Justice-centered educational programming language design

ANONYMOUS AUTHOR(S)

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*, *transtemporal*, *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**.

ACM Reference Format:
Anonymous Author(s). 2018. Justice-centered educational programming language design. *J. ACM* 37, 4, Article 111 (August 2018), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Broadening participation in computing is a profoundly complex endeavor [2], requiring massive, sustainable investments in teacher learning [69], curricula [104], communities [7], policy [36], and a rich ecosystem of informal learning [13]. It requires practice-centered research to ensure that change is effective, sustainable, and spreadable [24, 85]. Doing this equitably is an even grander challenge, given the broad lack of literacy about what equity means amongst teachers, school leaders, not-for-profits, policy makers, and computer scientists [116]. This requires reckoning with systems of power we need to dismantle in order to achieve equity [109], and routine resistance to that change felt daily by students, teachers, and advocates on the margins of computing.

There is one element, however, that receives comparatively little attention in computing education reform efforts: *educational programming languages* (EPL). While in computer science, "PL" often refers only to the academic field of programming languages, or the syntax and semantics of languages, we use the phrase broadly here, as teachers and learners often refer to it: 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.[1] By *educational* PL, we mean any PL that either was designed for learning (e.g., Pyret, Scratch), or is used in purposely pedagogical contexts in tandem with other learning technologies (e.g., Java + BlueJ) [46, 125].

[1]Prior work has sometimes referred to this broader scope as a programming "system" [71]; we use "language" to situate our arguments in terms more familiar to communities of learning, at the expense of some precision.

Manuscript submitted to ACM

Of course, EPL do receive some attention in research and advocacy. Some of us design, build, and maintain them [37, 55, 65, 117, 128], some of us create learning technologies for them (e.g., [82]), many of us study how people learn them (e.g., [40, 95, 133]. 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 [25, 101]. 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 [86]. 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 [127]. 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.

This leads to a fundamental 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 [63]. Some prioritize teaching and instruction [55]. Others prioritize creativity and youth identity [81]. Others still prioritize particular media to support practice-linked identity [94] and skill development in computing, e.g., by concentrating on media manipulation [49], data analysis [22], or storytelling [64]. This multiplicity of designs, while emergent and largely uncoordinated, *has* managed to serve many different communities of teachers and learners, as evidenced by the diversity of PL used in teaching computing [46, 125]. Recent calls have even argued for intentionally creating a multiplicity of small languages that meet a diversity of cross-disciplinary instructional needs [50].

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" [115]), 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 [111, 117]).
- Few EPL attend to *neurodiversity*, offering learners with ADHD[2] 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 [18].
- 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.
- EPL often overlook *financial inequality*: learners who do not have personal computers or access to the Internet cannot access computing education.

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 [9, 51]. 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

[2]Attention-deficit/hyperactivity disorder

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 focus on three questions:

- *RQ1*: How might we conceptualize JCEPL design?
- *RQ2*: What requirements might being justice-centered impose on EPL design?
- *RQ3*: What constraints do current PL design and implementation 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, 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*, 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.

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 [110], 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 — 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, but not necessarily by changing 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* as 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 [115]. Theories of justice centered on identity — for example, disability justice (e.g., [53]), racial justice (e.g., [145]), gender justice (e.g., [120]) — 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: 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 [56]. 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. And 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 literature on *educational justice*. The works most directly inspiring this paper are from thinkers and activists like Freire [45], who rejected banking models of learning (depositing facts into students but not seeing them as bringing valuable resources with them into the classroom), and advocated for critical consciousness through liberatory, community-centered discourse; and from those he inspired, like hooks [56], who 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. We also learn from literature in education sociology and policy [5, 88, 142], 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 [26], which builds on these many notions of justice, and direclty addresses design, unlike these other conceptions of justice. It's framework explain 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* [58] 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 [72, 73, 76, 87], 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 [109]. Scott et al. revisited culturally responsive pedagogy through the lens of collaboration, mentorship, and empowerment through identity [118]. Kirdani-Ryan and Ko [68] 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 [61]. These each illustrate how structural forces in education, including power, norms, and language can shape student learning.

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 [135]. Yadav et al. called for CS education that critically examines computing in the context of community and citizenship [148]. Morales-Navarro and Kafai offer three concrete strategies for bringing criticality into CS classrooms through inquiry, design, and re-imagination [91]. 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 [69]. 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 [124]. Other works disrupt traditional forms of classroom and school authority and focusing students on envisioning futures [19, 32, 35, 113, 130], centering youth ingenuity, counter-narratives, design partnership, and implicit power structures. Some examine the challenges of building relationships across culture and language [6, 139]. Others examine the broader family systems that enable learner-centered creative empowerment with code [114]. All generally 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.

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* [103] 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 [149].

2.3 Inclusive programming language design

Most of PL as a discipline is concerned with correctness, performance, security, and other technical qualities [119], but not justice. There are some efforts to use human-centered design methods to align PL with productivity needs [93]. This includes design methods such as *PLIERS*, [23], 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 [48]. 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 [59].

Some PL design has engaged concerns of equitable teaching. For example, the broader efforts of the How to Design Programs (HtDP) team, including the Bootstrap project, offer several examples [37]. *Racket*'s language levels, for example, offers a sequenced design of PLs, aligned with a learning trajectory in support of learners' diverse prior

knowledge. Bootstrap's subject-specific courses, such as *Bootstrap Physics*, are designed in close collaboration with physics teachers. Guzdial's teaspoon languages [50] similarly center teacher voices in shaping EPL design.

Some PL design focuses on blind and vision impaired (BVI) creators [80, 122]. For example, Quorum has focused on being more screen-readable [128], and innovations such as tangible media for computation [107], programmable audio games [62], tweaks to PL that enable mixed-ability families to code together [112] offer visions of more accessible programming. Other works explore how to make existing PL and tools more screen readable [10, 31, 90, 92, 92, 108, 117]. Some work finds that programmable media needs to build upon youth interests and needs [131, 132, 137], comprehensively support screen reading [52], 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.

There is also a growing, separate literature on supporting multilingual learners in computing education. Some EPL, for example, have been designed to be explicitly multilingual [55]. Most work on linguistic justice, however, has focused on instruction, and not tools. For example, there is evidence that English primacy harms learning [4, 51], and that learning in students' first languages promotes learning, engagement, and belonging [3, 28, 47, 67, 84, 98, 126, 140], but that English primacy in programming and learning materials limits their impact [75]. While these benefits are clear, many learners legitimately see learning English as a goal [134] and many approaches to including learners' first languages inequitably impose translation labor onto youth [105], most of which cannot be automated without compromises to content quality [106]. None of this literature engages the intersection between language and accessibility or teacher facilitation needs.

Aside from this handful of works, nearly all EPL have focused on innovations in new media, graphical editors, and runtime capabilities [63], and not explicitly on ensuring that EPL work for every learner. The 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 EPL might be viewed as a "master's tool" [78]: 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, education, and PL design 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.

How, then, can we organize EPL design to overcome these issues? 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 PL, let alone EPL, and even fewer 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, and evolution 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 yet, building trust to a point where centering their visions and values is possible, is vital JCEPL success.

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.

Design justice [26] offers one lens for thinking through how to distribute power. For example, the *Design Justice Network* has cultivated 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.

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 proposing seven justice-centered "requirements"[3] for EPL, suggesting minimum bar for achieving educational justice. The requirements together constitute, and spell out, a framework we call *ALTCODE*. 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.

Before presenting them, note that we do not claim that this is the *only* manifestation that qualifies as "justice-centered", or that it is even the best. Our focus is 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,

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

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.

3.1 Requirement *Accessible*

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

This requirement argues that all capabilities of an EPL, including use of it and its program outputs, must be usable by all stakeholders, independent of what physical, perceptual, and learning abilities they have. This problematizes the status quo in computing: 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 [20], and do not even require *equitably pleasant* experiences [66]. EPL are part of this broader inaccessible world, but also help reproduce it.

Accessibility is particularly important for 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 [144]. 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 aspect of programs (e.g., assets used in programs such as images, sounds, and other media), must be editable by *any* input modality and device. Anything less excludes people who might only read code via screen or braille reader, or who control a computer through low bandwidth eye or mouth input. This requires creating unified editors that support all 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. Finding and depicting (through whatever modality best serves a learner) symbolic forms to explain that a variable has a new value, that a conditional expression evaluated to false and is branching to an else case, or that a function is returning a value, are all essential, as *requiring* visual inference from program visualizations is ableist.

3.1.1 Analysis. The current ecosystem of EPL varies widely in this requirement. One EPL that does not meet it is *Scratch* [81]. 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!* [54], and led to block interface building toolkits such as *Blockly* [44], 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 block-based

editors outside of dominant platforms [90], 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* [128]. 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 [129].

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, as 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? Partnering with communities to co-design such experiences could 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 [12], 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.*

Many EPL, while used in education settings, do not reshape learners' conceptions of themselves or the world toward justice. Some EPL are platforms for creativity in which such learning might occur, or might not. 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. Moreover, many aspects of PL remain 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

[89] that computing cannot be understood as a social phenomenon without understanding its role in upholding these exploitative systems.

We conceptualize *liberatory* as offering youth the opportunity to deeply question the purpose of computing and computing skills as they learn them, so that they may navigate power structures as they are and harness those skills to create a more just future of computing. This conception argues for a particular kind of learning that evokes hook's arguments [56] 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 [103] 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. For example, EPL might critique themselves, articulating their limits, questioning the impact of abstractions they offer, and link their ideas to broader oppressive forces of racism, sexism, ableism, caste, and their intersections. This calls back to design justice ideas of sharing knowledge with communities, enabling youth to be critical designers themselves [26], to create their own futures.

Some EPL designers might view this requirement as a politicization of inherently apolitical artifacts, especially in light of legislative bans on 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., [14, 16, 34, 146]). These works make plain that behind every technical idea, computing included, are unexamined assumptions and priorities [99] 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*[4], 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 not 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.

While there are limited examples of EPL engaging liberatory discourse, there are some that offer glimpses. For example, the narrative framing of *Gidget*[5], 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 [74]. 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. This narrative is also a pedagogical one, positioning the learner as the person who needs to understand and fix programs to help Gidget complete its human-devised mission. The result is a perception by learners that compilers are just tools and have no claim to authority over people [21].

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

[4] codecombat.com
[5] https://helpgidget.org

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 [77], 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" [136], 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 with 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 *Transtemporal*

> *To foster youth agency via program comprehension, program execution must be navigable in both directions and at multiple levels of granularity.*

If youth and their teachers cannot understand *how* programs work, or *why* they behave the way they do, they cannot build self-efficacy in creating or critiquing programs [11, 41]. Such self-efficacy is central empowering learners to have agency over computing, and thus central to JCEPL.

While many things are necessary for comprehending program behavior, visibility into precisely how programs execute is central, as they do large numbers of things quickly, and by default, invisibly, hiding the complex sequence of events that cause code to produce output. Scaffolding visibility can help [27, 70, 95, 147], but prior work makes clear that precise, temporal, causal, reversible explanations of how programs execute are central to learning a PL's evaluation rules [29]. We therefore argue that *transtemporality* — the ability to fluidly examine program behavior across time and state — is a key requirement.

Transtemporarily could mean many things. Learners should be able to reverse time and inspect a conditional branch's decision. They should be able to demonstrate some input, replay it, and even change an input and replay from that changed point. They should be able request explanations of why a program did or did not do something (a la, [70]). They should be able to slow down or reverse time, to better manage their attention, align better with their motor-physical abilities, or give them 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. Control over time and visibility of causality 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* [42] 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. 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 orient learners towards building knowledge over how programs work, fostering agency.

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 [70], exploring interfaces for interrogating program output and simulating reversibility for simple procedural languages. But how might EPL be architected to allow for fine-grained over 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 (e.g. [65])? 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.

Other challenges are pedagogical. If the point of transtemporality is agency, how can comprehension features help youth examine the limitations of computing? How might transtemporal features be leveraged 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 responsive and sustaining in how they are designed, explained, and framed, enabling identity-inclusive pedagogy.*

Responding to and sustaining culture means more than just engaging culture. As Scott et al. delineate, it means embracing that all youth can innovate with computing, that learning be transformative to youth and their communities, that learning fosters sociocultural understanding and insight about youths' intersectional selves, and that success be framed around who creates and to what end [118]. It means co-constructing anti-racist learning experiences [35] and assessments [57] 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 [43]. 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 [17]. 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 [79]. 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 [139], 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 [60].

3.4.1 Analysis. 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.

One contrast to this is *Hedy*, a multi-lingual EPL that has grammars for 47 of the world's languages [55]. 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. There are many more things Hedy could do be culturally sustaining, such as having multilingual support, rather than segregating individual locales. Its focus on didactic pedagogies is also in tension with learner-centered paradigms for creative programming [114], which center learners' interests instead of examples crafted by educators that can homogenize student learning.

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 universal accessibility (see Section 3.1), may go a long way toward making this possible as well.

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 [39], our machines could operate based on students' notions of how they should work [102]. 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.

This requirement argues 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. The rationale is clear: educational justice, broadly construed, requires educational equity [142], 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 [124]. 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 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 [138] 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 hardware that may be shared resources, where Internet connections may be unreliable or slow. This requirement might be achieved by bridging digital divides [138], 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. It might be linked to digital divide [138] efforts, which imagine futures in which all of humanity has access to computing devices and the Internet.

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*[6]. 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. One example that meets this requirement is the Texas Instruments line of graphing calculators: they are low-cost, portable, battery-powered, often subsidized by schools, 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. 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 [1].

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 learner desires. Some of the challenges that surface in realizing this vision include: 1) how to finance and

[6] https://octostudio.com/

sustain such an ecosystem of hardware and software, 2) how to design its governance structures to center communities, 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.*

We define "open" as the source is available, there is a community process by which anyone can submit change requests to the design or its governance processes, and there is a process by which students and teachers can gain power, particularly if they are not being served by an EPL's design. This implies accountability and 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 [26].

Ultimately, this 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 design 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. 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*[7], 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.

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 [8], and how to organize product management strategies that are community-led, rather than designer-led. Most of partnership work 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.

[7] processingfoundation.org

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.*

This requirement recognizes that EPL are critical infrastructure for learning and that they must be sustainable in order to sustain learning. Many things are required to sustain EPL, including funding to pay for maintenance, but also a community's capacity to maintain 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 [38], including the unique burdens placed on experts that create single points of burnout and abandonment.

Enduring does not necessarily mean that an EPL last forever. Indeed, EPL projects being canceled, or becoming incompatible or unsupported, can be a threat to investments in curriculum, teacher knowledge, and youth identity. 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 [15, 141]), or on material resource mining, should not necessarily endure, especially when they physically harm the communities they serve [83, 123]. 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. *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.

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, contributing to unpredictable impacts on when, how, and whether we compute.

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*) one possible 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. [8]

That said, our community has made progress. 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, even if we have yet to question how or whether they endure. The computing education community is not necessarily "failing" to be justice-*centered* – every project of justice is a slow march [96] – 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 – *transtemporality*, 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 amongst 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 generative AI in EPL, with its clear impacts on 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 [100] or creating new ones [97].

[8] The word "hegemonic" pose 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 [143].

All of this will require research. Scholarly venues have historically been skeptical of analytical critiques of EPL, favoring positivist evaluations of learning outcomes. We hope that *ALTCODE* can provide one basis for rigorous analyses of EPL features and tools instead of empirical studies. We also need venues where EPL analyses and designs are welcomed and fairly evaluated within their epistemic goals. Some venues have signalled such interest [121].

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 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 paper is a catalyst for deeper discourse on how.

REFERENCES

[1] 2009. WSJ(9/9) Numbers Don't Add Up For A TI Calculator.

[2] 2023. Booting the system: Leadership practices for initiating and infrastructuring district-wide computer science instructional programs. *Policy Futures in Education* (2023).

[3] Suad Alaofi and Seán Russell. 2022. A validated computer terminology test for predicting non-native english-speaking CS1 students' academic performance. In *Proceedings of the 24th Australasian Computing Education Conference*. 133–142.

[4] Hend Alrasheed, Amjad Alnashwan, and Ruwayda Alshowiman. 2021. Impact of English Proficiency on Academic Performance of Software Engineering Students. In *2021 4th International Conference on Data Storage and Data Engineering*. 107–111.

[5] Jean Anyon. 2013. Social class, school knowledge, and the hidden curriculum: Retheorizing reproduction. In *Ideology, curriculum, and the new sociology of education*. Routledge, 37–45.

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

[7] William Aspray. 2016. *The Broadening Participation in Computing Alliances*. Springer International Publishing, Cham, 53–102. https://doi.org/10.1007/978-3-319-24832-5_3

[8] Ellen Bacon and Usa Bloom. 2000. Listening to student voices: How student advisory boards can help. *Teaching exceptional children* 32, 6 (2000), 38–43.

[9] Catherine M. Baker, Cynthia L. Bennett, and Richard E. Ladner. 2019. Educational Experiences of Blind Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 759–765. https://doi.org/10.1145/3287324.3287410

[10] 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 *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 3043–3052.

[11] 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 *Proceedings of the SIGCHI Conference on human factors in computing systems*. 869–878.

[12] Andrew Begel and Susan L Graham. 2006. An assessment of a speech-based programming environment. In *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 116–120.

[13] Andrew Begel and Amy J. Ko. 2019. *Cambridge Handbook on Computing Education Research*. Chapter Learning Outside the Classroom.

[14] Ruha Benjamin. 2023. Race after technology. In *Social Theory Re-Wired*. Routledge, 405–415.

[15] Adrien Berthelot, Eddy Caron, Mathilde Jay, and Laurent Lefèvre. 2023. Estimating the environmental impact of Generative-AI services using an LCA-based methodology. (2023).

[16] 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.

[17] George Boole. 2021. An Investigation of the Laws of Thought on which are founded the mathematical theories of Logic and Probabilities (1854). (2021).

[18] 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 *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*. 1–10.

[19] 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.

[20] Toni Cannady. 2019. Classifying WCAG 2.0 Guidlines as the Legal Standard for Website under Title III of the Americans with Disabilities Act. *Cath. UL Rev.* 68 (2019), 209.

[21] Polina Charters, Michael J Lee, Amy J Ko, and Dastyni Loksa. 2014. Challenging stereotypes and changing attitudes: the effect of a brief programming encounter on adults' attitudes toward programming. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 653–658.

[22] 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 *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference* (Chicago, IL, USA) *(IDC '23)*. Association for Computing Machinery, New York, NY, USA, 100–111. https://doi.org/10.1145/3585088.3589364

[23] 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.

[24] Cynthia E Coburn, Amy K Catterson, Jenni Higgs, Katie Mertz, Richard Morel, and Cynthia Coburn. 2013. Spread and scale in the digital age: A memo to the John D. and Catherine T. MacArthur Foundation. *Prepared for a convening at the MacArthur Foundation* (2013).

[25] Code.org. 2023. *2023 State of Computer Science Education.* Technical Report. Code.org.

[26] Sasha Costanza-Chock. 2020. *Design Justice: Community-Led Practices to Build the Worlds We Need.*

[27] 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.

[28] Sayamindu Dasgupta and Benjamin Mako Hill. 2017. Learning to code in localized programming languages. In *Proceedings of the fourth (2017) ACM conference on learning@ scale*. 33–39.

[29] Paul E Dickson, Neil CC Brown, and Brett A Becker. 2020. Engage against the machine: Rise of the notional machines as effective pedagogical devices. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 159–165.

[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.

[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 *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–21.

[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.

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

[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 *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 209–215.

[36] 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 *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 44–49. https://doi.org/10.1145/3159450.3159467

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

[38] 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 *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 1375–1386.

[39] 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 *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. 21–50.

[40] Kathi Fisler. 2014. The recurring rainfall problem. In *Proceedings of the tenth annual conference on International computing education research*. 35–42.

[41] 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* 23, 3 (2023), 1–23.

[42] Matthew Flatt. 2012. Creating languages in Racket. *Commun. ACM* 55, 1 (2012), 48–56.

[43] Ruth Frankenberg. 2020. The mirage of an unmarked whiteness. In *The new social theory reader*. Routledge, 416–421.

[44] Neil Fraser. 2015. Ten things we've learned from Blockly. In *2015 IEEE blocks and beyond workshop (blocks and beyond)*. IEEE, 49–50.

[45] Paulo Freire. 2020. Pedagogy of the oppressed. In *Toward a sociology of education*. Routledge, 374–386.

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

[47] 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.

[48] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[49] Mark Guzdial. 2003. A Media Computation Course for Non-Majors. *SIGCSE Bull.* 35, 3 (jun 2003), 104–108. https://doi.org/10.1145/961290.961542

[50] Mark Guzdial. 2022. Teaspoon Languages for Integrating Programming into Social Studies, Language Arts, and Mathematics Secondary Courses. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1027. https://doi.org/10.1145/3478432.3499240

[51] 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 *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 633–639.

[52] 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.

[53] 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 *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility* (<conf-loc>, <city>New York</city>, <state>NY</state>, <country>USA</country>, </conf-loc>) *(ASSETS '23)*. Association for Computing Machinery, New York, NY, USA, Article 26, 18 pages. https://doi.org/10.1145/3597638.3608389

[54] 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 *Proceeding of the 44th ACM technical symposium on Computer science education*. 759–759.

[55] Felienne Hermans. 2020. Hedy: a gradual language for programming education. In *Proceedings of the 2020 ACM conference on international computing education research*. 259–270.

[56] Bell Hooks. 2014. *Teaching to transgress*. Routledge.

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

[58] 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. *Computer Science Education: Perspectives on Teaching and Learning in School* (2023), 115.

[59] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2019. Analysis and Modeling of the Governance in General Programming Languages. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering* (Athens, Greece) *(SLE 2019)*. Association for Computing Machinery, New York, NY, USA, 179–183. https://doi.org/10.1145/3357766.3359533

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

[61] 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.

[62] Shaun K Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: accessible programming for accessible audio games. In *Proceedings of the 17th ACM conference on interaction design and children*. 132–142.

[63] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR)* 37, 2 (2005), 83–137.

[64] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>San Jose</city>, <state>California</state>, <country>USA</country>, </conf-loc>) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 1455–1464. https://doi.org/10.1145/1240624.1240844

[65] 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.

[66] 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 *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. 46–54.

[67] Taj Muhammad Khan and Syed Waqar Nabi. 2021. English versus native language for higher education in computer science: A pilot study. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research*. 1–5.

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

[69] 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 *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 583–589.

[70] Amy J Ko and Brad A Myers. 2004. Designing the whyline: a debugging interface for asking questions about program behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 151–158.

[71] Amy J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206.

[72] Amy J Ko, Alannah Oleson, Neil 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.

[73] Michael Lachney, Jean Ryoo, and Rafi Santo. 2021. Introduction to the special section on justice-centered computing education, part 1. , 15 pages.

[74] Michael J Lee. 2014. Gidget: An online debugging game for learning and engagement in computing education. In *2014 ieee symposium on visual languages and human-centric computing (vl/hcc)*. IEEE, 193–194.

[75] Yinchen Lei and Meghan Allen. 2022. English Language Learners in Computer Science Education: A Scoping Review. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 57–63.

[76] Kevin Lin. 2022. Cs education for the socially-just worlds we need: The case for justice-centered approaches to cs in higher education. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 265–271.

[77] 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 *Proceedings of the 2016 CHI conference on human factors in computing systems*. 1449–1461.

[78] Audre Lorde. 2003. The master's tools will never dismantle the master's house. *Feminist postcolonial theory: A reader* 25 (2003), 27.

[79] Nigel Love. 2017. On languaging and languages. *Language Sciences* 61 (2017), 113–147.

[80] 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 *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–18.

[81] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (nov 2010), 15 pages. https://doi.org/10.1145/1868358.1868363

[82] Yana Malysheva and Caitlin Kelleher. 2020. Using Bugs in Student Code to Predict Need for Help. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–6. https://doi.org/10.1109/VL/HCC50065.2020.9127252

[83] 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.

[84] Raina Mason and Carolyn Seton. 2021. Leveling the playing field for international students in IT courses. In *Proceedings of the 23rd Australasian Computing Education Conference*. 138–146.

[85] 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 *Proceedings of the ACM Conference on Global Computing Education Vol 2* (<conf-loc>, <city>Hyderabad</city>, <country>India</country>, </conf-loc>) *(CompEd 2023)*. Association for Computing Machinery, New York, NY, USA, 185–186. https://doi.org/10.1145/3617650.3624927

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

[87] Gabriel Medina-Kim. 2021. Towards Justice in Undergraduate Computer Science Education: Possibilities in Power, Equity, and Praxis. In *2021 ASEE Virtual Annual Conference Content Access*.

[88] Hugh Mehan. 1992. Understanding inequality in schools: the contribution of interpretative studies. *Sociology of Education* 65, 1 (1992), 1–20.

[89] Jodi Melamed. 2015. Racial capitalism. *Critical ethnic studies* 1, 1 (2015), 76–85.

[90] Lauren R Milne. 2017. Blocks4All: making block programming languages accessible for blind children. *ACM SIGACCESS Accessibility and Computing* 117 (2017), 26–29.

[91] 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.

[92] 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.

[93] 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.

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

[95] 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 *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/3105726.3106178

[96] 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.

[97] Pippa Norris. 2004. Young people & political activism. *Harvard University, John F. Kennedy School of Government.(32p)* (2004).

[98] Oluwakemi Ola. 2023. Using Near-Peer Interviews to Support English Language Learners. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 952–958.

[99] Alannah Oleson. 2022. CIDER: A Method to Teach Practical Critical Software Design Skills. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2*. 7–9.

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

[101] Anne T Ottenbreit-Leftwich, Sarah Dunton, Carol Fletcher, Joshua Childs, Minji Jeon, Maureen Biggers, Leigh Ann DeLyser, John Goodhue, Debra Richardson, Alan Peterfreund, et al. 2022. How to change a state: Broadening participation in K-12 computer science education. *Policy Futures in Education* (2022), 14782103221123363.

[102] 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.

[103] Seymour A Papert. 2020. *Mindstorms: Children, computers, and powerful ideas.* Basic books.

[104] Miranda C. Parker and Leigh Ann DeLyser. 2017. Concepts and Practices: Designing and Developing A Modern K-12 CS Framework. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 453–458. https://doi.org/10.1145/3017680.3017778

[105] Piumi Perera and Supunmali Ahangama. 2021. SimplyTrans: A Simplified Approach to Sinhala-Based Coding and Introductory Programming Language Localization. In *2021 IEEE 16th International Conference on Industrial and Information Systems (ICIIS)*. IEEE, 318–323.

[106] Chris Piech and Sami Abu-El-Haija. 2020. Human languages in source code: Auto-translation for localized instruction. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*. 167–174.

[107] 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 *Proceedings of the Interaction Design and Children Conference*. 148–160.

[108] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 chi conference on human factors in computing systems*. 1–11.

[109] 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.

[110] John Rawls. 1971. *A Theory of Justice.*

[111] Luz Rello and Ricardo Baeza-Yates. 2016. The Effect of Font Type on Screen Readability by People with Dyslexia. *ACM Trans. Access. Comput.* 8, 4, Article 15 (may 2016), 33 pages. https://doi.org/10.1145/2897736

[112] 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 *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.

[113] 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.

[114] Ricarose Roque. 2020. Building projects, building relationships: Designing for family learning. *Designing Constructionist Futures: The Art, Theory, and Practice of Learning Designs* (2020), 195–203.

[115] Michael Sandel. 2010. *Justice: What's the Right Thing to Do?* Farrar, Straus and Giroux.

[116] 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 *2019 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*. 1–8. https://doi.org/10.1109/RESPECT46404.2019.8985901

[117] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-based programming for visually-impaired programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 773–779.

[118] 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.

[119] Robert Sebesta. 2015. *Concepts of Programming Languages.* Pearson.

[120] Julia Serano. 2007. *Whipping Girl: A Transsexual Woman on Sexism and the Scapegoating of Femininity.*

[121] R Benjamin Shapiro, Kayla DesPortes, and Betsy DiSalvo. 2023. Improving Computing Education Research through Valuing Design. *Commun. ACM* 66, 8 (2023), 24–26.

[122] 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.

[123] Vishal Sharma, Neha Kumar, and Bonnie Nardi. 2023. Post-growth Human–Computer Interaction. *ACM Transactions on Computer-Human Interaction* 31, 1 (2023), 1–37.

[124] 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).

[125] 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 *2021 16th International Conference on Computer Science & Education (ICCSE)*. 407–412. https://doi.org/10.1109/ICCSE51940.2021.9569444

[126] 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 *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[127] Andreas Stefik and Stefan Hanenberg. 2014. The Programming Language Wars: Questions and Responsibilities for the Programming Language Community. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Portland, Oregon, USA) *(Onward! 2014)*. Association for Computing Machinery, New York, NY, USA, 283–299. https://doi.org/10.1145/2661136.

2661156

[128] Andreas Stefik and Richard Ladner. 2017. The quorum programming language. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 641–641.

[129] Andreas Stefik and Richard E Ladner. 2015. Introduction to AccessCS10K and accessible tools for teaching programming. In *Proceedings of the 46th ACM technical symposium on computer science education*. 518–519.

[130] 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.

[131] 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* (2024).

[132] 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 *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*. 40–51.

[133] 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*. (Glasgow, United Kingdom) *(UKICER '20)*. Association for Computing Machinery, New York, NY, USA, 10–16. https://doi.org/10.1145/3416465.3416475

[134] Judith Uchidiuno, Amy Ogan, Evelyn Yarzebinski, and Jessica Hammer. 2016. Understanding ESL Students' Motivations to Increase MOOC Accessibility. In *Proceedings of the third (2016) ACM conference on Learning@ Scale*. 169–172.

[135] 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.

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

[137] 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* 23, 1 (2022), 1–21.

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

[139] 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.

[140] 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.

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

[142] Mark R Warren. 2014. Transforming public education: The need for an educational justice movement. *New England Journal of Public Policy* 26, 1 (2014), 11.

[143] AN Washington. 2022. Teaching to Transgress in Computing: Faculty Perspectives on Developing Identity-Inclusive Computing Courses. In *Conference on Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*.

[144] David Weintrop and Uri Wilensky. 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th international conference on interaction design and children*. 199–208.

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

[146] Langdon Winner. 2017. Do artifacts have politics? In *Computer ethics*. Routledge, 177–192.

[147] Benjamin Xie, Greg L Nelson, and Amy J Ko. 2018. An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM technical symposium on computer science education*. 344–349.

[148] Aman Yadav, Marie Heath, and Anne Drew Hu. 2022. Toward justice in computer science through community, criticality, and citizenship. *Commun. ACM* 65, 5 (2022), 42–44.

[149] 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.