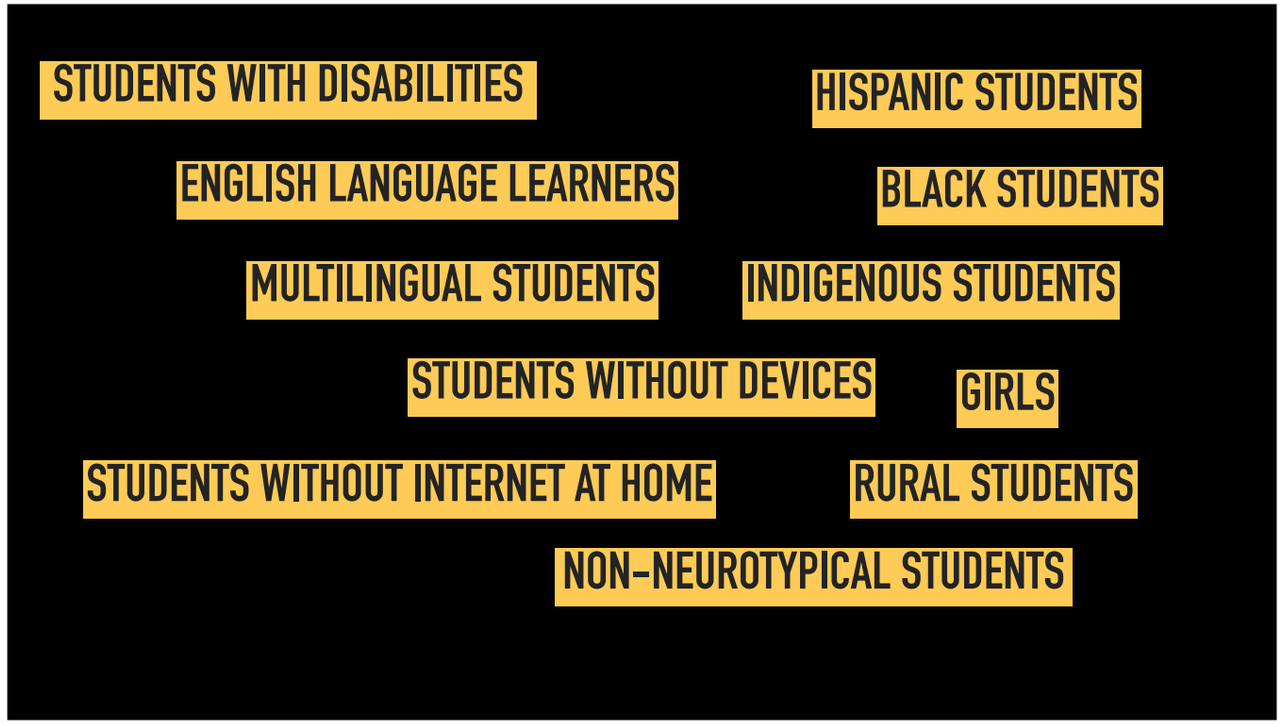
- How is everyone doing?
- It's a time of transitions: Thanksgiving, a destabilizing transfer of power, and the coming decades of climate change are all on my mind, making me wonder about where to put my attention.
- What matters? What can I do now that helps me realize my long term dream of everyone just having peace, safety, respect?
- But it also makes me think about how we got here. How I got here.

- 1997, college visits, this felt like a place I might call home. But I didn’t think I could afford it.
- At the time, I was a particular kind of CS-interested student. I’d spent my middle and high school days mostly learning CS alone: a TI-82, some old 286 PCs in an abandoned computer lab, no teachers, some textbooks from Powell’s technical books. The internet was nascent in the early 90’s and so learning could be quite solitary.
- But I liked it that way: there was no sense that I was ahead or behind, that I belonged or I didn’t. There was no competition to get into CS departments, or to prove myself. It was just me, the weird animations and tools that I made, the little proof of concept games my illustration and music obsessed friends would make.

- How things have changed!
- After the Obama administration fueled the CS for All movement with federal funding in 2016, right at the end of Obama’s second term, the race was on.
- Now, there are millions of students desperate to learn CS, for prestige, economic opportunity, power.
- Some students have teachers, tutors, classrooms, devices, curricula, role models, and more! Maybe that was some of you.
- But this is not everywhere.

- In most schools in this country, there are still no CS classes, no teachers, no community, no pathways — not even awareness of what CS is and how its changing our world.
- Partly, this is just school funding. Our nation's buildings are crumbling, conservative politicians work to move funding away from public schools to private ones, and states, constitutionally mandated to have balanced budgets, look to schools first for cuts when they can't raise taxes.
- But in CS in particular, it's much more than just funding: we're missing teachers, culturally responsive curricula, professional communities, policy. Nearly everything that does exist was built for wealthy, White, and Asian communities, and is responsive to those communities alone.
- This means many things:



- All of these groups of students are excluded from learning
- And these groups are neither monolithic, or separate: every combination of these exists
- Creating computer science education that works for everyone means acknowledging this diversity of identities, communities, and needs, and designing explicitly for that diversity, rather than assuming that everyone will be the same as the groups that dominate CS today.
- That means classrooms that work for wheelchairs
- Teachers of color to role model and mentor around identity
- Bridging digital divides of devices and internet
- Making CS work for all language fluencies, not just English
- Engaging the problems that matter to the infinite diversity of radicalized experiences
- Imagining pedagogies that work for the many ways that students make sense of information and communicate
- Not only is that a lot of work, but it means undoing a lot of work, because most of what we have made for CS education doesn’t work for most of these students.
- There is one thing in particular, though, that I want to focus on today.

PROGRAMMING LANGUAGES (PL)

And all of the ways they interact with funding, curricula, teachers, and other key resources.

- Java, Python, JavaScript, C#, yes,
- But also all of the educational programming languages that exist now, like Scratch, Snap!, Pyret,
- And all the platforms to come before it like Alice, Pascal, BASIC, and more.
- Despite all of these being designed for learning, they do not work for most of the people in the world.
- I'd like to talk about why.
- Let's pick on a few

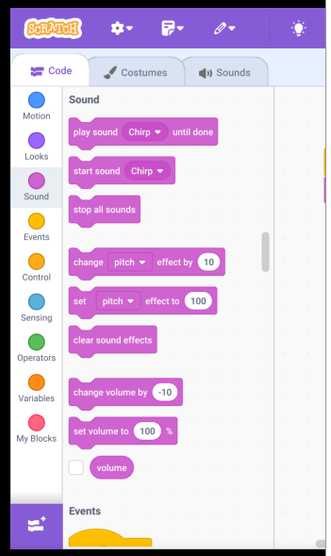
JAVA

- *It is ubiquitous in CS education, including the AP CS A exam that drives so much CS learning in high schools.*
- And yet…
  - English-only syntax and documentation
  - Poor screen readability of code, inaccessible tools
  - Requires installation, device ownership
  - Designed for professionals, not learners

TEXT

SCRATCH

- *Ubiquitous in K-8 CS education, responsive to students desire to express themselves, many curricula options, many languages supported*
- And yet…
  - Requires use of a pointing device, including students who are blind or can't perform fine motor movements
  - Provides almost no debugging support
  - Closed source, unresponsive to teacher and student needs



TEXT

BBC MICRO:BIT

- *Uniquely simplifies sensor-based software programming, creating rich interactive hardware experiences*
- And yet
  - Requires *two* devices to use
  - Inaccessible to blind and motor impaired students
  - Board is primary industry representatives, not teachers or students.

THE EDUCATIONAL PROGRAMMING LANGUAGES WE HAVE ARE DESIGNED FOR THE FEW, NO FOR ALL

WHAT WOULD IT MEAN TO DESIGN PL FOR ALL?

TEXT

THIS TALK

- This will not be a talk about novel algorithms, data structures, or systems. But it will be a **novel argument** about what those systems should be, as philosophers do.
- You will leave with questions and perspectives and gestures toward answers.
- I'll share:
  - My background and positionality
  - What is justice?
  - Seven justice-centered requirements for educational programming languages
  - Q&A

WHO AM I TO SPEAK ON THIS?

Positionality

- I am, in some ways, part of the dominant groups in CS: White, Asian, interested in programming, self-taught, and love computers, English fluent, born in the United States, not disabled.
- But I am on the margins in other ways.
- I am transgender
- I am a woman
- I am mixed race
- I am more interested in what CS can do for people than the ideas in CS themselves
- I do not believe that computing is unequivocally good: it started as a weapon of war, it now regularly amplifies inequality, division, and misinformation.
- But it is also not unequivocally bad: I think of code as a medium that can be used for liberatory purposes as well; it just tends to not be used for those purposes, because we let capitalism drive our priorities instead of freedom or justice.
- My views are marginal in CS, and ones that often lead people to think that I am either not a computer scientist, or do not like computer science.
- But this couldn’t be further from the truth: I love CS and computing, and that is why I am critical of it. I want to search of ways to make it better.

WHAT DOES “BETTER” FOR “EVERYONE” MEAN?

We’ll build on three ideas.

FOUNDATIONS

RAWLSIAN JUSTICE

- John Rawls' seminal *A Theory of Justice* (1971) defines justice through two principles:
  - Every person deserves a claim to the same set of equal basic liberties. (*i.e., there should be no "birthright" to greater freedom*).
  - Any social inequalities must satisfy two conditions:
    - They must stem solely from equality of opportunity (*not birthright)*
    - They must be to the greatest benefit of the least advantaged (*addressing inequities inherent to birth*).

FOUNDATIONS

EDUCATIONAL JUSTICE

- Paulo Freire (*Pedagogy of the Oppressed*) 
  - Rejected school as a context for "depositing" knowledge in minds
  - Viewed education explicitly for fostering liberatory, collective, critical consciousness about learners' "limiting situations", through dialog, mutual understanding
- bell hooks (*Teaching to Transgress*) 
  - Freire's ideas, in practice, are constrained by racial and patriarchal capitalism that Freire overlooked. These social and economic hierarchies limit what dialog students will engage
  - Advocated for school to be a place to see these forces, connect them to students' lived experiences, and organize around dismantling them

FOUNDATIONS

DESIGN JUSTICE

- Sasha Costanza-Chock (*Design Justice)* applies Rawlsian justice to design, centering design choices at the margins, in communities:
  - Heal and empower communities
  - Center direct stakeholder voices
  - Prioritize community impact over design intent
  - View partnership as ongoing collaboration
  - Frame designers as facilitators not deciders
  - Value stakeholders' lived experiences
  - Share design knowledge with communities
  - Work toward community-led, sustainable outcomes
  - Reconnect communities rather than exploit them
  - Designers should understand a communities existing solutions before building new ones

WHAT DOES ANY OF THIS MEAN FOR EDUCATIONAL PL DESIGN?

- I worked with my colleague **R. Ben Shapiro** and my doctoral students **Jayne Everson** and **Megumi Kivuva** to translate these ideas of justice and our joint lived experience teaching computing into **design requirements** that we think best address injustices in current PL design for education.

7 JUSTICE-CENTERED REQUIREMENTS FOR EDUCATIONAL PROGRAMMING LANGUAGES

REQUIREMENTS

OUR APPROACH

- Costanza-Chock's community design principles were our starting point.
- From there, we examined the intersections between those principles, the spectrum of marginalization in education mapped by education justice researchers, and the design choices inherent to educational PL.
- This led to **7 design requirements** for educational PL. Meeting them means meeting the the many principles of justice we just discussed.

ALTCODE — A TRAGICOMIC MNEMONIC

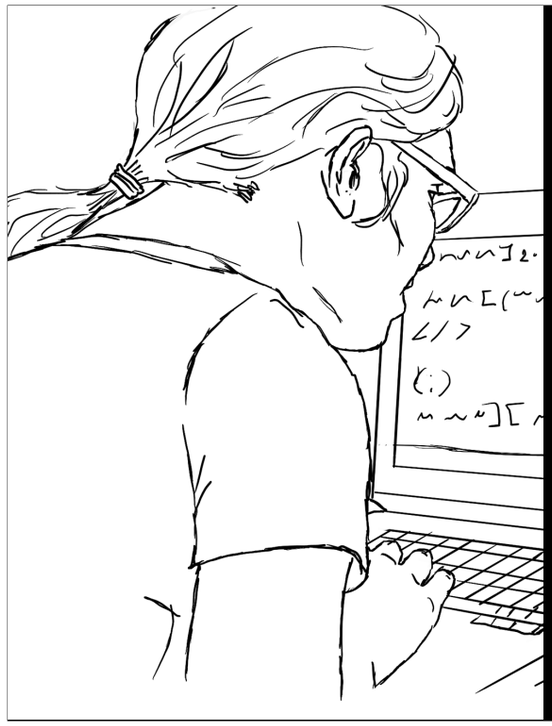
- **A**ccessible – everyone, all abilities
  - **L**iberatory – seeing computing for what it is
    - **T**ransparent – comprehensible, inspectable
      - **C**ultural – centering learners' communities, values, languages
        - **O**btainable – free and feasible to access and use
          - **D**emocratic – shaped by everyone, rather than centralized teams
            - **E**nduring – lasting and sustainable, as long as it is needed

CAVEATS

- There are 7, but that is not a magic number
- We don't claim this is the *only* "right" notion of justice – conceptions of justice evolve over time, and we don't represent all voices
- We *do* claim that if these requirements were met, there would be *many* more people globally who would be able to learn what programming languages are, how to use them, and possibly use them for problems in their community that no big tech company ever would.

EXAMPLES

- Throughout, I'll critique and praise various systems for their strengths and weakness
- I'll also include examples from my own lab's work on **Wordplay**, our attempt at making one example of a justice-centered educational programming language. Not because Wordplay is perfect or best, but just because it tries things others haven't, in pursuit of justice.



SUPPORT ALL ABILITIES

ACCESSIBLE

ACCESSIBLE

THE REQUIREMENT

- *Learners and teachers must be able to use the full functionality of an educational programming language with **whatever input** they can provide and **whatever output** they can perceive and comprehend*
- In practice, this means:
  - Not just mice and keyboards, but speech, Braille keyboards and displays, switches, gaze
  - Not just perceptual and motorphysical, but also diversity in **reading** abilities, **learning**, **sensory processing**, and more.

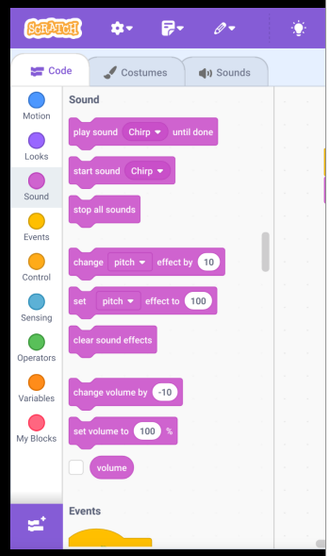
WHY?

- **Disability justice**: all people deserve the right to participate in our computational worlds, independent of what abilities they were born with, lost, gained.
- The world should be designed in a way that accounts for diversity in abilities, not in a way that privileges one set of abilities over others. That is not the world that we have.

ACCESSIBLE

BAD: SCRATCH

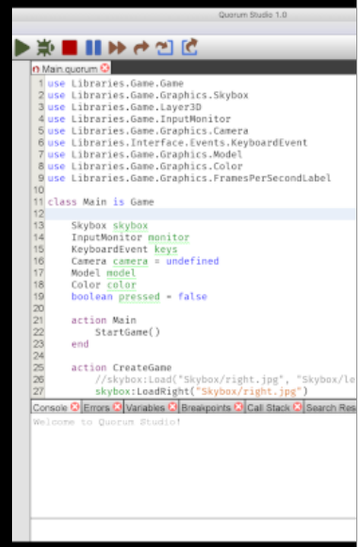
- **Scratch** requires the use of a **pointer** (mouse or touch).
- This excludes anyone who cannot use such an input device; it's success at popularizing the structured code editors of the 1980's, and the drag and drop paradigm of Alice of the 2000's, has meant a proliferation of "block-based languages" that blind learners cannot use, that learners with motor tremors cannot use, that quadriplegic learners cannot use.
- Advocacy to the Scratch team has led to little change in Scratch's accessibility, despite multiple opportunities during rewrites and redesigns over the past 20 years.



ACCESSIBLE

BETTER: QUORUM

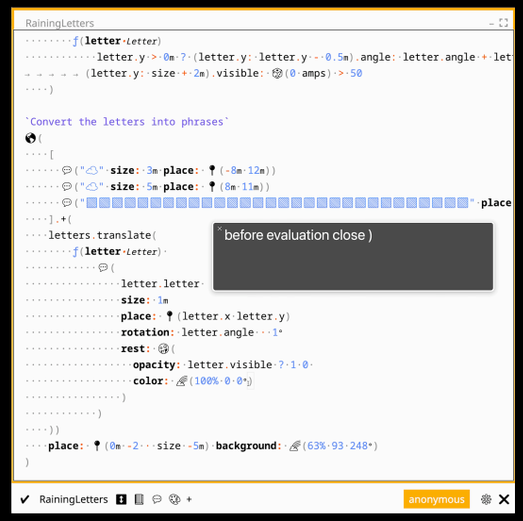
- Quorum's language is designed to be highly screen readable for learners who are blind or dyslexic, and rely on screen readers.
- It also offers screen readable output of 2D and 3D graphics.
- It has been widely adopted in schools for the blind as it is the only screen readable language, IDE, and platform that works and isn't designed for professional developers.



ACCESSIBILITY

WORDPLAY: ALL ABILITIES

- A multi-modal, WCAG compliant editor that supports text editing, block editing, menu editing
- Capacity for future design and development of speech editing.
- Fully accessible, WCAG-compliant program output that comes for free.



OPEN QUESTIONS

- Few PL are designed *with* learners with disabilities around the things they might want to make
  - What would a gaze, sound and movement-based IDE for making purely gaze, sound, and movement-based apps be like?
- How can code editors seamlessly integrate speech and audio feedback?
- How can program output of all kinds be made accessible?
- How can PL be designed to make it easier to make software itself more accessible?

FOSTER CRITICAL CONSCIOUSNESS

LIBERATORY

THE REQUIREMENT

- *Educational PL 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.*
- In practice, this means:
  - Centering the reality that computing is both amazing and powerful, but also kills, harms, marginalizes, and disempowers.
  - Making space in PL design, tools, tutorials and communities for the inherently political nature of computing.

WHY?

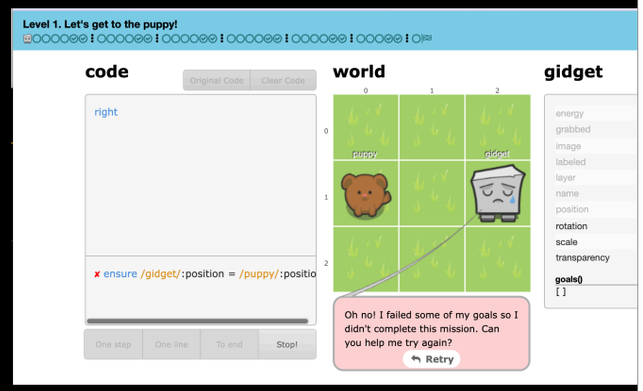
- **Critical consciousness.** To have a just world, everyone must understand how and why it is unjust, so we can fix it together.
- That includes the **computational** world, and programming languages are the media that shapes the computational world.

BAD: CODE COMBAT

- A for profit platform that centers war, violence, "*the feeling of wizardly power at their fingertips by using typed code*", and learners as factory workers producing more than "*1 billion lines of code*"
- Erases the reality that code is literally a tool of war, used to more efficiently kill people at scale, to silence resistance to dictators, etc.

BETTER: GIDGET

- It's not the most political of PL, but it does frame robots and computers as ***fallible, ignorant, but reliable tools***
- This framing is used throughout the game to show learners that machine intelligence is limited and largely stems from human intelligence, demystifying code as "magic".

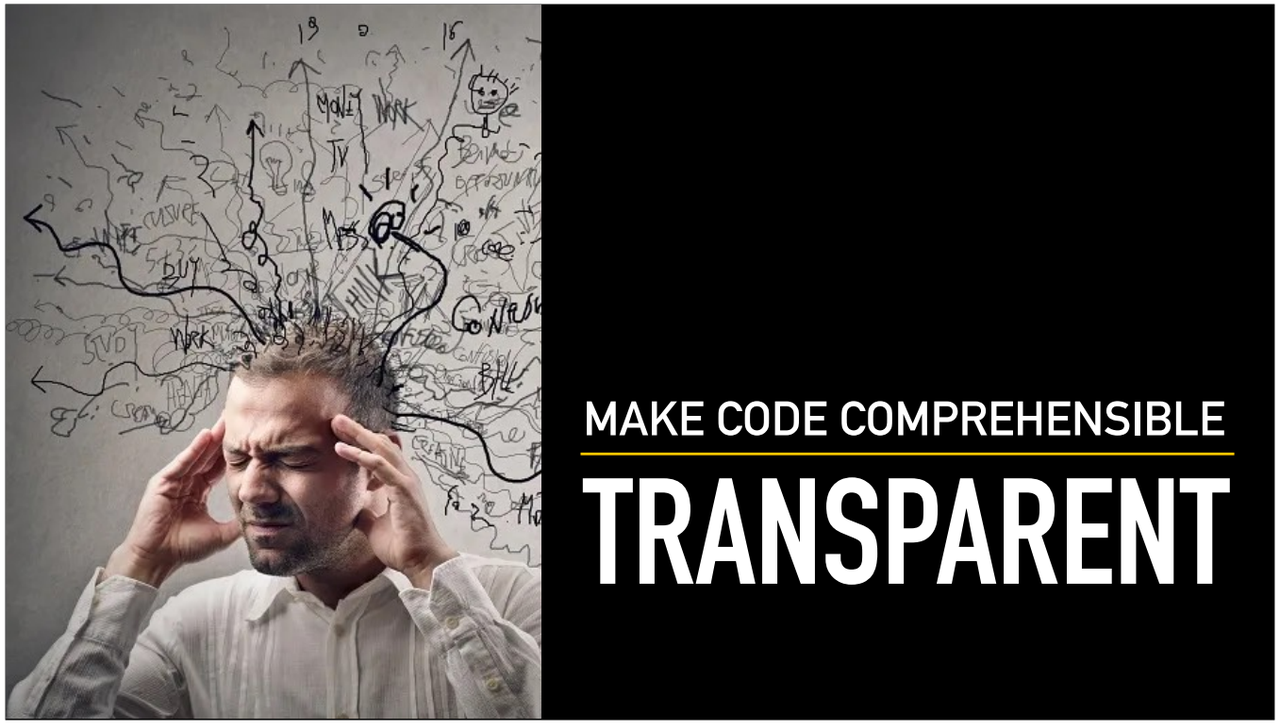


WORDPLAY: LIBERATORY

- Language constructs are anthropomorphized with personalities and relationships with each other than center the limited and narrow views with which they conceive the world.
- Learners are positioned as the only ones of overcoming these limitations, by understanding the nuances of human experience fully.

OPEN QUESTIONS

- Can programming language syntax and semantics be sociopolitical? How?
- What are the opportunities and limits of PL themselves promoting learners' critical consciousness about the good and bad of computing in society?
- How might liberatory PL be resisted by schools, governments, and parents who do not want youth to know about computing's dark side? Are there ways that PL can be subversively political?



MAKE CODE COMPREHENSIBLE

TRANSPARENT

TRANSPARENT

THE REQUIREMENT

- *To foster youth agency via program comprehension, program execution must be navigable in both directions and at multiple levels of granularity.*
- This requirement is essential to **agency**: learners must feel they understand and have control over program behavior, rather than it controlling them.
- In practice, this means:
  - Flexible, accessible control over the speed and direction of a program's execution
  - Explanations of program execution that enable youth to understand what programs do, how they do them, demystifying them

Without this, too often youth view computing as something out of their control, for others to define. And that not only do youth not grow up to build or influence what is made with computing, but they often defer politically to those who can as the experts.

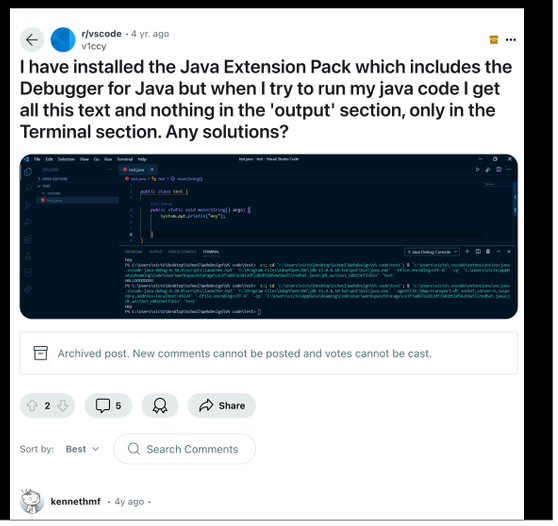
WHY?

- One cannot critique, control, or reimagine something if one does not know what it is or how it words.
- Code has for too long been incomprehensible, enriching and empowering computer scientists and software developers at everyone else's expense.
- Centering comprehensibility, and transparency of software behavior more broadly, is central to **agency**.

TRANSPARENT

BAD: NEARLY ALL PROFESSIONAL PROGRAMMING LANGUAGES

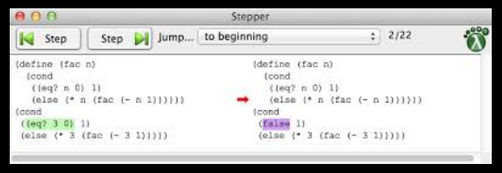
- Everything except for print statement requires complex configuration, poor control over execution, no reversibility.
- This poor support for transparency of execution means learners who try to comprehend programs in these languages struggle far more to understand what code is doing.



TRANSPARENT

BETTER: RACKET + DR. RACKET

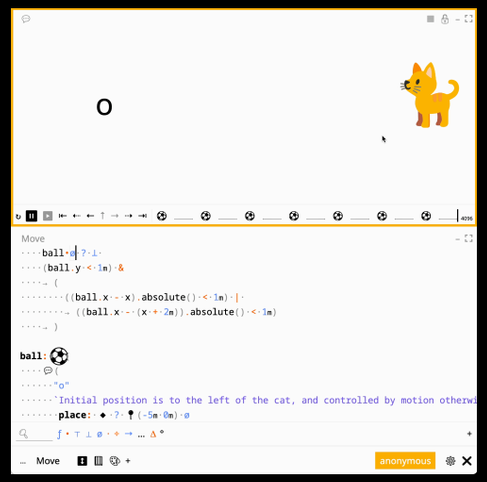
- Racket offers a nice reversible stepper, allowing learners to go forward and backward through an expression's evaluation, using a "rewriting" metaphor
- In addition to being reversible, this is more granular than line-by-line stepping, giving precise visibility into program behavior.



TRANSPARENT

WORDPLAY: REVERSIBLE, GRANULAR

- In Wordplay, programs can be run forward and backwards, infinitely and instantaneously
- Program evaluation can be stepped at an an extremely fine granularity, giving localized, accessible, explanations of every step in all supported languages.



TRANSPARENT

OPEN QUESTIONS

- How can *all* EPL support highly flexible, reversible, granular inspectability of program evaluation?
- How might LLMs be used to explain program execution for different literacy levels, in different natural languages, across different cultures?
- How can technical transparency support liberatory, critical learning about what programs do and why?


EMBRACE ALL LANGUAGES,
CULTURES, AND VALUES
CULTURAL

CULTURAL

THE REQUIREMENT

- *EPL must be culturally responsive and sustaining in how they are designed, explained, and framed, enabling identity-inclusive pedagogy.*
- In practice, this means:
  - Supporting multilingual learners, using language flexibly, not just English
  - Drawing upon many cultures to describe and explain concepts in programming, not just Western, white settler cultures
  - Questioning the Western cultural ideas embedded in CS, including binary truth values, discrete math, and rigid categories

Without this, too often youth view computing as something out of their control, for others to define. And that not only do youth not grow up to build or influence what is made with computing, but they often defer politically to those who can as the experts.

CULTURAL

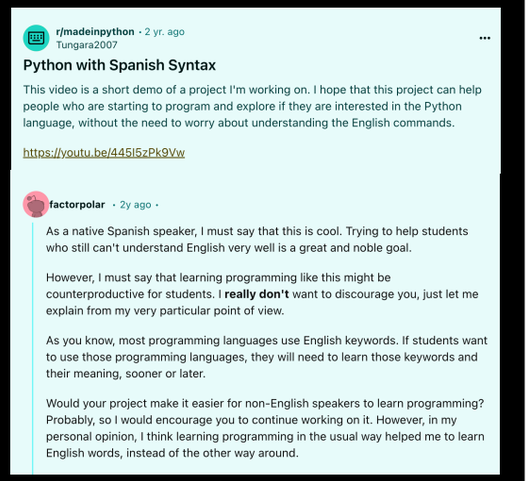
WHY?

- **Decolonization.** Our social worlds are shaped by a history that has centered the culture and language of colonizers, and steadily erased all other culture.
- Humanity deserves to shape the cultural worlds they live in, including restoring those from the past and creating new ones.
- Computer science has not resisted colonization, it has embraced it and amplified it. It has even become a discipline that itself colonizes in intellectual ways.

CULTURAL

BAD: PYTHON

- Syntax is English only, no translations, only a few non-English locales of documentation
- Python 2 had very weak Unicode support, privileging Latin characters only
- Libraries are full of English metaphors (“pickle”, ”nanny”, ”abc”)
- ”Zen of Python” simplicity mantras are in tension with diversity:
  - *“There should be one — and preferably only one — obvious way to do it.”* — obvious to whom?
  - *“Special cases aren’t special enough to break the rules.”* — whose rules and why not?

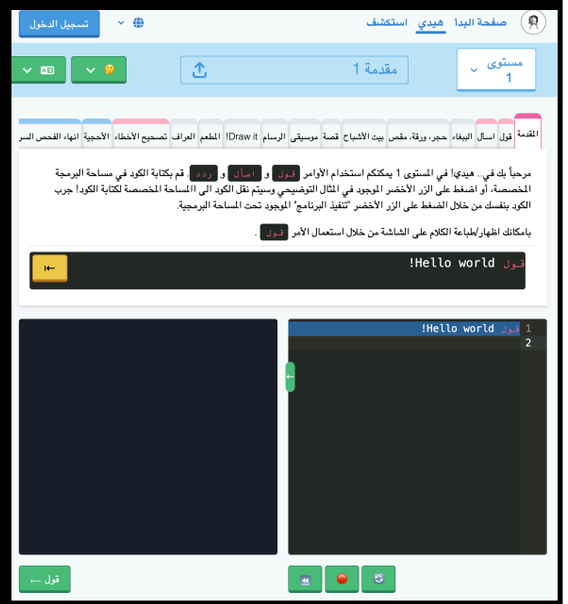


In other words, *the colonizers won, stop trying to decolonize Python, its not realistic.*

CULTURAL

BETTER: HEDY

- 47 different language supported, even localizing the language syntax to mirror different language grammars.
- Doesn’t support mixing languages to support our multilingual world, but gives a glimpse of what a truly global language and platform might look like.



CULTURAL

WORDPLAY: MULTILINGUAL CODE AND OUTPUT

- All names, text, and documentation in programs can have any number of language-tagged aliases
- This allows programs *and output* to be "skinned" and automatically translated into any combination of natural languages

CULTURAL

OPEN QUESTIONS

- How can EPL support multilingual learners, while also supporting their very rational economic motivation to be English fluent?
- How can data structures and algorithms be described with a multiplicity of cultural metaphors, rather than just English, Western ones?
- How might youth be empowered to create their own EPL, with their own ideas about how computation should work?

REQUIRE NO COST

OBTAINABLE

OBTAINABLE

THE REQUIREMENT

- *Learners must be able to access an EPL and its tools and resources independent of their financial means.*
- In practice, this means:
  - EPL must be free
  - EPLs must not require paid access to the internet
  - EPLs must not require purchasing personal devices
  - EPL must assume old hardware, constrained and slow internet access.

- These are essential because if an EPL *does* require any of these things, many youth globally, and even here in the U.S., will be category excluded from participating.
- This is because most youth do not have their own devices, and if they do, they may be smartphones with limited data access, or even smartphones that are time shared with siblings and parents.
- Many have no devices at all, except to those available at school or at public libraries.
- Designing for these constraints is essential, as efforts to bridge these digital divides are regularly obstructed by the cost of bridging inequities.

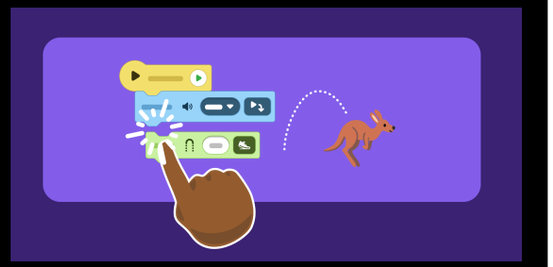
WHY?

- **Economic justice**. People's ability to participate in the world should not be shaped by the economic conditions in which they are born, or the opportunities shaped by the systems of oppression that surround them.
- Computer science has broadly ignored this right, instead designing for those that can access modern devices and the internet, and leaving everyone else behind, in pursuit of profit.

OBTAINABLE

BAD: OCTOSTUDIO

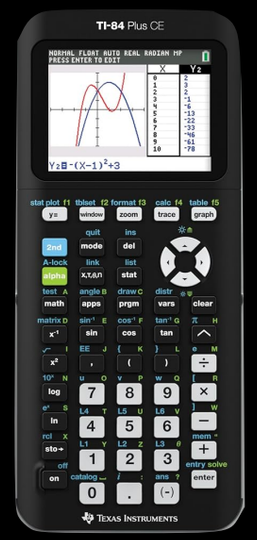
- It is free and only requires internet access to download, which is just.
- But it requires access to an Android 8 or iOS 15 compatible device, the ability to install applications on it, and time to use the device.
- The only youth who might have this access are those either with their own devices, or in schools with enough resources to maintain 1:1 device access.



OBTAINABLE

BETTER: TI GRAPHING CALCULATORS

- Low cost, and most schools already own them for math education, and have existing subsidies.
- Portable, battery powered, requires no internet access, and has a simple PL with access to a variety of sensors (speakers, LEDs).
- Problematic in how TI has a near monopoly over this market, accruing massive profit margins, limiting innovation.



OBTAINABLE

WORDPLAY: ANY BROWSER, ANY DEVICE

- Wordplay is free, on the web, and does not require an active internet connection
- Its footprint is tiny, as text, emojis, and programs require only minimal device storage

OPEN QUESTIONS

- How can EPLs be financed to sustain an ecosystem of hardware and software without exploiting youth and schools for profit?
- How can we reconcile a need for a multiplicity of platforms to meet a diversity of learner needs with the limited capacity to sustain platforms?
- If we embrace EPLs that aren't obtainable, how can we sustainably subsidize access to EPLs to make them obtainable when school funding continues to decay?

CENTER POWER AT THE MARGINS

DEMOCRATIC

DEMOCRATIC

THE REQUIREMENT

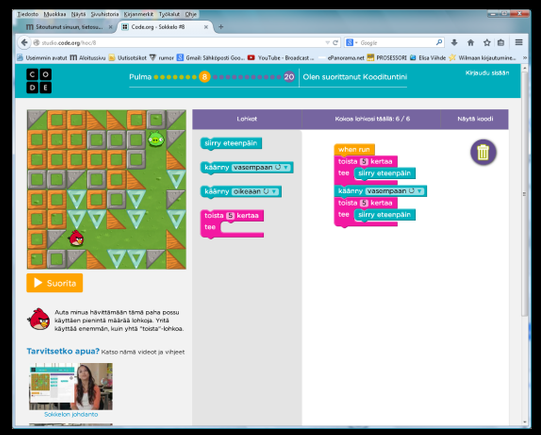
- *EPLs must be governed by and accountable to learners and their communities of support, especially those marginalized in computing and society more broadly.*
- In practice, this means:
  - EPL must be open source
  - EPL designers must give up the power to design to teachers and students
  - They must have community processes to engage, gain power, and influence design
  - Design processes must be organized to center community needs, not other goals, like research, profit, or innovation

WHY?

- The power to shape programmable media should be one that **everyone** has, as the media is used to shape what rights and opportunities everyone has.
- In other words, programming language creators have no right to control the language unilaterally without the voices of those who are impacted by them, directly, or indirectly.

BAD: CODE.ORG STUDIO

- Open source with contributors guidelines, with advisory boards to shape product priorities
- Unfortunately, design authority is centralized in code.org's design and engineering staff, not in the youth or teachers that they serve



DEMOCRATIC

BETTER: PROCESSING.ORG

- Open source, with ample community contributions and pull requests
- The foundation runs public events that solicit advocacy
- Funds fellowships for teachers to explore and shape the platform
- Partners with advocacy organizations at the margins of computing
- Directly engages communities and community leaders to shape priorities

DEMOCRATIC

WORDPLAY: STUDENT- AND TEACHER-LED

- We run a quarterly design studio with middle, high, and college students and teachers to contribute design, development, localization, community organizing, and governance, to the open source project
- We've hosted a youth and teacher advisory council to inform critical design and governance choices, guiding the project priorities

OPEN QUESTIONS

- How to sustain the creation and support of communities, especially with low resource schools and families?
- How can we manage conflict in communities with different needs, and who should hold power to resolve these conflicts?
- How can EPL remain *redesignable* in response to evolving needs in a community, when they are often built in such immutable ways?


BUILT TO LAST
ENDURING

ENDURING

THE REQUIREMENT

- *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.*
- In practice, this means:
  - EPL must be sustainable, maintainable, and resilient
  - EPL must also be discardable when they no longer serve justice

ENDURING

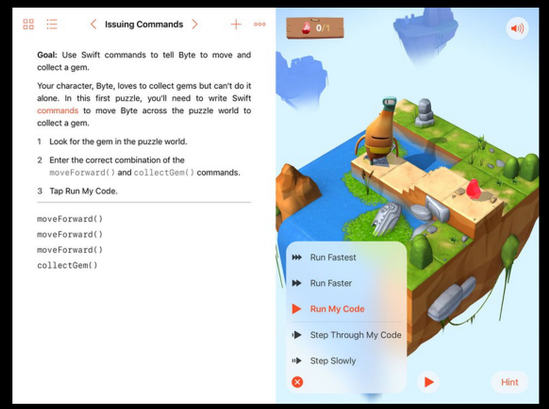
WHY?

- Educational programming languages, in service of public education, or public **infrastructure**.
- Infrastructure should be sustainable and built to last, but also amenable to replacement when it no longer serves the public good.
- Current EPL governance is far from sustainable or replaceable: most are built with very little support, and problematic languages that become popular are hard to replace.

ENDURING

BAD: SWIFT PLAYGROUNDS

- Solid platform and curriculum, billions in funding to sustain it
- No statement of how long it will be supported, limiting adoptability by teachers and districts long term
- No way to stop or mitigate Apple's capitalist efforts to weave it into classrooms, even when such efforts might do harm



ENDURING

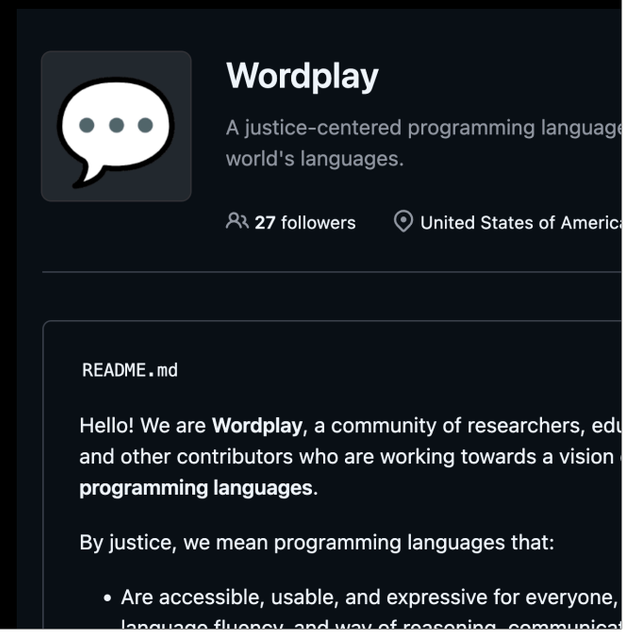
BETTER: SCRATCH

- Large base of funding, now centralized in the Scratch Foundation
- More than 20 years of support, including multiple re-implementations.
- Limited openness means that community's capacity to maintain the platform may be limited if the foundation were to stop supporting the project.

ENDURING

WORDPLAY: BUILT TO LAST

- The platform is fully open source, with extensive onboarding documentation for contributions
- The platform is fully web standards compliant, with minimal cloud-dependencies for persistence
- The platform relies on text, no images, minimizing energy and storage use



ENDURING

OPEN QUESTIONS

- What are justice-centered models for sustaining EPL technically, socially, and politically, to promote resilience?
- How can governance be organized to give teachers and youth power to retire EPL that are doing more harm than good?

WHAT´S NEXT?

THE KEY POINT

- Educational PL 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
- ALTCODE requirements are one possible way to operationalize justice for EPL design and they raise many technical, social, and political grand challenges for future work.

WHAT'S NEXT?

THIS IS HARD

- Some challenges are **technical**: making EPL transparent is a hard engineering challenge. Must we trade performance for transparency?
- Some challenges are **social**: how can we organize intergenerational, multicultural, ability diverse groups to design EPL together?
- Some challenges are **political**: who will pay for the creation of justice-centered EPL, in a world that increasingly rejects equity, doubling down on racial and patriarchal capitalism?

WHAT'S NEXT?

JUSTICE-CENTERED EDUCATIONAL PL ARE HARDLY ENOUGH

- We still need:
  - Properly funded public schools
  - A diverse, well-supported CS teaching workforce
  - Accessible classrooms
  - Universal access to devices and the internet
  - Teaching methods that are culturally responsive, sustaining

SOME OF US WILL BUILD, SOME OF US WILL ORGANIZE, AND SOME OF US WILL LEAD.

I HOPE SOME OF YOU WILL FIND A WAY TO JOIN THIS WORK, AND CREATE A COMPUTATIONAL WORLD THAT WORKS FOR EVERYONE.

Accessible
Liberatory
Transparent
Cultural
Obtainable
Democratic
Enduring

JUSTICE-CENTERED EDUCATIONAL PROGRAMMING LANGUAGES

DISCUSSION