computer terminology test for predicting non-native English-speaking CS1 students' academic performance. In *Australasian Computing Education Conference (ACE)*. 133–142. doi:10.1145/3511861.3511876

[3] Hend Alrasheed, Amjad Alnashwan, and Ruwayda Alshowiman. 2021. Impact of English proficiency on academic performance of software engineering students. In *International Conference on Data Storage and Data Engineering*. 107–111. doi:10.1145/3456146.3456163

[4] Jean Anyon. 2006. Social class, school knowledge, and the hidden curriculum: Retheorizing reproduction. In *Ideology, Curriculum, and The New Sociology of Education*. Routledge, 37–45. doi:10.4324/9780203112878

[5] Ian Arawjo and Ariam Mogos. 2021. Intercultural computing education: Toward justice across difference. *ACM Transactions on Computing Education (TOCE)* 21, 4 (2021), 1–33. doi:10.1145/3458037

[6] William Aspray. 2016. *The broadening participation in computing alliances*. Springer International Publishing, 53–102. doi:10.1007/978-3-319-24832-5_3

[7] Ellen Bacon and Usa Bloom. 2000. Listening to student voices: How student advisory boards can help. *Teaching Exceptional Children* 32, 6 (2000), 38–43. doi:10.1177/004005990003200605

[8] Catherine M. Baker, Cynthia L. Bennett, and Richard E. Ladner. 2019. Educational experiences of blind programmers. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. ACM, 759–765. doi:10.1145/3287324.3287410

[9] Catherine M Baker, Lauren R Milne, and Richard E Ladner. 2015. Structjumper: A tool to help blind programmers navigate and understand the structure of code. In *ACM Conference on Human Factors in Computing Systems (CHI)*. 3043–3052. doi:10.1145/2702123.2702589

[10] Laura Beckwith, Margaret Burnett, Susan Wiedenbeck, Curtis Cook, Shraddha Sorte, and Michelle Hastings. 2005. Effectiveness of end-user debugging software features: Are there gender issues?. In *ACM SIGCHI Conference on Human Factors in Computing Systems*. 869–878. doi:10.1145/1054972.1055094

[11] Andrew Begel and Susan L. Graham. 2006. An assessment of a speech-based programming environment. In *IEEE Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 116–120. doi:10.1109/VLHCC.2006.9

[12] Andrew Begel and Amy J. Ko. 2019. *Cambridge handbook on computing education research*. Taylor & Francis, Chapter Learning outside the classroom.

[13] Ruha Benjamin. 2023. *Race after technology: Abolitionist tools for the New Jim Code*. Routledge. 405–415 pages.

[14] Adrien Berthelot, Eddy Caron, Mathilde Jay, and Laurent Lefèvre. 2023. Estimating the environmental impact of Generative-AI services using an LCA-based methodology. *Procedia CIRP* (2023). doi:10.1016/j.procir.2024.01.098

[15] Rena Bivens. 2017. The gender binary will not be deprogrammed: Ten years of coding gender on Facebook. *New Media & Society* 19, 6 (2017), 880–898. doi:10.1177/1461444815621527

[16] George Boole. 1854. *An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities.* Walton and Maberly.

[17] Margaret Burnett, Scott D Fleming, Shamsi Iqbal, Gina Venolia, Vidya Rajaram, Umer Farooq, Valentina Grigoreanu, and Mary Czerwinski. 2010. Gender differences and programming environments: across programming populations. In *ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–10. doi:10.1145/1852786.1852824

[18] Angela M Calabrese Barton, Kathleen Schenkel, and Edna Tan. 2021. The ingenuity of everyday practice: A framework for justice-centered identity work in engineering in the middle grades. *Journal of Pre-College Engineering Education Research (J-PEER)* 11, 1 (2021), 6. doi:10.7771/2157-9288.1278

[19] Toni Cannady. 2019. Classifying WCAG 2.0 guidelines as the legal standard for website under title III of the Americans with Disabilities Act. *Cath. UL Rev.* 68 (2019), 209.

[20] Ruijia Cheng, Aayushi Dangol, Frances Marie Tabio Ello, Lingyu Wang, and Sayamindu Dasgupta. 2023. Concepts, practices, and perspectives for developing computational data literacy: Insights from workshops with a new data programming system. In *ACM Interaction Design and Children Conference*. ACM, 100–111. doi:10.1145/3585088.3589364

[21] Michael Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A. Myers. 2021. PLIERS: A process that integrates user-centered methods into programming language design. *ACM Transactions on Computer-Human Interaction (TOCHI)* 28, 4 (2021), 1–53. doi:10.1145/3452379

[22] Cynthia E Coburn, Amy K Catterson, Jenni Higgs, Katie Mertz, and Richard Morel. 2013. *Spread and scale in the digital age: A memo to the John D. and Catherine T. MacArthur Foundation.* Technical Report. MacArthur Foundation. https://informalscience.org/wp-content/uploads/2019/02/spread_and_scale_in_a_digital_age_12-31-13_to_share.pdf

[23] Code.org. 2025. *2025 State of AI & Computer Science Education.* Technical Report. Code.org. https://drive.google.com/file/d/1p1x-UFVec0fwXOBv2B17j65wiD8KYSvQ/view

[24] Patricia Hill Collins. 1998. The social construction of Black feminist thought. In *Feminist foundations: Toward transforming sociology*. SAGE, 371–396.

[25] Sasha Costanza-Chock. 2020. *Design justice: Community-led practices to build the worlds we need.* MIT Press.

[26] Kathryn Cunningham, Shannon Ke, Mark Guzdial, and Barbara Ericson. 2019. Novice rationales for sketching and tracing, and how they try to avoid it. In *Proceedings of the 2019 acm conference on innovation and technology in computer science education*. 37–43. doi:10.1145/3304221.3319788

[27] Sayamindu Dasgupta and Benjamin Mako Hill. 2017. Learning to code in localized programming languages. In *ACM Conference on Learning @ Scale*. 33–39. doi:10.1145/3051457.3051464

[28] Paul E. Dickson, Neil C. C. Brown, and Brett A Becker. 2020. Engage against the machine: Rise of the notional machines as effective pedagogical devices. In *ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. 159–165. doi:10.1145/3341525.3387404

[29] Jerry A. DiColo. 2009. Numbers don't add up for a TI Calculator. *The Wall Street Journal* (2009). https://www.wsj.com/articles/SB125244891686393811

[30] Ron Eglash, Audrey Bennett, Laquana Cooke, William Babbitt, and Michael Lachney. 2021. Counter-hegemonic computing: Toward computer science education for value generation and emancipation. *ACM Transactions on Computing Education (TOCE)* 21, 4 (2021), 1–30. doi:10.1145/3449024

[31] Md Ehtesham-Ul-Haque, Syed Mostofa Monsur, and Syed Masum Billah. 2022. Grid-coding: An accessible, efficient, and structured coding paradigm for blind and low-vision programmers. In *ACM Symposium on User Interface Software and Technology (UIST)*. 1–21. doi:10.1145/3526113.3545620

[32] Sheena Erete, Karla Thomas, Denise Nacu, Jessa Dickinson, Naomi Thompson, and Nichole Pinkard. 2021. Applying a transformative justice approach to encourage the participation of Black and Latina Girls in computing. *ACM Transactions on Computing Education (TOCE)* 21, 4 (2021), 1–24. doi:10.1145/3451345

[33] Thomas Hylland Eriksen. 1992. Linguistic hegemony and minority resistance. *Journal of Peace Research* 29, 3 (1992), 313–332. doi:10.1177/0022343392029003007

[34] Virginia Eubanks. 2018. *Automating inequality: How high-tech tools profile, police, and punish the poor.* St. Martin's Press.

[35] Jayne Everson, F Megumi Kivuva, and Amy J Ko. 2022. "A key to reducing inequities in like, AI, is by reducing inequities everywhere first": Emerging critical consciousness in a co-constructed secondary CS classroom. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 209–215. doi:10.1145/3478431.3499395

[36] Jayne Everson, F Megumi Kivuva, Eman Sherif, Alannah Oleson, and Amy J Ko. 2026. Deconstructing Conceptions of Rigor in Computer Science Education. *ACM Transactions on Computing Education* 26, 1 (2026), 1–25.

[37] Jayne Everson, Rotem Landesman, F Megumi Kivuva, and Amy J Ko. 2025. Dreaming of difference: Imagining the future of CS education pedagogy with secondary students. In *ACM Conference on International Computing Education Research (ICER)*. 394–406. doi:10.1145/3702652.3744203

[38] Cheri Fancsali, Linda Tigani, Paulina Toro Isaza, and Rachel Cole. 2018. A landscape study of computer science education in NYC: Early findings and implications for policy and practice. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. Association for Computing Machinery, 44–49. doi:10.1145/3159450.3159467

[39] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2018. *How to design programs: An introduction to programming and computing.* MIT Press.

[40] Casey Fiesler, Shannon Morrison, R Benjamin Shapiro, and Amy S Bruckman. 2017. Growing their own: Legitimate peripheral participation for computational learning in an online fandom community. In *ACM Conference on Computer Supported Cooperative Work and Social Computing*. 1375–1386. doi:10.1145/2998181.2998210

[41] Sally Fincher, Johan Jeuring, Craig S Miller, Peter Donaldson, Benedict Du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, et al. 2020. Notional machines in computing education: The education of attention. In *ACM Innovation and Technology in Computer Science Education (ITiCSE)*. 21–50. doi:10.1145/3437800.3439202

[42] Kathi Fisler. 2014. The recurring rainfall problem. In *ACM Conference on International Computing Education Research (ICER)*. 35–42. doi:10.1145/2632320.2632346

[43] Abraham E. Flanigan, Markeya S. Peteranetz, Duane F. Shell, and Leen-Kiat Soh. 2023. Relationship between implicit intelligence beliefs and maladaptive self-regulation of learning. *ACM Transactions on Computing Education (TOCE)* 23, 3 (2023), 1–23. doi:10.1145/3595187

[44] Matthew Flatt. 2012. Creating languages in Racket. *Commun. ACM* 55, 1 (2012), 48–56. doi:10.1145/2063176.2063195

[45] Ruth Frankenberg. 2020. The mirage of an unmarked whiteness. In *The New Social Theory Reader*. Routledge, 416–421. doi:10.4324/9781003060963

[46] Neil Fraser. 2015. Ten things we've learned from Blockly. In *IEEE Blocks and Beyond Workshop*. IEEE, 49–50. doi:10.1109/BLOCKS.2015.7369000

[47] Paulo Freire. 1968. *Pedagogy of the oppressed*. Routledge.

[48] Varvara Garneli, Michail N. Giannakos, and Konstantinos Chorianopoulos. 2015. Computing education in K-12 schools: A review of the literature. In *IEEE Global Engineering Education Conference (EDUCON)*. 543–551. doi:10.1109/EDUCON.2015.7096023

[49] Geneva Gay. 2015. The what, why, and how of culturally responsive teaching: International mandates, challenges, and opportunities. *Multicultural Education Review* 7, 3 (2015), 123–139. doi:10.1080/2005615X.2015.1072079

[50] Bishnu Goswami and Sarmila Pal. 2022. Introduction of two new programming tools in Bengali and measurement of their reception among high-school students in Purba Bardhaman, India with the prototypic inclusion of a vector-biology module. *Education and Information Technologies* (2022), 1–23. doi:10.1007/s10639-021-10663-4

[51] Thomas R. G. Green. 1989. Cognitive dimensions of notations. *People and Computers V* (1989), 443–460.

[52] Mark Guzdial. 2003. A Media computation course for non-majors. In *ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, 104–108. doi:10.1145/961290.961542

[53] Mark Guzdial. 2022. Teaspoon languages for integrating programming into social studies, language arts, and mathematics secondary Courses. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. ACM, 1027. doi:10.1145/3478432.3499240

[54] Carmen Nayeli Guzman, Anne Xu, and Adalbert Gerald Soosai Raj. 2021. Experiences of non-native English speakers learning computer science in a US university. In *ACM Technical Symposium on Computer Science Education (SIGCSE*. 633–639. doi:10.1145/3408877.3432437

[55] Alex Hadwen-Bennett, Lulu Healy, and Sue Sentance. 2025. An embodied sense of programming: an anti-ableist framework for the analysis of learners' developing senses of programming. *ACM Transactions on Computing Education* (2025).

[56] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. 2018. Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools* 2, 2 (2018), 3–13. doi:10.21585/ijcses.v2i2.25

[57] Christina N. Harrington, Aashaka Desai, Aaleyah Lewis, Sanika Moharana, Anne Spencer Ross, and Jennifer Mankoff. 2023. Working at the intersection of race, disability and accessibility. In *ACM SIGACCESS International Conference on Computers and Accessibility (ASSETS)*. ACM, Article 26, 18 pages. doi:10.1145/3597638.3608389

[58] Brian Harvey, Daniel D Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. Snap! (build your own blocks). In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 759–759. doi:10.1145/2445196.2445507

[59] Colin Hennessy Elliott, Jessie Nixon, Alexandra Gendrau Chakarov, Jeffrey B Bush, Michael J Schneider, and Mimi Recker. 2024. Characterizing teacher support of debugging with physical computing: Debugging pedagogies in practice. *ACM Transactions on Computing Education (TOCE)* 24, 4 (2024), 1–28. doi:10.1145/3677612

[60] Felienne Hermans. 2020. Hedy: A gradual language for programming education. In *ACM Conference on International Computing Education Research (ICER)*. 259–270. doi:10.1145/3372782.3406262

[61] Felienne Hermans and Ari Schlesinger. 2024. A case for feminism in programming language design. In *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 205–222. doi:10.1145/3689492.3689809

[62] Charlotte Hill, Hilary A Dwyer, Tim Martinez, Danielle Harlow, and Diana Franklin. 2015. Floors and flexibility: Designing a programming environment for 4th-6th grade classrooms. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 546–551. doi:10.1145/2676723.2677275

[63] Bell Hooks. 1994. *Teaching to transgress*. Routledge.

[64] Asao B Inoue. 2015. *Antiracist writing assessment ecologies: Teaching and assessing writing for a socially just future*. Parlor Press LLC.

[65] Maya Israel, Latoya Chandler, Alexis Cobo, and Lauren Weisberg. 2023. Increasing access, participation and inclusion within K–12 CS education through universal design for learning and high leverage practices. In *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing. https://www.google.com/books/edition/Computer_Science_Education/xUmlEAAAQBAJ?hl=en&gbpv=0

[66] Maya Israel, Gakyung Jeong, Meg Ray, and Todd Lash. 2020. Teaching elementary computer science through universal design for learning. In *Proceedings of the 51st ACM technical symposium on computer science education*. 1220–1226. doi:10.1145/3328778.3366823

[67] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2019. Analysis and modeling of the governance in general programming languages. In *ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 179–183. doi:10.1145/3357766.3359533

[68] Michelle M Jacob. 2013. *Yakama rising: Indigenous cultural revitalization, activism, and healing*. University of Arizona Press.

[69] Sharin Rawhiya Jacob, Jonathan Montoya, Ha Nguyen, Debra Richardson, and Mark Warschauer. 2022. Examining the what, why, and how of multilingual student identity development in computer science. *ACM Transactions on Computing Education (TOCE)* 22, 3 (2022), 1–33. doi:10.1145/3500918

[70] Shaun K Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: Accessible programming for accessible audio games. In *ACM Conference on Interaction Design and Children (IDC)*. 132–142. doi:10.1145/3202185.3202754

[71] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *Comput. Surveys* 37, 2 (2005), 83–137. doi:10.1145/1089733.1089734

[72] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling Alice motivates middle school girls to learn computer programming. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. 1455–1464. doi:10.1145/1240624.1240844

[73] Annie Kelly, Lila Finch, Monica Bolles, and R. Benjamin Shapiro. 2018. BlockyTalky: New programmable tools to enable students' learning networks. *International Journal of Child-Computer Interaction* 18 (2018), 8–18. doi:10.1016/j.ijcci.2018.03.004

[74] Brian Kelly, David Sloan, Lawrie Phipps, Helen Petrie, and Fraser Hamilton. 2005. Forcing standardization or accommodating diversity? A framework for applying the WCAG in the real world. In *International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. 46–54. doi:10.1145/1061811.1061820

[75] Taj Muhammad Khan and Syed Waqar Nabi. 2021. English versus native language for higher education in computer science: A pilot study. In *ACM Koli Calling International Conference on Computing Education Research*. 1–5. doi:10.1145/3488042.3488070

[76] Mara Kirdani-Ryan, Amy J. Ko, and Emilia A. Borisova. 2023. "Taught to be automata": Examining the departmental role in shaping initial career choices of computing students. *Computer Science Education* (2023), 1–27. doi:10.1080/08993408.2023.2171689

[77] Amy J. Ko, Carlos Aldana Lira, and Isabel Amaya. 2025. Wordplay: Accessible, Multilingual, Interactive Typography. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. 1–20. doi:10.1145/3706598.3713196

[78] Amy J. Ko, Anne Beitlers, Jayne Everson, Brett Wortzman, and Dan Gallagher. 2023. Proposing, planning, and teaching an equity-and justice-centered secondary pre-service CS teacher education program. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 583–589. doi:10.1145/3545945.3569735

[79] Amy J. Ko and Brad A. Myers. 2004. Designing the Whyline: A debugging interface for asking questions about program behavior. In *ACM SIGCHI Conference on Human Factors in Computing Systems*. 151–158. doi:10.1145/985692.985712

[80] Amy J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *IEEE Symposium on Visual Languages-Human Centric Computing (VL/HCC)*. IEEE, 199–206. doi:10.1109/VLHCC.2004.47

[81] Amy J. Ko, Alannah Oleson, Mara Kirdani-Ryan, Yim Register, Benjamin Xie, Mina Tari, Matthew Davidson, Stefania Druga, and Dastyni Loksa. 2020. It is time for more critical CS education. *Commun. ACM* 63, 11 (2020), 31–33. doi:10.1145/3424000

[82] Michael Lachney, Jean Ryoo, and Rafi Santo. 2021. Introduction to the special section on justice-centered computing education, part 1. 15 pages. doi:10.1145/3477981

[83] Gloria Ladson-Billings. 1995. Toward a theory of culturally relevant pedagogy. *American Educational Research Journal* 32, 3 (1995), 465–491. doi:10.3102/00028312032003465

[84] Gloria Ladson-Billings and William F. Tate IV. 1995. Toward a critical race theory of education. *Teachers College Record* 97, 1 (1995), 47–68. doi:10.1177/016146819509700104

[85] Michael J Lee. 2014. Gidget: An online debugging game for learning and engagement in computing education. In *IEEE Symposium on Visual Languages and Human-Centric Computing (vl/hcc)*. IEEE, 193–194. doi:10.1109/VLHCC.2014.6883051

[86] Yinchen Lei and Meghan Allen. 2022. English language learners in computer science education: A scoping review. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 57–63. doi:10.1145/3478431.3499299

[87] Kevin Lin. 2022. CS education for the socially-just worlds we need: The case for justice-centered approaches to CS in higher education. In *ACM Technical Symposium on Computer Science Education*. 265–271. doi:10.1145/3478431.3499291

[88] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *ACM SIGCHI Conference on Human Factors in Computing Systems*. 1449–1461. doi:10.1145/2858036.2858252

[89] Audre Lorde. 1983. The master's tools will never dismantle the master's house. In *This Bridge Called My Back: Writings by Radical Women of Colour*. Kitchen Table Press.

[90] Nigel Love. 2017. On languaging and languages. *Language Sciences* 61 (2017), 113–147. doi:10.1016/j.langsci.2017.04.001

[91] Kelly Mack, Emma McDonnell, Dhruv Jain, Lucy Lu Wang, Jon E. Froehlich, and Leah Findlater. 2021. What do we mean by "accessibility research"? A literature survey of accessibility papers in CHI and ASSETS from 1994 to 2019. In *ACM CHI Conference on Human Factors in Computing Systems (CHI)*. 1–18. doi:10.1145/3411764.3445412

[92] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch programming language and environment. *ACM Transactions on Computing Education* 10, 4, Article 16 (November 2010), 15 pages. doi:10.1145/1868358.1868363

[93] Yana Malysheva and Caitlin Kelleher. 2020. Using bugs in student code to predict need for help. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–6. doi:10.1109/VL/HCC50065.2020.9127252

[94] Richard A. Marcantonio, Sean P. Field, Papanie Bai Sesay, and Gary A. Lamberti. 2021. Identifying human health risks from precious metal mining in Sierra Leone. *Regional environmental change* 21, 1 (2021), 2. doi:10.1007/s10113-020-01731-5

[95] Raina Mason and Carolyn Seton. 2021. Leveling the playing field for international students in IT courses. In *Australasian Computing Education Conference (ACE)*. 138–146. doi:10.1145/3441636.3442316

[96] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: A review of the literature from an educational perspective. *Computer Science Education* 18, 2 (2008), 67–92. doi:10.1080/08993400802114581

[97] Monica M. McGill, Leigh Ann DeLyser, Ismaila Temitayo Sanusi, and Selina Marianna Shah. 2023. Meeting the needs of all learners through high quality K-12 computing education research. In *ACM Conference on Global Computing Education (CompEd)*. Association for Computing Machinery, 185–186. doi:10.1145/3617650.3624927

[98] L. McIver and D. Conway. 1996. Seven deadly sins of introductory programming language design. In *International Conference Software Engineering, Education and Practice*. 309–316. doi:10.1109/SEEP.1996.534015

[99] Gabriel Medina-Kim. 2021. Towards justice in undergraduate computer science education: Possibilities in power, equity, and praxis. In *ASEE Virtual Annual Conference Content Access*. https://peer.asee.org/37923.pdf

[100] Hugh Mehan. 2022. Understanding inequality in schools: The contribution of interpretative studies. In *Handbuch Bildungs- und Erziehungssoziologie*. Springer. doi:10.1007/978-3-658-31395-1_22-1

[101] Jodi Melamed. 2015. Racial capitalism. *Critical Ethnic Studies* 1, 1 (2015), 76–85. doi:10.5749/jcritethnstud.1.1.0076

[102] Lauren R Milne. 2017. Blocks4All: Making block programming languages accessible for blind children. *ACM SIGACCESS Accessibility and Computing* 117 (2017), 26–29. doi:10.1145/3051519.3051525

[103] Amr Mohamed, Maram Assi, and Mariam Guizani. 2025. The impact of LLM-assistants on software developer productivity: A systematic literature review. *arXiv* (2025). doi:10.48550/arXiv.2507.03156

[104] Luis Morales-Navarro and Yasmin B Kafai. 2023. Conceptualizing approaches to critical computing education: Inquiry, design, and reimagination. In *Past, Present and Future of Computing Education Research: A Global Perspective*. Springer, 521–538. doi:10.1007/978-3-031-25336-2_21

[105] Aboubakar Mountapmbeme, Obianuju Okafor, and Stephanie Ludi. 2022. Accessible blockly: An accessible block-based programming library for people with visual impairments. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–15. doi:10.1145/3517428.3544806

[106] Aboubakar Mountapmbeme, Obianuju Okafor, and Stephanie Ludi. 2022. Addressing accessibility barriers in programming for people with visual impairments: A literature review. *ACM Transactions on Accessible Computing (TACCESS)* 15, 1 (2022), 1–26. doi:10.1145/3507469

[107] Brad A. Myers, Amy J. Ko, Thomas D. LaToza, and YoungSeok Yoon. 2019. Human-centered methods to boost productivity. *Rethinking Productivity in Software Engineering* (2019), 147–157. doi:10.1007/978-1-4842-4221-6_13

[108] Na'ilah Suad Nasir. 2002. Identity, goals, and learning: Mathematics in cultural practice. *Mathematical thinking and learning* 4, 2-3 (2002), 213–247. doi:10.1207/S15327833MTL04023_6

[109] Greg L. Nelson, Benjamin Xie, and Amy J. Ko. 2017. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *ACM Conference on International Computing Education Research (ICER)*. ACM, 2–11. doi:10.1145/3105726.3106178

[110] Kate J. Neville and Sarah J. Martin. 2023. Slow justice: A framework for tracing diffusion and legacies of resistance. *Social Movement Studies* 22, 2 (2023), 190–210. doi:10.1080/14742837.2022.2031955

[111] Pippa Norris. 2004. Young people & political activism. In *Civic Engagement in the 21st Century: Toward a Scholarly and Practical Agenda*.

[112] Oluwakemi Ola. 2023. Using near-peer interviews to support English language learners. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 952–958. doi:10.1145/3545945.3569868

[113] Alannah Oleson. 2022. CIDER: A method to teach practical critical software design skills. In *ACM Conference on International Computing Education Research (ICER)*. 7–9. doi:10.1145/3501709.3544295

[114] Jan Olsson and Erik Hysing. 2012. Theorizing inside activism: Understanding policymaking and policy change from below. *Planning Theory & Practice* 13, 2 (2012), 257–273. doi:10.1080/14649357.2012.677123

[115] Anne T Ottenbreit-Leftwich, Sarah Dunton, Carol Fletcher, Joshua Childs, Minji Jeon, Maureen Biggers, Leigh Ann DeLyser, John Goodhue, Debra Richardson, Alan Peterfreund, Mark Guzdial, Rick Adrion, Barbara Ericson, Renee Fall, and Victoria Abramenka. 2022. How to change a state: Broadening participation in K-12 computer science education. *Policy Futures in Education* (2022). doi:10.1177/14782103221123363

[116] John F. Pane, Brad A. Myers, et al. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* 54, 2 (2001), 237–264. doi:/10.1006/ijhc.2000.0410

[117] Seymour A Papert. 1981. *Mindstorms: Children, computers, and powerful ideas*. Basic books.

[118] Django Paris. 2012. Culturally sustaining pedagogy: A needed change in stance, terminology, and practice. *Educational Researcher* 41, 3 (2012), 93–97. doi:10.3102/0013189X12441244

[119] Miranda C. Parker and Leigh Ann DeLyser. 2017. Concepts and practices: Designing and developing a modern K-12 CS framework. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 453–458. doi:10.1145/3017680.3017778

[120] Piumi Perera and Supunmali Ahangama. 2021. SimplyTrans: A simplified approach to Sinhala-based coding and introductory programming language localization. In *International Conference on Industrial and Information Systems (ICIIS)*. IEEE, 318–323. doi:10.1109/ICIIS53135.2021.9660709

[121] Chris Piech and Sami Abu-El-Haija. 2020. Human languages in source code: Auto-translation for localized instruction. In *ACM Conference on Learning @ Scale*. 167–174. doi:10.1145/3386527.3405916

[122] Ana Cristina Pires, Filipa Rocha, Antonio José de Barros Neto, Hugo Simão, Hugo Nicolau, and Tiago Guerreiro. 2020. Exploring accessible programming with educators and visually impaired children. In *ACM Interaction Design and Children Conference (IDC)*. 148–160. doi:10.1145/3392063.3394437

[123] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. Codetalk: Improving programming environment accessibility for visually impaired developers. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. 1–11. doi:10.1145/3173574.3174192

[124] Yolanda A Rankin, Jakita O Thomas, and Sheena Erete. 2021. Black women speak: Examining power, privilege, and identity in CS education. *ACM Transactions on Computing Education (TOCE)* 21, 4 (2021), 1–31. doi:10.1145/3451344

[125] John Rawls. 1971. *A theory of justice*. Routledge.

[126] Justin Reich. 2020. *Failure to disrupt: Why technology alone can't transform education*. Harvard University Press.

[127] Luz Rello and Ricardo Baeza-Yates. 2016. The effect of font type on screen readability by people with dyslexia. *ACM Transactions on Accessibility in Computing* 8, 4, Article 15 (May 2016), 33 pages. doi:10.1145/2897736

[128] Filipa Rocha, Filipa Correia, Isabel Neto, Ana Cristina Pires, João Guerreiro, Tiago Guerreiro, and Hugo Nicolau. 2023. Coding together: On co-located and remote collaboration between children with mixed-visual abilities. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*. 1–14. doi:3544548.3581261

[129] Wendy Roldan, Karla Badillo-Urquiola, Kiley Sobel, Kung Jin Lee, Pamela J. Wisniewski, June Ahn, Tamara Clegg, and Jason Yip. 2021. Justice-centered design engagements with children and teens: What's at stake, the actions we take, and the commitments we make. In *ACM Interaction Design and Children Conference (IDC)*. 666–669. doi:10.1145/3459990.3460515

[130] Wendy Roldan, Kung Jin Lee, Kevin Nguyen, Lia Berhe, and Jason Yip. 2022. Disrupting computing education: Teen-led participatory design in libraries. *ACM Transactions on Computing Education (TOCE)* 22, 3 (2022), 1–33. doi:10.1145/3484494

[131] Ricarose Roque. 2020. Building projects, building relationships: Designing for family learning. In *Designing Constructionist Futures: The Art, Theory, and Practice of Learning Designs*. MIT Press, 195–203. doi:10.7551/mitpress/12091.003.0026

[132] David Rose. 2000. Universal design for learning. *Journal of Special Education Technology* 15, 4 (2000), 47–51. doi:10.1177/016264340001500407

[133] Michael Sandel. 2010. *Justice: What's the right thing to do?* Farrar, Straus and Giroux.

[134] Rafi Santo, Leigh Ann Delyser, and June Ahn. 2023. Booting the system: leadership practices for initiating and infrastructuring district-wide computer science instructional programs. *Policy Futures in Education* (2023). doi:10.1177/14782103231178069

[135] Rafi Santo, Leigh Ann DeLyser, June Ahn, Anthony Pellicone, Julia Aguiar, and Stephanie Wortel-London. 2019. Equity in the who, how and what of computer science education: K12 school district conceptualizations of equity in 'CS for All' initiatives. In *Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*. IEEE, 1–8. doi:10.1109/RESPECT46404.2019.8985901

[136] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-based programming for visually-impaired programmers. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. ACM, 773–779. doi:10.1145/3287324.3287499

[137] Kimberly A. Scott, Kimberly M. Sheridan, and Kevin Clark. 2015. Culturally responsive computing: A theory revisited. *Learning, Media and Technology* 40, 4 (2015), 412–436. doi:10.1080/17439884.2014.924966

[138] Robert Sebesta. 2015. *Concepts of programming languages*. Pearson.

[139] Julia Serano. 2007. *Whipping girl: A transsexual woman on sexism and the scapegoating of femininity*. Hachette UK.

[140] R. Benjamin Shapiro, Kayla DesPortes, and Betsy DiSalvo. 2023. Improving computing education research through valuing design. *Commun. ACM* 66, 8 (2023), 24–26. doi:10.1145/3604633

[141] Ather Sharif, Ploypilin Pruekcharoen, Thrisha Ramesh, Ruoxi Shang, Spencer Williams, and Gary Hsieh. 2022. "What's going on in accessibility research?" Frequencies and trends of disability categories and research domains in publications at ASSETS. In *ASSETS*. 46–1. doi:10.1145/3517428.3550359

[142] Vishal Sharma, Neha Kumar, and Bonnie Nardi. 2023. Post-growth Human-Computer Interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)* 31, 1 (2023), 1–37. doi:10.1145/3624981

[143] Molly V Shea, A Susan Jurow, Jovita Schiffer, Meg Escudé, and Aurora Torres. 2023. Infrastructural injustices in community-driven afterschool STEAM. *Journal of Research in Science Teaching* (2023). doi:10.1002/tea.21852

[144] Robert M. Siegfried, Katherine G. Herbert-Berger, Kees Leune, and Jason P. Siegfried. 2021. Trends Of commonly used programming languages in CS1 And CS2 learning. In *International Conference on Computer Science & Education (ICCSE)*. 407–412. doi:10.1109/ICCSE51940.2021.9569444

[145] Geovana Silva, Giovanni Santos, Edna Dias Canedo, Vandor Rissoli, Bruno Praciano, and Guilherme Andrade. 2020. Impact of calango language in an introductory computer programming course. In *IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9. doi:10.1109/FIE44824.2020.9274150

[146] Andreas Stefik and Stefan Hanenberg. 2014. The programming language wars: Questions and responsibilities for the programming language community. In *ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. ACM, 283–299. doi:10.1145/2661136.2661156

[147] Andreas Stefik and Richard Ladner. 2017. The Quorum programming language. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*. 641–641. doi:10.1145/3017680.3022377

[148] Andreas Stefik and Richard E Ladner. 2015. Introduction to AccessCS10K and accessible tools for teaching programming. In *ACM Technical Symposium on Computer Science Education*. 518–519. doi:10.1145/2676723.2677321

[149] Stephanie Tena-Meza, Miroslav Suzara, and AJ Alvero. 2022. Coding with purpose: Learning AI in rural California. *ACM Transactions on Computing Education (TOCE)* 22, 3 (2022), 1–18. doi:10.1145/3513137

[150] Tiffany Tseng, Matt J. Davidson, Luis Morales-Navarro, Jennifer King Chen, Victoria Delaney, Mark Leibowitz, Jazbo Beason, and R. Benjamin Shapiro. 2024. Co-ML: Collaborative machine learning model building for developing dataset design practices. *ACM Transactions on Computing Education (TOCE)* (2024). doi:10.1145/3641552

[151] Tiffany Tseng, Jennifer King Chen, Mona Abdelrahman, Mary Beth Kery, Fred Hohman, Adriana Hilliard, and R. Benjamin Shapiro. 2023. Collaborative machine learning model building with families using Co-ML. In *ACM Interaction Design and Children Conference (IDC)*. 40–51. doi:10.1145/3585088.3589356

[152] Ethel Tshukudu, Emma Dodoo, Felienne Hermans, and Monkgogi Mudongo. 2024. Bilingual programming: A study of student attitudes and experiences in the African context. In *ACM Koli Calling International Conference on Computing Education Research*. 1–11. doi:10.1145/3699538.3699561

[153] Ethel Tshukudu and Siri Annethe Moe Jensen. 2020. The role of explicit instruction on students learning their second programming language. In *United Kingdom & Ireland Computing Education Research Conference*. ACM, 10–16. doi:10.1145/3416465.3416475

[154] Judith Uchidiuno, Amy Ogan, Evelyn Yarzebinski, and Jessica Hammer. 2016. Understanding ESL students' motivations to increase MOOC accessibility. In *ACM conference on Learning @ Scale*. 169–172. doi:10.1145/2876034.2893398

[155] Sepehr Vakil. 2018. Ethics, identity, and political vision: Toward a justice-centered approach to equity in computer science education. *Harvard Educational Review* 88, 1 (2018), 26–52. doi:10.17763/1943-5045-88.1.26

[156] Sepehr Vakil and Maxine McKinney de Royston. 2022. Youth as philosophers of technology. *Mind, Culture, and Activity* 29, 4 (2022), 336–355. doi:10.1080/10749039.2022.2066134

[157] Anna van Der Meulen, Mijke Hartendorp, Wendy Voorn, and Felienne Hermans. 2022. The perception of teachers on usability and accessibility of programming materials for children with visual impairments. *ACM Transactions on Computing Education (TOCE)* 23, 1 (2022), 1–21. doi:10.1145/3561391

[158] Jan Van Dijk. 2020. *The digital divide*. John Wiley & Sons.

[159] Sara Vogel. 2021. "Los programadores debieron pensarse como dos veces": Exploring the intersections of language, power, and technology with bi/multilingual students. *ACM Transactions on Computing Education (TOCE)* 21, 4 (2021), 1–25. doi:10.1145/3447379

[160] Sara Vogel, Christopher Hoadley, Ana Rebeca Castillo, and Laura Ascenzi-Moreno. 2020. Languages, literacies and literate programming: can we use the latest theories on how bilingual people learn to help us teach computational literacies? *Computer Science Education* 30, 4 (2020), 420–443. doi:10.1080/08993408.2020.1751525

[161] Jing Wang and Yubing Xu. 2021. Internet usage, human capital and CO2 emissions: A global perspective. *Sustainability* 13, 15 (2021), 8268. doi:10.3390/su13158268

[162] Mark R. Warren. 2014. Transforming public education: The need for an educational justice movement. *New England Journal of Public Policy* 26, 1 (2014), 11. https://scholarworks.umb.edu/nejpp/vol26/iss1/11

[163] A. N. Washington. 2022. Teaching to transgress in computing: Faculty perspectives on developing identity-inclusive computing courses. In *IEEE Conference on Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*.

[164] David Weintrop and Uri Wilensky. 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *ACM International Conference on Interaction Design and Children (IDC)*. 199–208. doi:10.1145/2771839.2771860

[165] Uri Wilensky and Seymour Papert. 2010. Restructurations: Reformulations of knowledge disciplines through new representational forms. *Constructionism* 17, 2010 (2010), 1–15. https://ccl.northwestern.edu/2010/wilensky_restructurations_Constructionism%202010-latest.pdf

[166] Isabel Wilkerson. 2020. *Caste: The origins of our discontents*. Random House.

[167] Langdon Winner. 2017. Do artifacts have politics? In *Computer Ethics*. Routledge, 177–192. doi:10.4324/9781315259697

[168] Benjamin Xie, Greg L. Nelson, and Amy J. Ko. 2018. An explicit strategy to scaffold novice program tracing. In *ACM Technical Symposium on Computer Science Education*. 344–349. doi:10.1145/3159450.3159527

[169] Aman Yadav and Marc Berges. 2019. Computer science pedagogical content knowledge: Characterizing teacher performance. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–24. doi:10.1145/3303770

[170] Aman Yadav, Marie Heath, and Amber Hu. 2022. Toward justice in computer science through community, criticality, and citizenship. *Commun. ACM* 65, 5 (2022), 42–44. doi:10.1145/3527203

[171] José Pablo Zagal and Amy S. Bruckman. 2005. From samba schools to computer clubhouses: Cultural institutions as learning environments. *Convergence* 11, 1 (2005), 88–105. doi:10.1177/135485650501100107

[172] Abigail Zimmermann-Niefield, Makenna Turner, Bridget Murphy, Shaun K. Kane, and R. Benjamin Shapiro. 2019. Youth learning machine learning through building models of athletic moves. In *ACM International Conference on Interaction Design and Children (IDC)*. 121–132. doi:10.1145/3311927.3323139