Searching for Justice in Programming Language Design

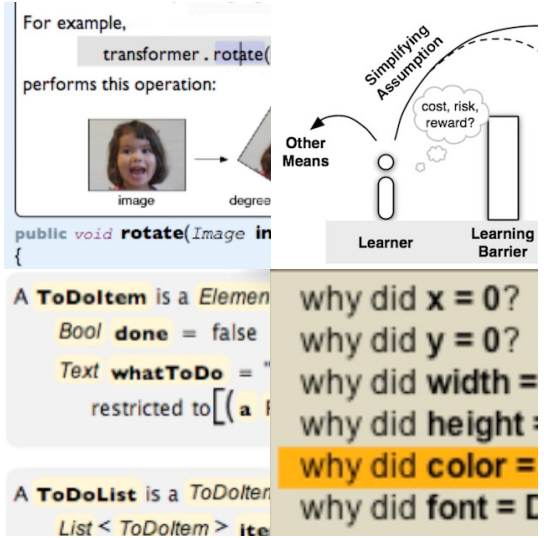
Amy J. Ko, Ph.D.
The Information School
The Paul G. Allen School of Computer Science & Engineering (courtesy)
University of Washington, Seattle

About ACM and the distinguished speakers program

- The Distinguished Speakers Program is made possible by the **Association for Computing Machinery**, the premier global community of computing professionals and students with nearly 100,000 members in more than 170 countries.
- Its **mission** is to help computing professionals to be their best and most creative.
- Its **vision** is a world where computing makes a positive social impact throughout the world.
- I am proud to be an ACM Member

A bit about my scholarly start…

I began my career inventing **interactive developer tools**, trying to make programming more productive with new kinds of editors, debuggers, and verification tools at the intersection of HCI and Software Engineering. Building things was fun, and there were millions of developers who I tried to help have better experiences in engineering software.



But I set this work aside for good reasons…

A decade of computing education research

Back in ~2010, I saw a world that was increasingly computational, but also increasingly **complex**, **centralized**, and **colonial**, “eating” the world in both powerful and oppressive ways.

I wanted to help create a different world where a more **critical computing literacy** was equitably available to everyone.

Public education is the biggest lever we have, and so I joined the global CS for All movement to help broaden participation, dismantle barriers, and address inequities in CS education.

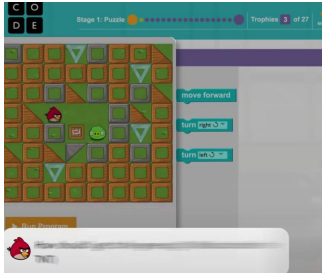
From learning to justice

As my lab's work progressed, my perspective shifted from the **neoliberal** goals that dominate computing — train students for FAANG jobs — to **justice**:

- Our world is built to reinforce what Patricia Hill Collins called the **matrix of oppression** — the social systems that entrench power hierarchies by erasing intersecting identities.
- Computing and computing education reinforces this matrix, framing computing and learning as tool of corporate profit.
- Justice, in my view, is **dismantling** this matrix, and creating new equitable systems work for everyone, instead of just those with dominant identities.

We've worked toward justice on many fronts

A justice-focused K-12 **teacher education** program that frames CS concepts sociotechnically (w/ studies of how this shapes teacher identity)



Studies of how **bias** hides in CS assessments, creating structural forms of gatekeeping of CS literacies.

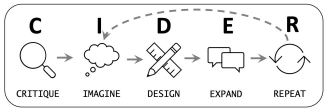
Books that prepare students and teachers to see computing through the lens of justice, and discover their own "limiting situations"

"There's ACM guidelines that sort of tell you what you should be covering... I've not looked at those guidelines in a while, but I doubt that [it is]."

Studies of **norms** and **fears** that deter CS teachers from teaching about diversity, equity, inclusion, accessibility, and ethics.

```
object sapling()
function removeGoop()
    goto /goop/
    remove /goop/
    create kitten()
removeGoop()
```

Teaching methods that improve learning and self-efficacy in programming by resisting authoritative framings of compilers.

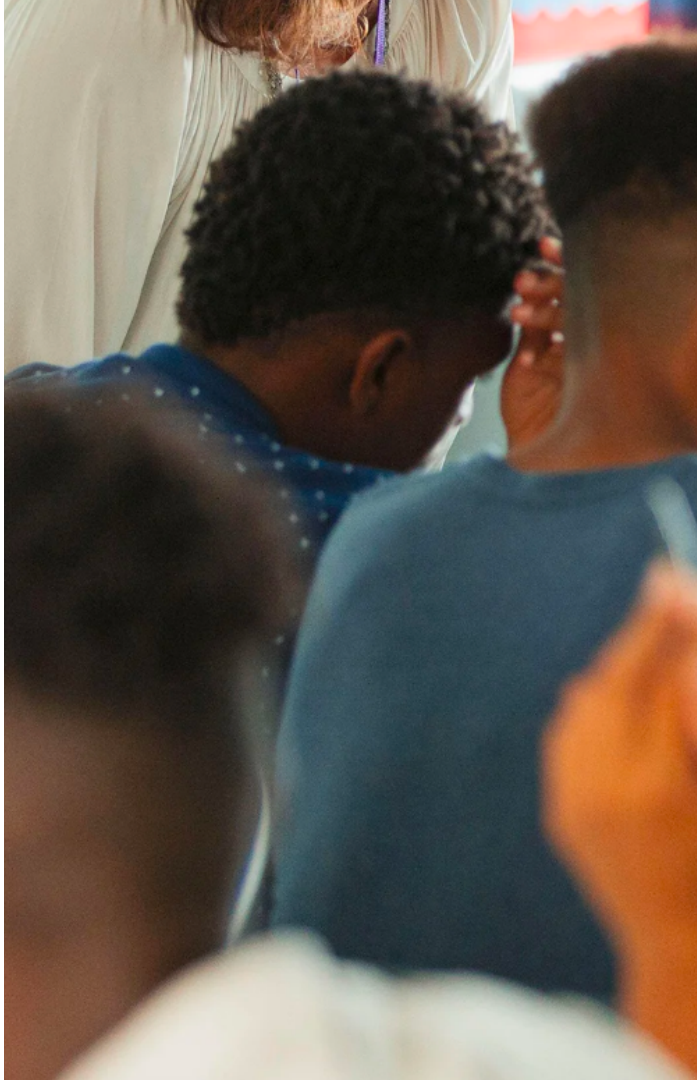


Teaching methods for surfacing **assumptions** about identity and ability in algorithms and data.

My lab and I take these discoveries into the world, shaping state and federal policy, curricula, learning technologies, and teacher education pathways. Our work has reached millions of youth through **curriculum**, **policy**, and **learning technologies**.

Despite all of this work, however, the **tools** of computing still stand in the way critical computing literacy.

I was in a south Seattle math classroom last Spring. Most were refugees, most were learning English, and many had disabilities, including dyslexia, blindness, low vision, and motor impairments. The 25 kids spoke 17 different languages.



(stock photo)

The teacher had completed some equity-centered CS professional development and wanted to integrate CS in some of her algebra lessons in culturally sustaining ways. She had many questions...

(stock photo)

What platforms would work for my **blind**, **low vision**, and/or **motor impaired** students?

What would work for the **17 different languages** in my classroom, and for English-language learners?

What platforms can center my student own languages and cultures, instead of **Western**, **American** ones?

What platforms seriously engage **math and computing**, but make aren't boring?

I had no answer.

This is because most of our educational programming languages and tools are designed with the same set of assumptions...

- Students can read **English**
- Students can **see**
- Students can use a **mouse**
- Students are **interested** in CS
- Students will **persist**

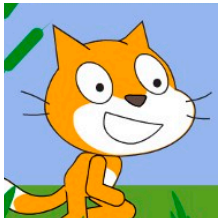
Most of these were not true for her students. And of course, these aren't true for most students in the world. They're really only true for English-speaking, normatively abled youth who fall in love with computing itself.

None of these structural forms **ability**, **culture**, **language**, and **identity** exclusion are surprising.

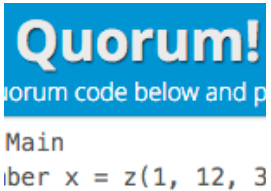
They are the consequence of **ableist**, **colonizing, hegemonic** decisions made by computer scientists from 1960's to today, centering white, Western, ability-normative ideas of who CS is for in our programming languages and tools.

The computing ed community is just starting to make progress on breaking these assumptions.

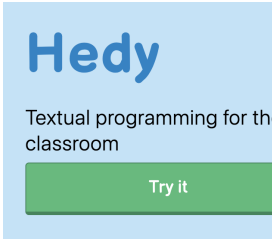
Bootstrap has deep integration with algebra, and focus on some aspects of accessibility, creative expression, and a Spanish curriculum.



Scratch centers expression and is localized in many languages, but requires use of pointing device.



Quorum was designed to be screen readable and keyboard accessible, but assumes English and de-centers student culture, language, and identity.



Hedy embraces dozens of natural languages, even down to the syntax level, but segregates them, and leaves little room for expression.

Where are the creative coding platforms that celebrate the beauty of computing, but also center **disability justice** and **decolonization**?

I decided to humbly make one

- **Sabbatical** gifted me ~1,000 hours over ~15 months to build
- I've approached the work as **art therapy, not design** (because academic leadership, pandemics, and gender transitions are stressful)
- I've had **many** ideas about programming languages that I haven't had time to explore in the past 20 years. This was my chance!
- As a typography nerd, I was excited about the new **Noto font**, which supports nearly all of the language scripts in Unicode.
- I explicitly deferred **evaluation**, and so view the work as generating new questions, not answers.

My aesthetic goals

Global — Celebrate the world's languages without centering English

Accessible — Center ability diversity and neurodiversity, and their tensions

Playful — Embrace the silliness of adolescence, not CS seriousness.

Simple — Ruthlessly avoid complexity

💬 Wordplay

Learn to code by animating words, emojis, and symbols from the world's languages, whatever your abilities.

Wordplay is an innovative programming language 💻 with block and text editing 📝, playful inputs like time 🕐, sound 🎤, webpages 🔗, and physics 🌎, instant time-travel debugging ⏪, and comprehensive documentation 🗂️. All in a growing list of the world's languages 🌐, following the latest web accessibility standards ♿. Share with friends, private groups, or the world ↗. Free forever from the University of Washington Information School ↗.

Learn

global

embracing the
world’s languages

problem: English all the way down

Most programming languages are designed to mimic **English**. English keywords, English grammar, English concepts. They aren't designed to be translated, and have no built-in support for translating their output.

This is not an accident: it is the direct result of winner-takes-all **settler colonialism.**

idea: translation all the way down

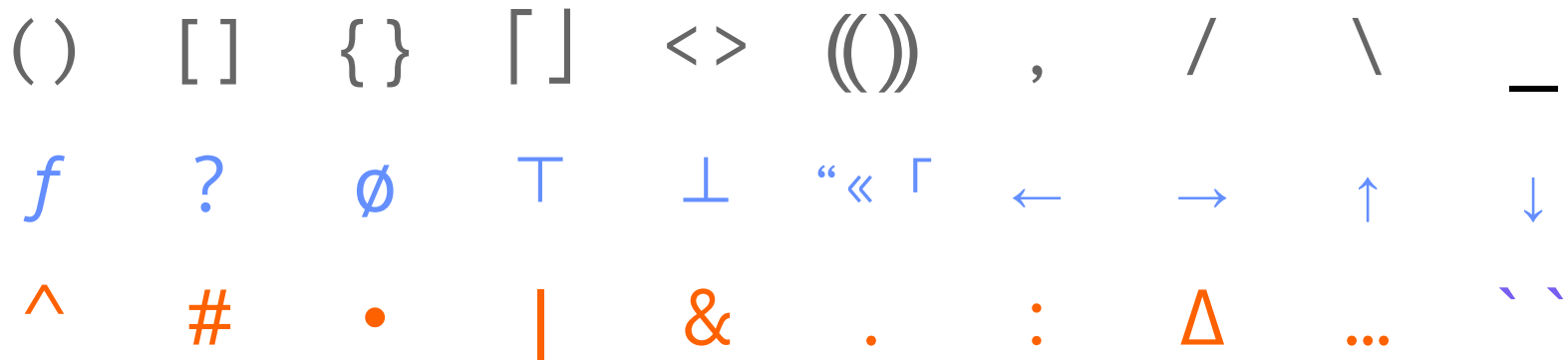
Linguistic justice (e.g., Baker-Bell 2020) might mean that all languages are supported, and none privileged.

Wordplay operationalizes this by:

1. Only using abstract **symbols** for the syntax — no words (e.g., ~~`function`~~, ~~`for`~~)
2. Viewing **names**, documentation, and output as a set of language tagged aliases

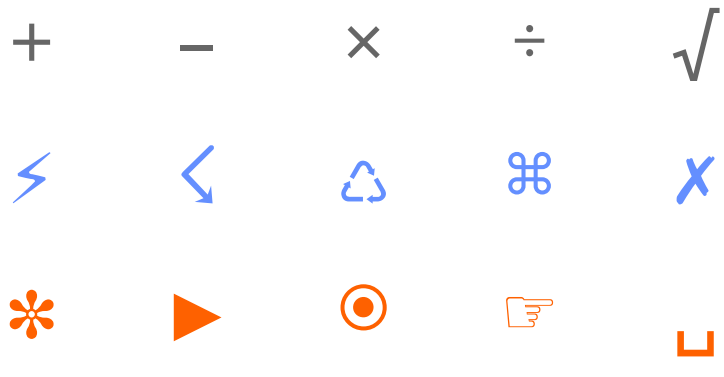
30 symbols/pairs that aspire to be **global**

It's hard to choose symbols that don't have deeply situated culture meaning. e.g., `false` often translates to "lie" or "deception". Choosing symbols (`⊤`, `⊥`) helps avoid cultural assumptions, at the possible expense of clarity.

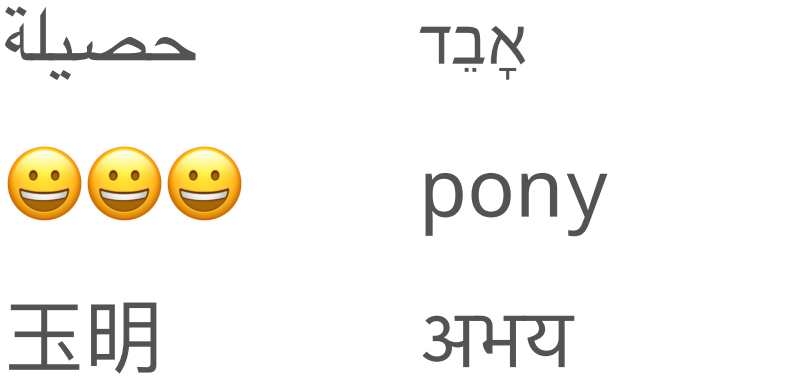


all other symbols are names

infix operators can be any non-emoji character in the **symbols** category, e.g.:



names can be **any** sequence of non-reserved, non-operator characters.



numbers from across the world, intermingled

Arabic, Japanese, Roman, Greek, and more — Wordplay embraces all of the world's number systems and numerals and allows them to be mixed together.

names and documentation are **translations**

start

`The sum of 1 and 2`/*en*

```
sum: 1 + 2
```

data structures are typographically spare, avoiding culturally bound keywords

```
nothing: ø
list: [1 2 3 4 5]
set: {'cat' 'dog' 'pig'}
map: {'Amy': T 'Dr.Ko': T 'yo': ⊥}
•Kitty(name•'' breed•'') (
  ƒ hello() "Hi, my name is \name\ and I'm a \breed "
)
```

a functional grammar

Wordplay blends Smalltalk's love of **objects**, Lisp's love of **parentheses**, APL's love of **symbols**, and functional programming's love of **expressions**, while avoiding natural language mimicry to avoid linguistic hegemony.

```
f · !(n•#) ·
  n·= 1·?
    1
    n···!(n -·1)

!(5)
```

number

120

All of these ideas enable 1) **instant localization** of code and output and the use of 2) **multiple languages** in code.

questions about being global

- What is gained and lost with this **“deep” localization** of a programming system, in learning, teaching, play (e.g., shared language for concepts)?
- What are the opportunities for 1) student **translanguaging**, and 2) teacher facilitation with English-language learners?
- What can be taught about **localization** itself by building concepts of localization directly into a language?
- How do **English learners’** perceptions of CS change when they see CS concepts in their languages instead of English?

Is this justice?

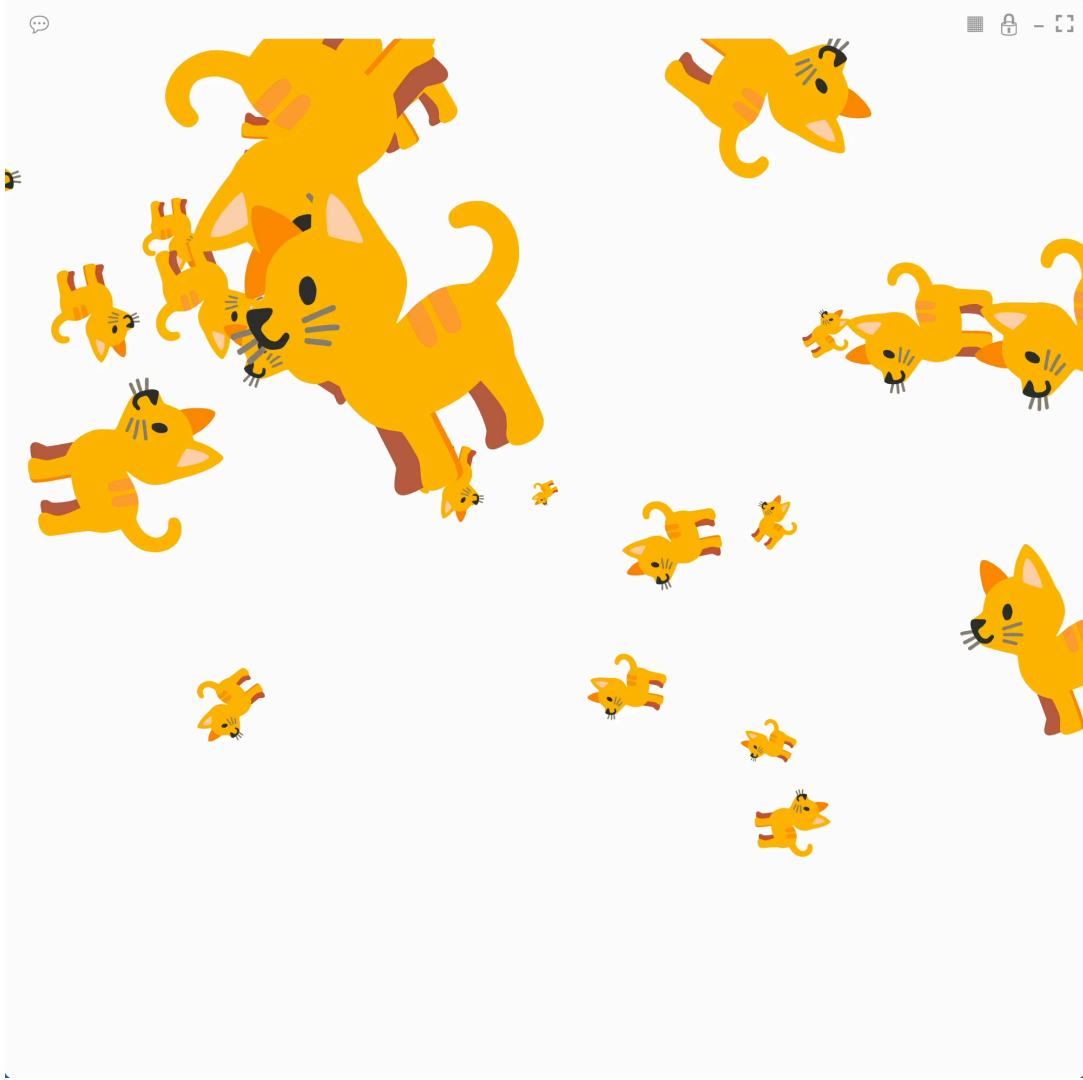
Perhaps in a mundane way. It feels to me like the **least** programming languages could do.

In particular, it leaves a mountain of **translation labor** to do, in the language and documentation, but also in every program.

It leaves intact the broader forces that privilege English and western civilization, including those in the very machine translation tools that might help address these gaps.

playful

centering
silliness



problem: PL indirectly out-groups

Community is often the first thing that learners experience — it's signaled in tooling, documentation, learning materials, and more, and conveys group membership in ways shape who codes and how they do it.

This is not an accident: dominant groups in CS uphold an epistemic hegemony that privileges **western rationality** and rejects **subjectivity**.

idea: computational ideas as social beings

Epistemic justice (Fricker 2007) might mean actively resisting the idea that programming languages and their designers are the sole sources of authority, truth, and objectivity.

Wordplay operationalizes this by anthropomorphizing computing concepts through **lore**, offering building a world in which computing concepts interact, have conflict, and collaborate.

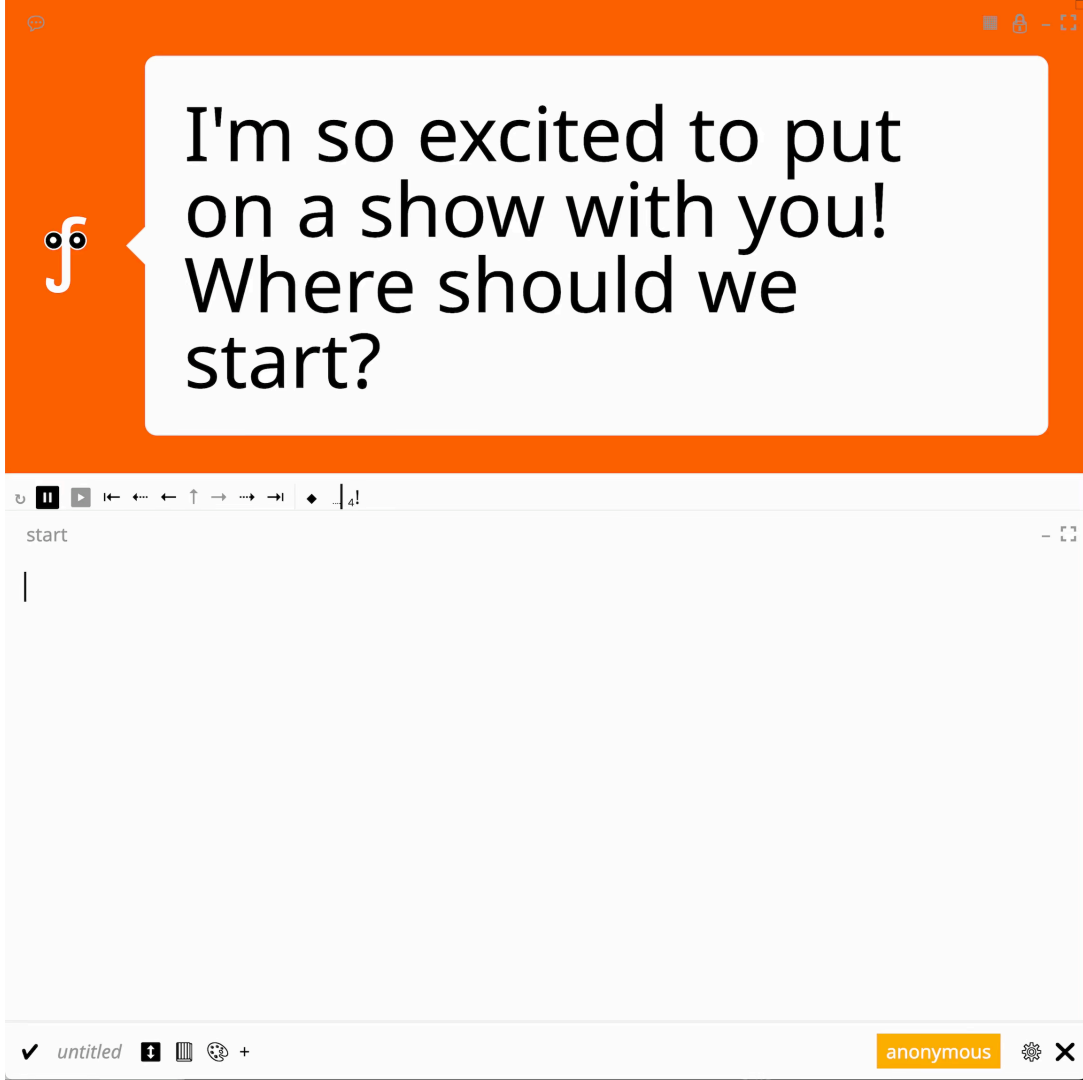
A **community** of characters

- The **verse** is a place with ~150,000 residents, spanning 161 scripts.
- Each resident is a **character,** corresponding to a Unicode code point.
- Characters like to put on elaborate **performances** (programs) in collaboration with **choreographers** (programmers)
- Some characters like to be on **stage** (output), but some like to **choreograph** (code), doing set design, controlling lighting, etc.

All Unicode glyphs (Credit: Ian Albert)

Every character has a **personality** and **positionality**.

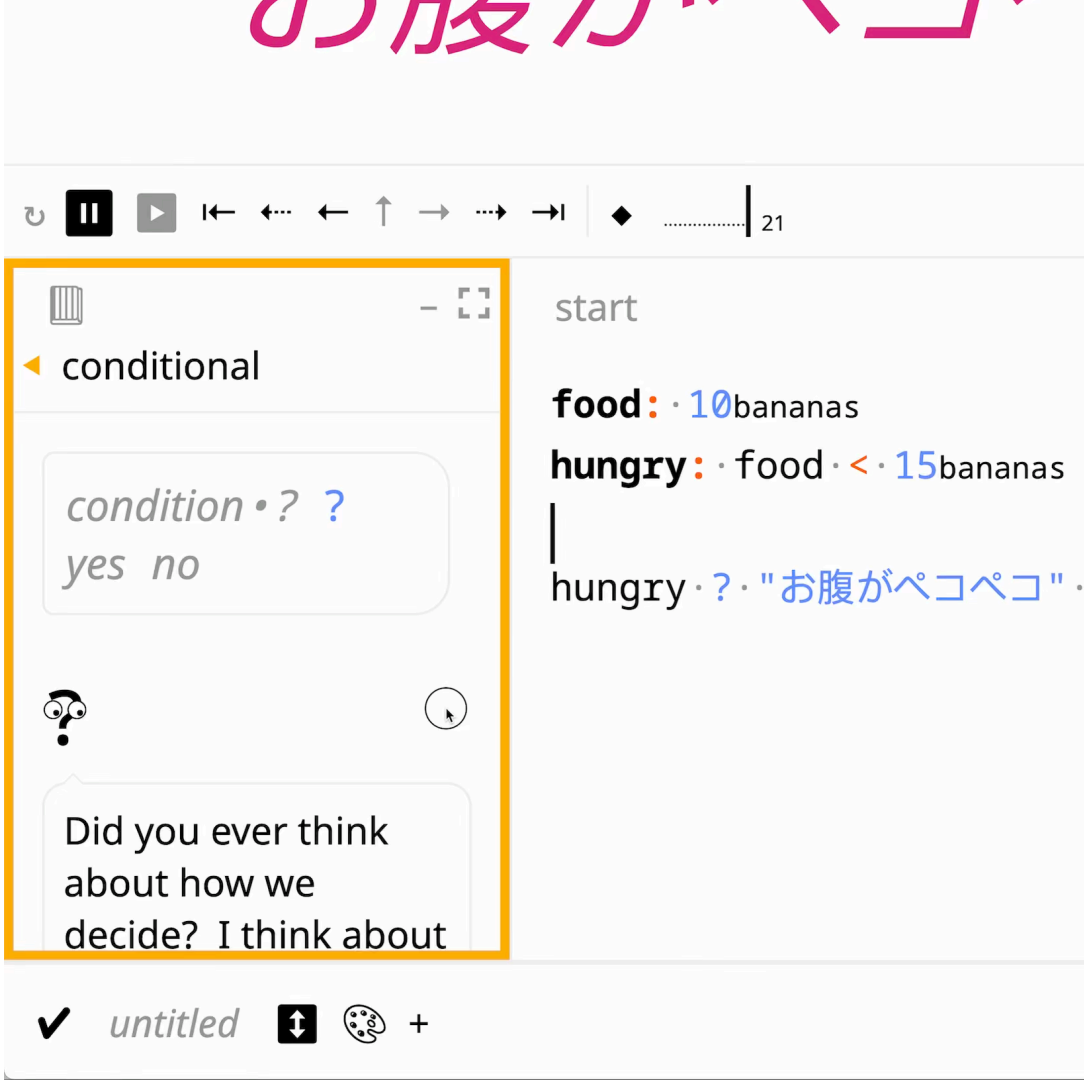
Program nodes, for example are presented by ***f***, who is always excited about planning a performance.



Some characters convey **epistemic struggles** with their computational purpose.

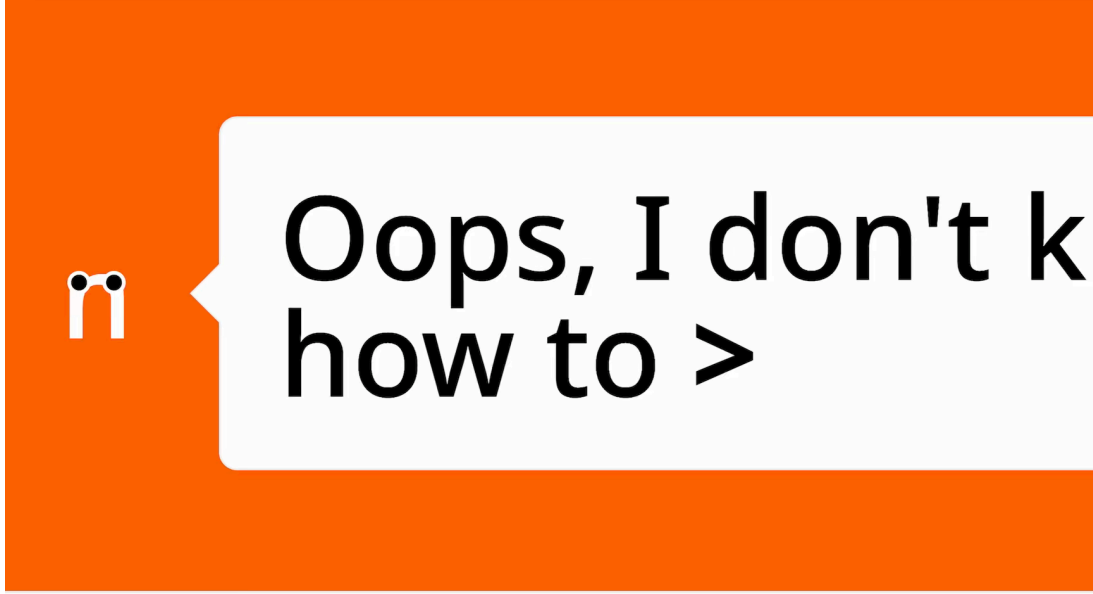
These are conveyed in **diaries** (documentation), where language concepts project their purpose, values, and concerns.

Here, **conditional** (represented by **?**) wrestles existentially with binary decision making and their skepticism of dichotomous truth values.



Wordplay frames “errors” as **conflicts** between characters that need to be resolved before a performance can proceed.

Here, a function definition and a function evaluation have a conflict about the type of an input, and it’s up to the choreographer to resolve it.



start

```
`True if n is between min and max, exclusive`
ƒ·between(n·#·min·#·max·#)·(n·>·min)·&·(n·<·max)

`Let's test this!`
between("1"·0·5)
```

questions about being playful

- What effect does **anthropomorphization** of programming language concepts have on learning, self-efficacy, theory of intelligence?
- How might lore be written to align with different cultural **values** and **ideas**?
- What effect does silliness have on how youth perceive computer science as a discipline?

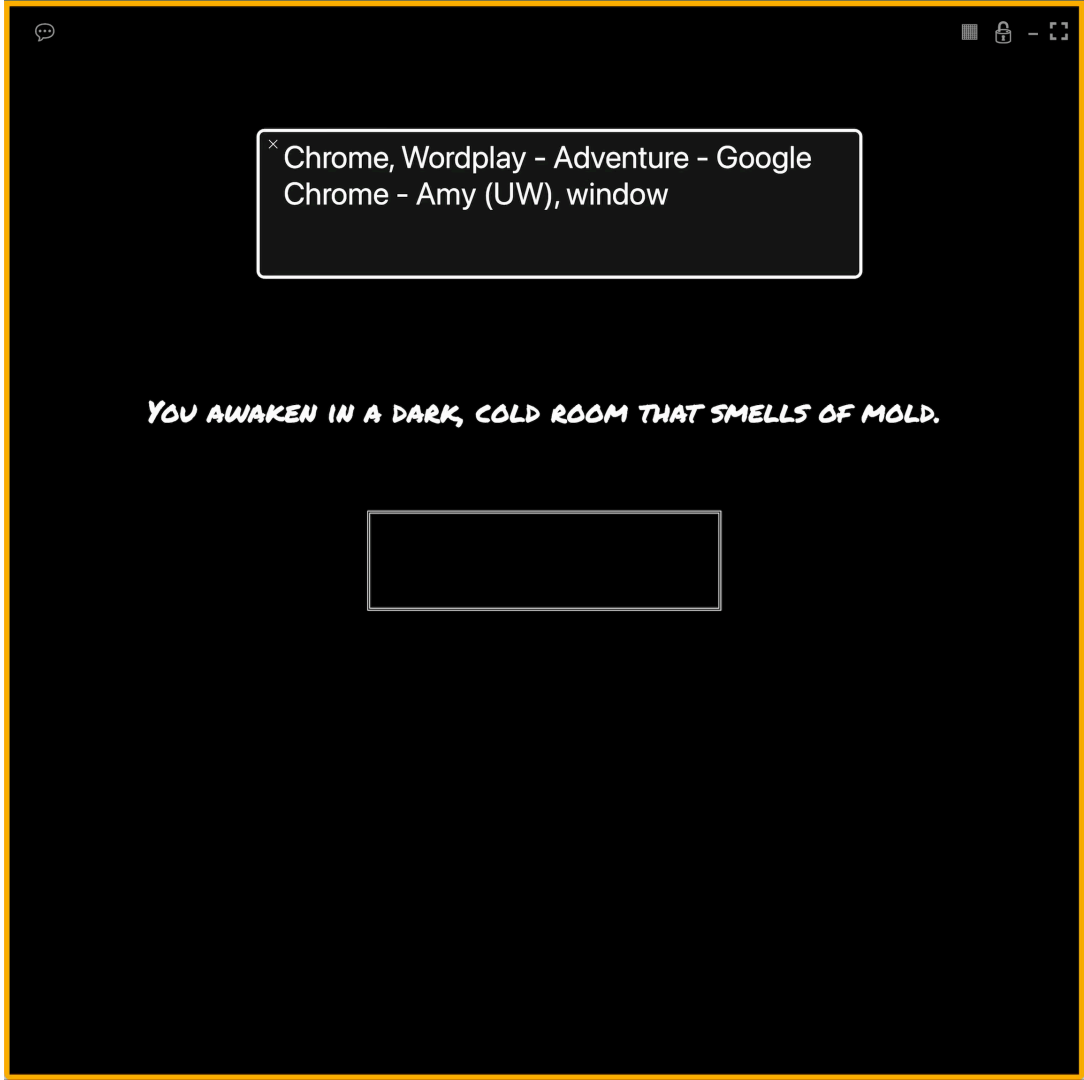
Is this justice?

It is certainly **resistance**. It is one language amongst thousands, and perhaps the only one that explicitly questions the epistemic claims of computing directly inside a computing medium.

But justice might mean **all** programming languages and their communities centering humility about computing and its uses, even advocating for refusal (e.g., not building).

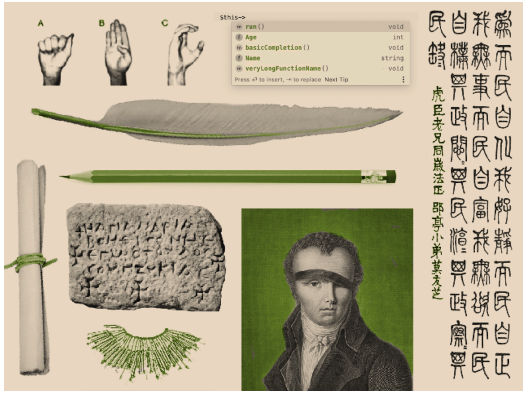
accessible

all abilities, no exceptions



problem: PL stacks are inaccessible

People are immensely diverse in their abilities and cognition, but programming languages tend to work for a narrow band of human ability, forcing mouse or keyboard use, visual output, complex language.



This is not an accident: PL is just one example of the broader **ignorance** and **disregard** for disability in computing and the world, and one that is now self-reinforcing.

idea: multiple representations of code and output

Disability justice means many things (e.g., Berne 2018), but particularly **agency** amidst broader ideas of collective access, interdependence, cross-disability solidarity.

In Wordplay, this might mean **flexibility**: multiple modalities for input, output when reading, writing, and evaluating code, but also control over time, color, and other details typically under the control of a computer, runtime, or designer.

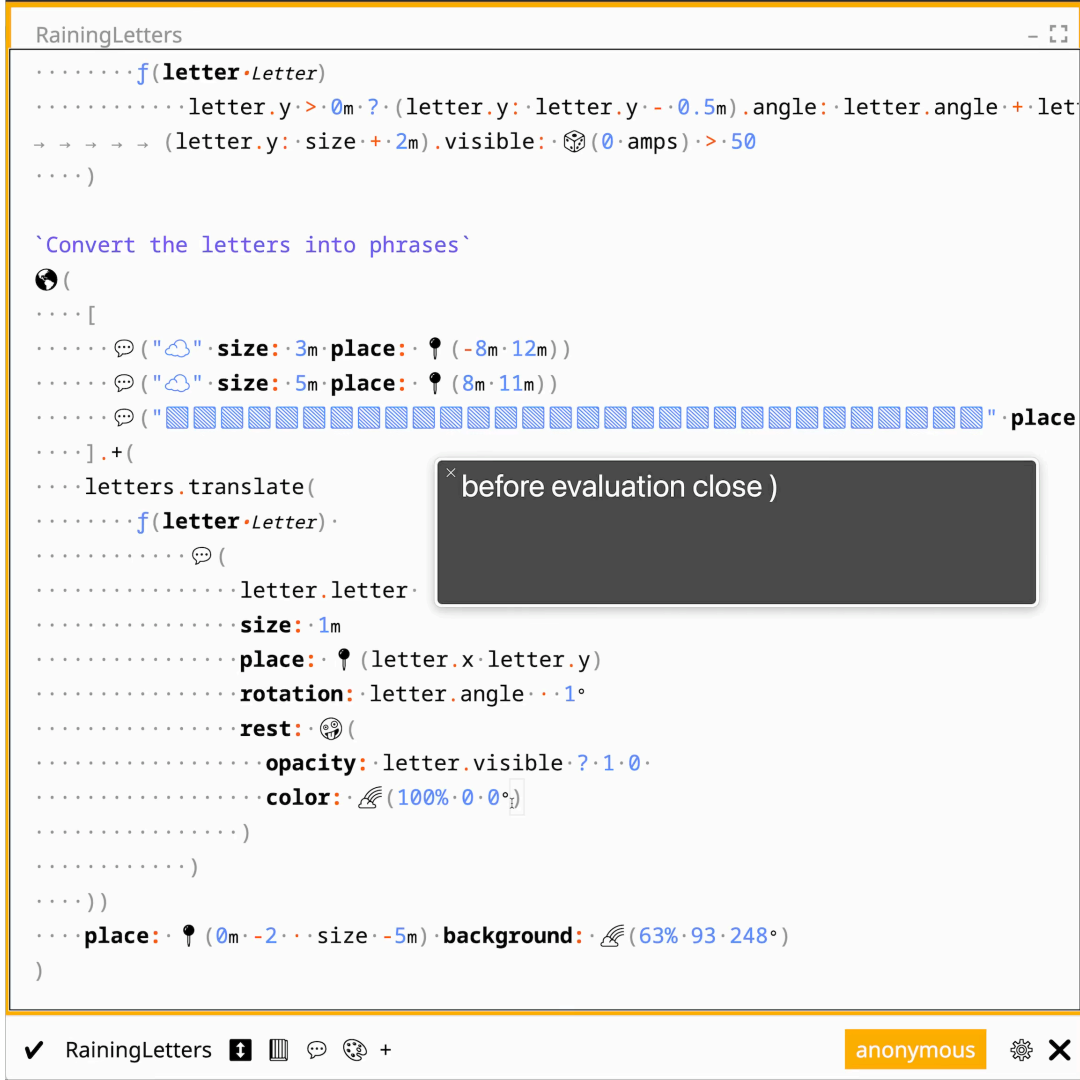
Wordplay offers the world's first **hybrid text and block-based editors**, providing options:

- Quick but error prone typing
- Slow but error preventing drag and

Creators can choose how to edit based on their abilities, knowledge, and risk aversion, without the stigma of **segregation**.

The editor allows for visual and audio navigation of **program structure** via keyboard, climbing the tree, moving to siblings and children.

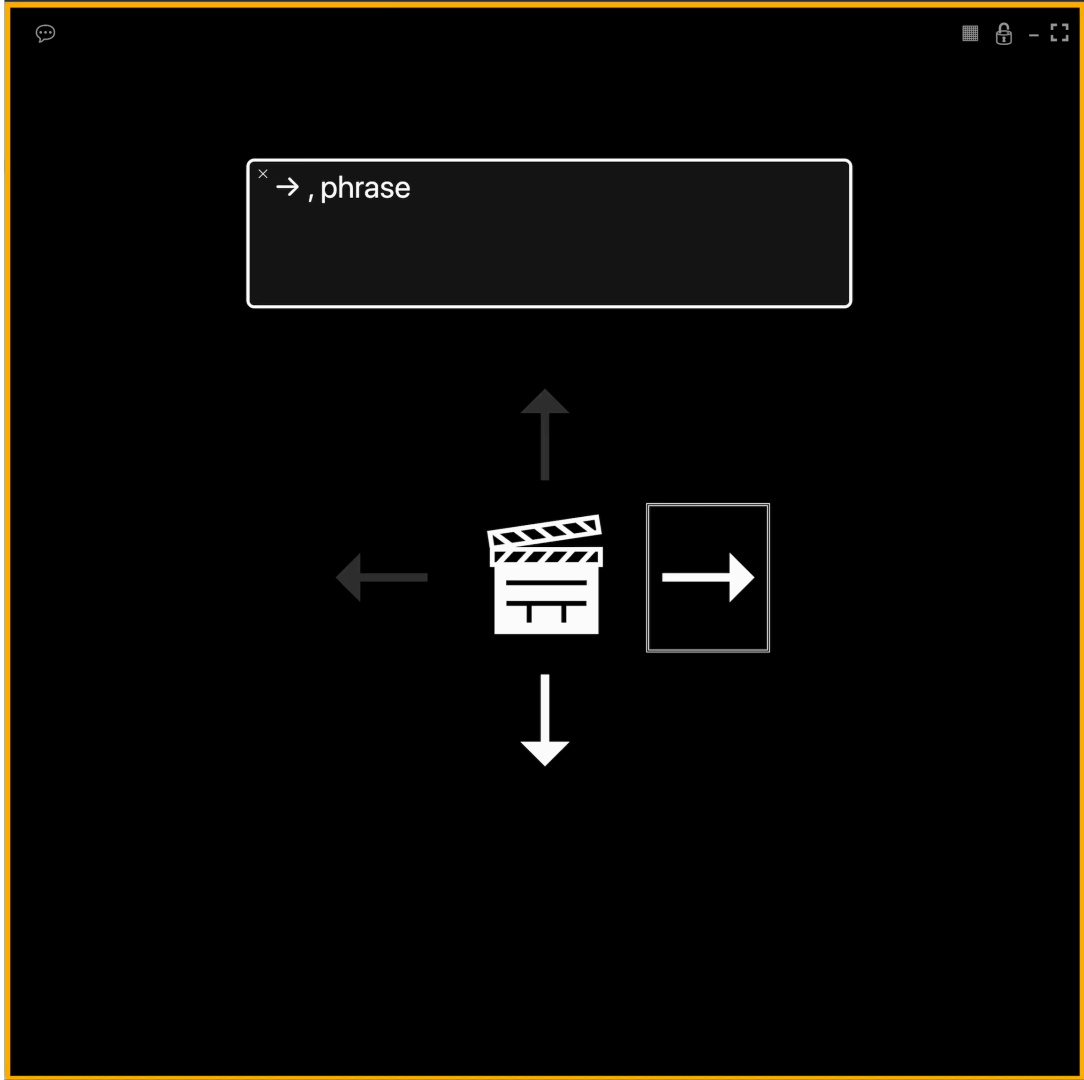
The screen reader reads a **localized description** of each node in this abstract syntax tree instead of reading program text verbatim.



Output is a scene of **phrases** that enter, change, and exit stage.

Phrases are both **visual** and **textual**, as are changes to phrases.

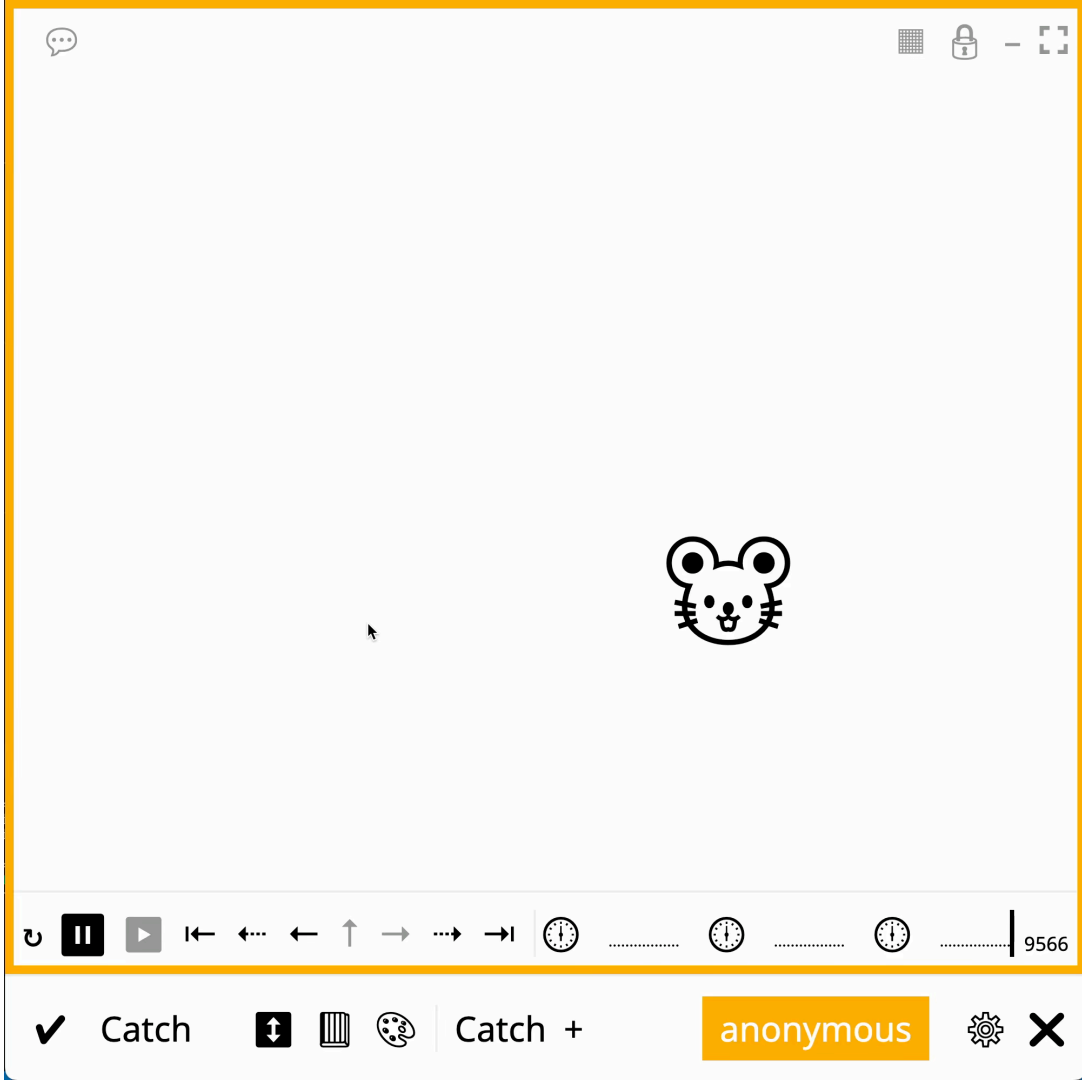
The declarative nature of functional code enables a kind of live captioning.



Timing and animation are **globally configurable** — without requiring program-level support.

Here, a catch-the-mouse game moves a bit too fast, but slowing down time can make the game more tractable.

Turning off animations altogether can address motion sensitivity.



questions about accessibility

- What **tradeoffs** does Wordplay's accessible hybrid editor pose to complexity, scalability, error-proneness?
- What might students learn about **accessible computing** by creating programs that are accessible by default?
- What are **other forms** of input and output are possible with multiple representations? (e.g., speech input, tactile output?)

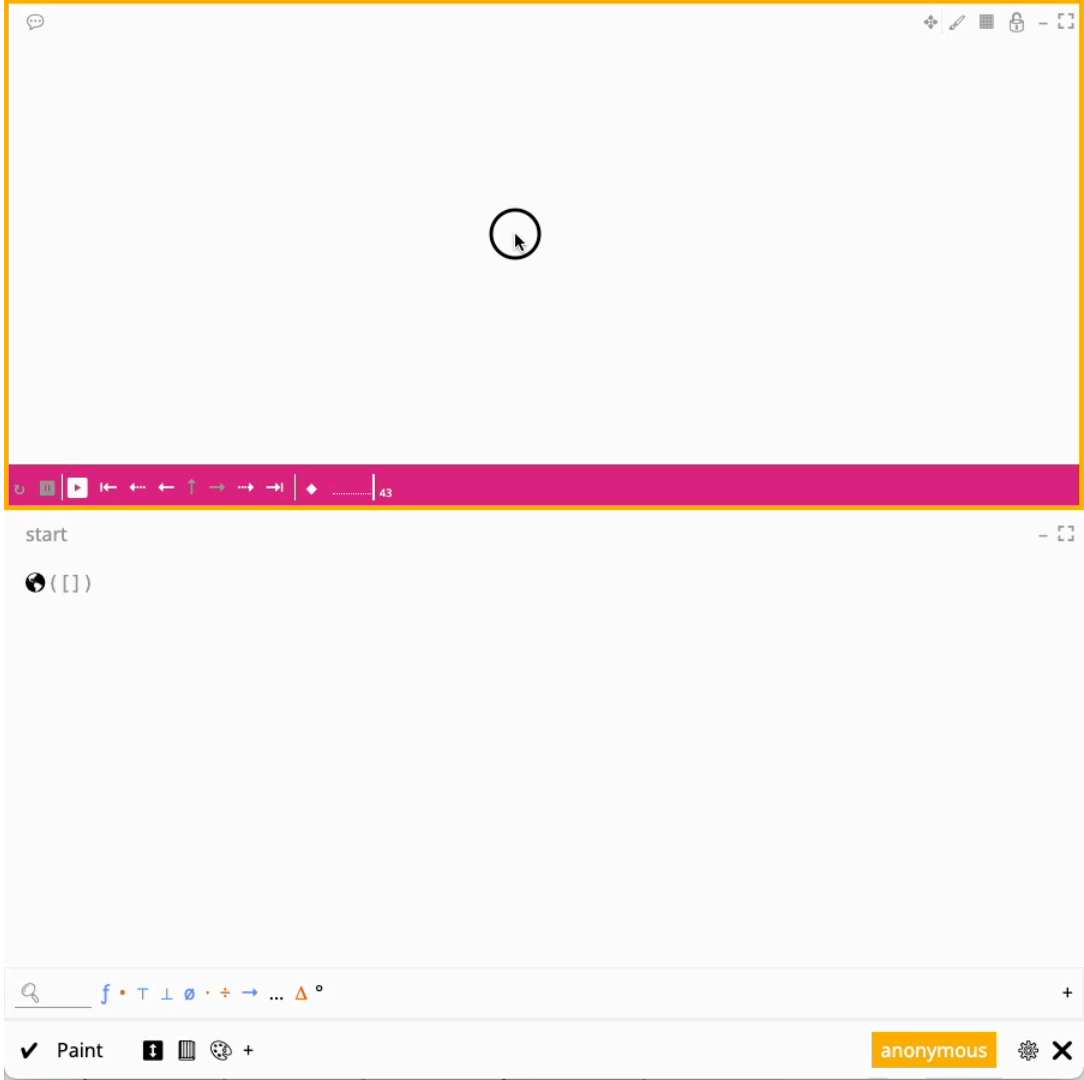
Is this justice?

Only in the most prosaic sense. It is the **bare minimum** of access, opening up input, output, and code to more abilities, and not yet all, and only for this one language.

True disability justice would mean not only having all of these features be standard in all programming languages, but having disabled communities **lead** the design of this platform and its accessibility. We are far from that.

simplicity

reducing
complexity



problem: complexity causes difficulty

Understanding and **debugging** code has always been the central difficulty, and much of this stems from language features such as mutability, but also a lack of tool support, exacerbating language and accessibility barriers.

This is not an accident. Computing has long prized **performance** over **comprehensibility**, as part of a broader project of capitalism, burdening programmer's with cognitive labor to buy speed. Educational programming languages inherit these priorities, placing learners in the same bind, limiting participation in computing.

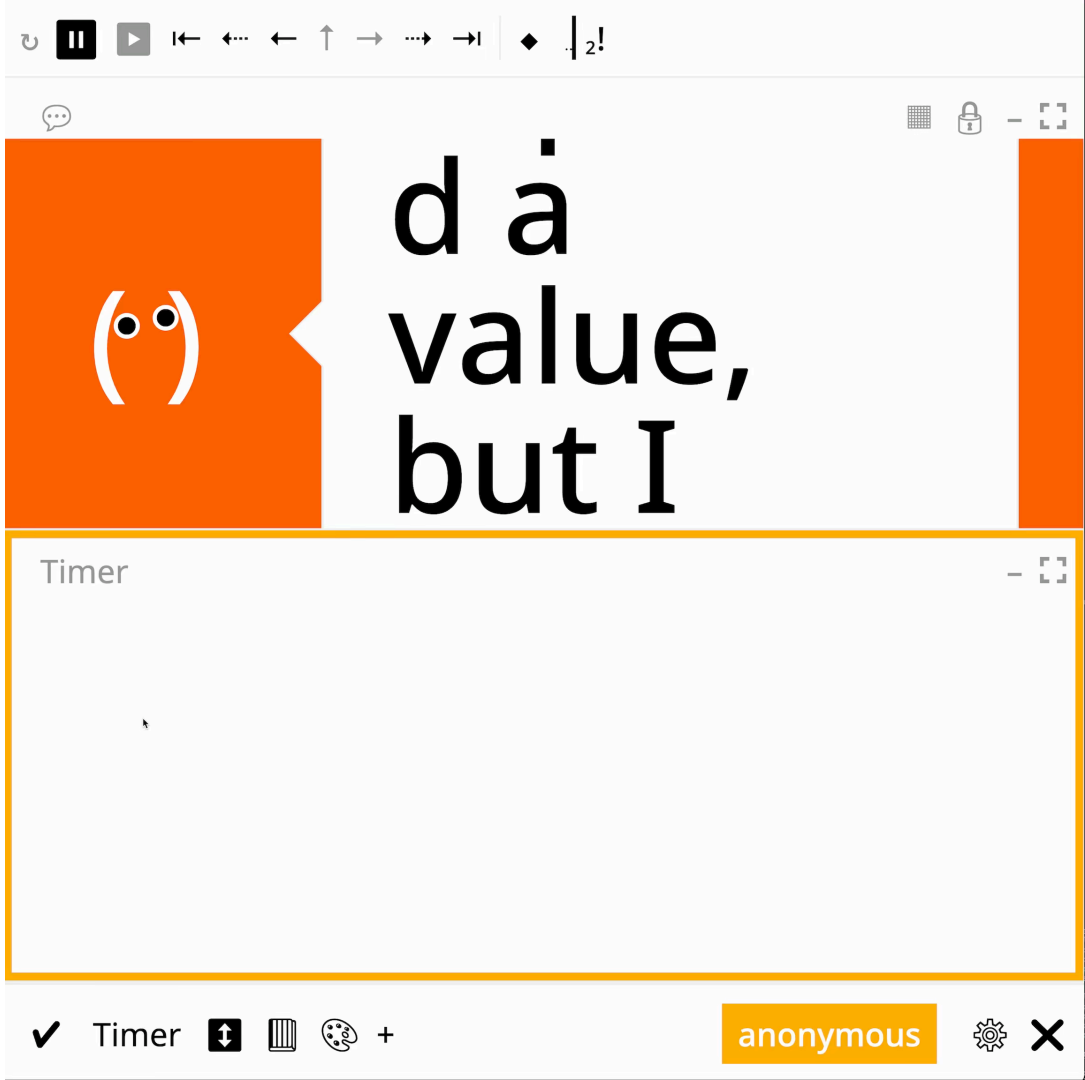
idea: A purely functional, stream-based design

An **anti-capitalist** (e.g., Tormey 2013) programming language might mean liberating learners from these capitalist cognitive burdens, at the expense of speed.

Wordplay operationalizes this with **pure functions**, **immutable data**, and **stream-based reactions**, in an attempt to simplify program comprehension. These features mean only one source of change in program behavior, **input**, and that program output is completely determined by code, not runtime state.

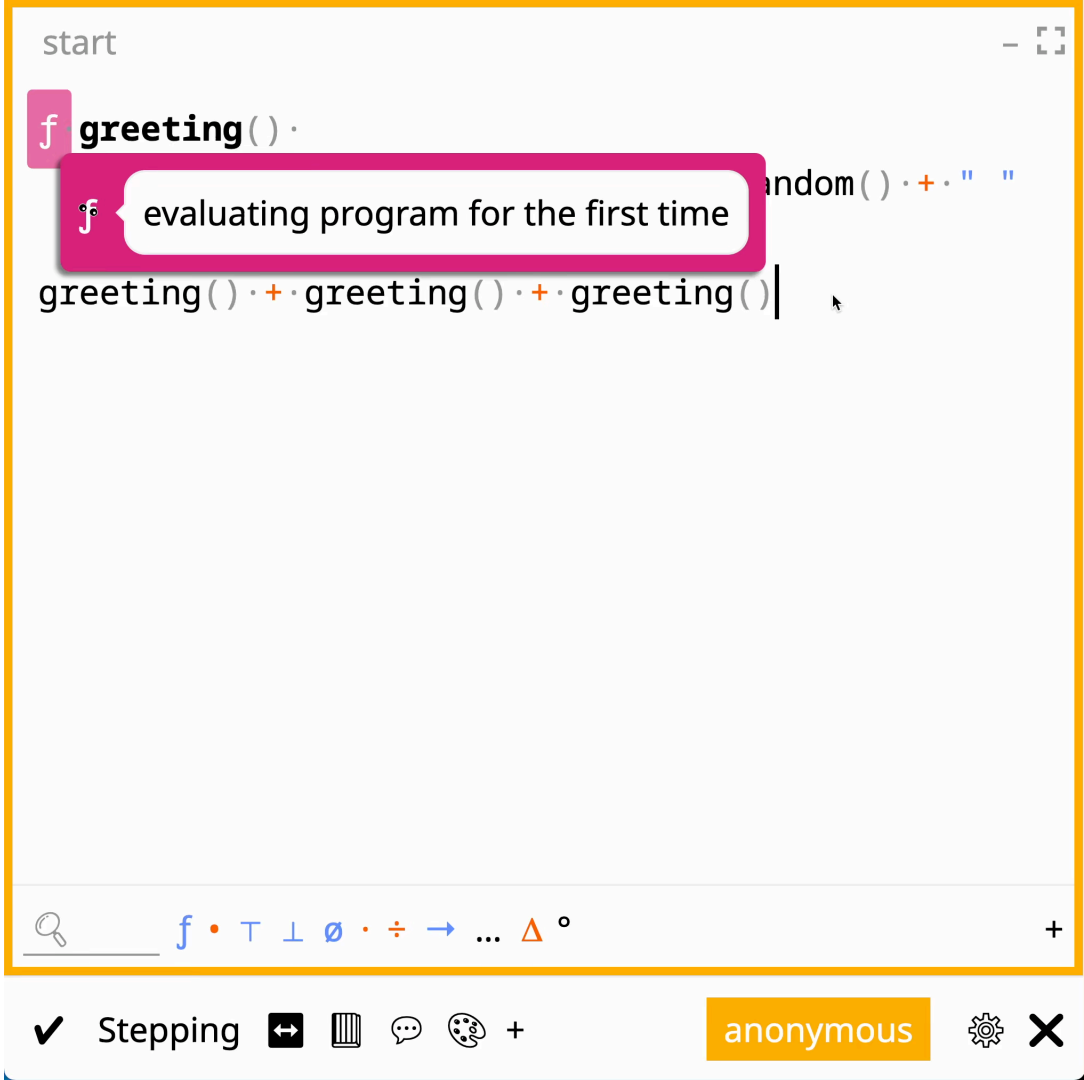
Programs can make and react to **streams** of input that trigger program re-evaluation each time they change. This means that every program is therefore a **recurrence relation** on stream input and prior values.

Here, we use a time stream to create different kinds of timers.



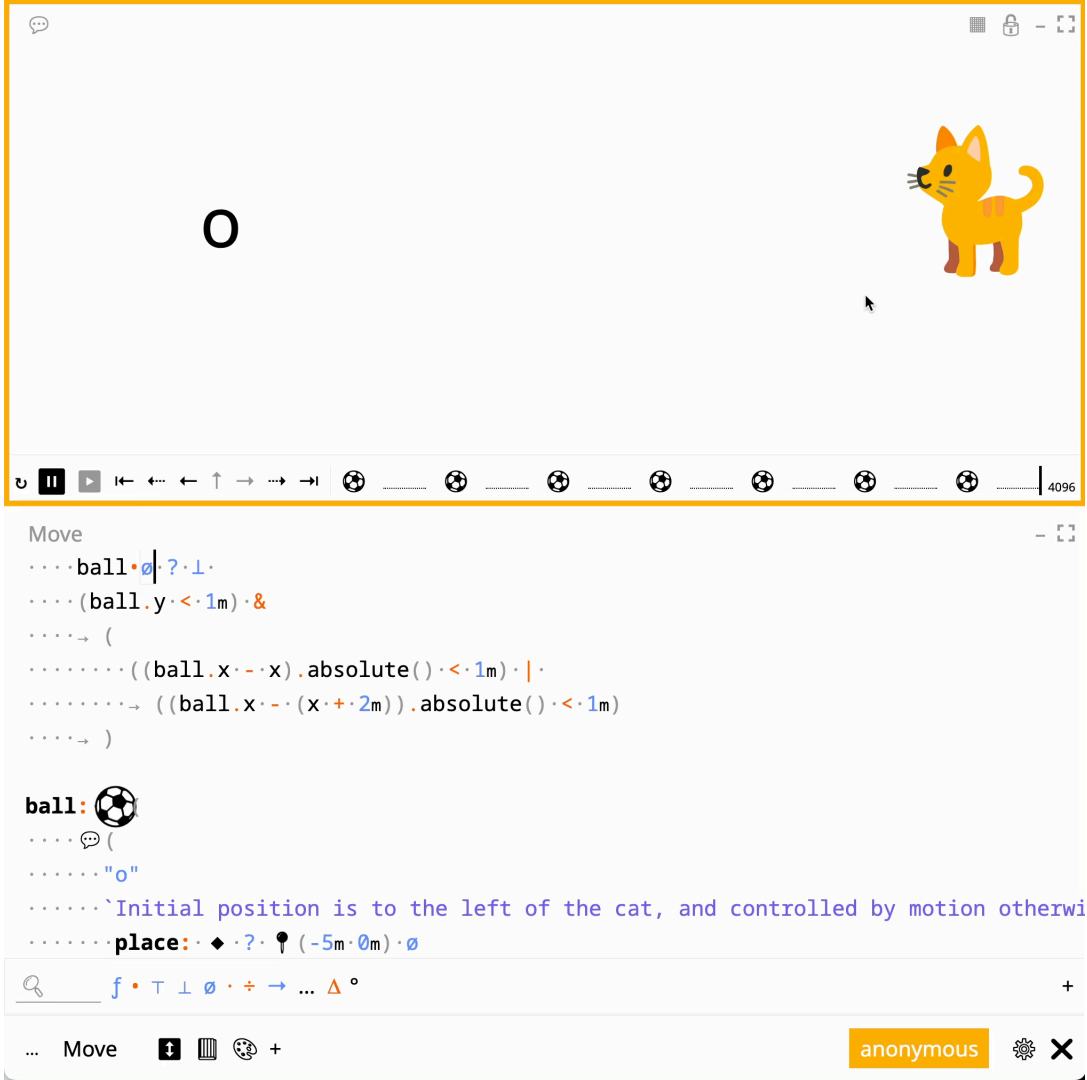
Because program evaluation is just function evaluation, we can step through evaluation one expression at a time, seeing how the program is **translated** into a value.

Here, we create a list of greetings in different languages by appending a **greeting** function's random value three times.



But program output is just a time series of values, and we can recreate any program state from the stream history. This makes **time travel** trivial.

Let's find each time the cat collides with the 'o' by stepping backwards to time, by **scrubbing**, stepping to prior **inputs**, and stepping to prior expression **evaluation**.



Wordplay retains value **provenance**, linking values to the expressions that created them.

This, and the declarative nature of functional code, enables bidirectional editing, enabling direct manipulation of output, despite the lack of mutable state.

questions about complexity

- Is the **lack of mutable state** in functional programming an inherent difficulty, or just a property of poor tooling and lack of interactivity in classic functional languages?
- How does the ability to **manipulate time** change the difficulties of debugging?
- Does **stream input** add complexity relative to stateful event-based interactivity, or reduce it, relative to event-based or constraint-based models?

Is this justice?

Hardly. Making program evaluation comprehensible by making time malleable is the smallest form of liberation from efficiency, and does little to change these broader systems. It is an equitable refuge, surrounded by inescapable forces of labor exploitation.

A broader goal might be **economic justice** (Hahnel, 2005), where programming languages are tools of liberation, creativity, and community, and a source of empowerment for learners to demand and make change.

Wordplay is many things

A purely functional, stream-based, reactive programming language

A new medium for creating interactive, multilingual, accessible typographic media

A fantasy world in which characters collaborate with people and resolve conflicts to create typographic performances

A cultural mashup of language, typography, interactivity, and logic

A small form of resistance to the overwhelming domination of computing ideals in society.

Is all of this enough support the teacher and students I mentioned earlier?

No. A platform is key, but we also need teacher education, curriculum, community, and more.

(stock photo)

But that would not be justice either

Justice is more than just a better tool, and some missing resources.

It is also **agency** and **power** over the future of computing. Because I initially created this alone, on behalf of others, it offers neither of those things. It is simply my worldview, projected onto others' needs.

Justice also means mean sharing power, having programming language designs be **community-led**, rather than led by me.

(stock photo)

I’ve just begun to attempt to do this, starting an open source community of more than 120+ undergraduates.

Many have disabilities, many multilingual English learners, many allies, and have begun to explore how to structure power over Wordplay’s design around the needs, values, and dreams of youth in Puget Sound.

(stock photo)

What I’ve learned so far from students

- Many youth fear the responsibility of power and devalue the expertise of their lived experience, deferring to my authority.
- Many youth cannot imagine a world that was designed around them, instead of one they have to adapt to, and struggle to envision more radical futures.
- Some youth, in contrast to the above, are inspired by radical futurism, and are far more engaged in making a new world than succumbing to the one we have. These are the youth I want to empower to lead this work.

This, of course, is just the beginning.

“We must learn that passively to accept an unjust system is to cooperate with that system, and thereby to become a participant in its evil.” — MLK

Thank you!

Try Wordplay at **wordplay.dev**.

Consider **contributing** as a developer, helping us built a more just future of programming.

Consider **donating**, if you'd like help pay the rent and tuition of our student leaders, so they can work on this instead of working part time at Chipotle.

This work was supported by the University of Washington, National Science Foundation, unrestricted gifts from Google, Microsoft, and Adobe. Thanks also to the Svelte community for making an outstanding modern platform for richly interactive web applications.