KidLisp '26: A Minimal Lisp for Generative Art on a Social Platform

3 March 2026

Summary

KidLisp is a minimal Lisp dialect designed for generative art and interactive audio-visual experiences. It runs inside Aesthetic Computer (AC), a browser-based creative computing platform. KidLisp programs are stored in MongoDB, assigned short alphanumeric codes (e.g., `$cow`, `$27z`), and instantly executable at a URL. A 15,000-line tree-walking evaluator provides 118 built-in functions spanning drawing, color, transformation, math, animation, audio, and input — with no file I/O, networking, or general string manipulation — making it safe to run arbitrary user-submitted code in the browser.

Statement of need

Creative coding environments like Processing (Reas and Fry 2007), p5.js (McCarthy 2015), and Sonic Pi (Aaron 2016) have significantly lowered the barrier to computational expression. However, they still require familiarity with general-purpose language syntax (Java, JavaScript, Ruby) and operate as standalone tools without built-in social distribution.

KidLisp targets the remaining gap: users who want to produce visual and sonic output with zero boilerplate, share it instantly, and build on each other's work. A complete animated program can be a single line:

```
(ink rainbow) (repeat 100 i (circle (wiggle width) (wiggle
    height) 10))
```

Programs are shareable by pasting a short code into the AC prompt or visiting a URL. Embedding other programs is a first-class operation — `($cow)` renders the program stored under code `cow` as a composable layer. As of March 2026, 59 authors have created over 16,000 programs on the platform.

State of the field

Among creative coding tools, Hydra (Jack 2019) is closest in spirit — browser-native, live-coded visuals — but uses JavaScript method chaining and lacks persistent storage or social features. Racket's `2htdp/image` library (Felleisen et al. 2018) uses S-expressions for educational graphics but targets computer science pedagogy, not generative art. Fluxus (Griffiths 2007) used Scheme for live-coded 3D visuals but is no longer maintained.

KidLisp combines Lisp syntax with a browser-native canvas runtime, persistent social storage, and a deliberately constrained function set chosen for generative art safety: no arbitrary code execution, no system access, no imports.

Architecture

Evaluator

The evaluator (`kidlisp.mjs`) parses S-expressions into an AST and walks it, dispatching to built-in functions that map to HTML Canvas 2D, Web Audio API, and WebGL operations. Key design decisions:

- **Flat programs**: No user-defined functions beyond `def`/`let`. Abstraction is achieved by embedding other programs rather than defining procedures.
- **Deterministic rendering**: Given the same random seed and frame count, output is identical — enabling reproducible generative art.
- **Timing syntax**: Temporal operators schedule expressions without explicit loops: `2.5s (wipe red)` executes after 2.5 seconds; `1s... (spin 5)` repeats every second; `3s! (write "Done")` fires exactly once at 3 seconds.
- **Chaos mode**: Invalid or random input triggers artistic visual output rather than error messages. A confidence-scored detector examines word recognition rate, special character ratio, and parenthesis balance to decide whether input is code or chaos.

Storage

Each program is stored as a MongoDB document in the `kidlisp` collection:

```
{
  code: "cow",                        // nanoid, unique
  source: "(wipe blue)\n(ink yellow)\n(circle width/2 height/2 50)",
  hash: "a3f2...",                    // SHA-256 of source, unique (deduplication)
  when: ISODate("2025-06-15"),
  lastAccessed: ISODate("2026-03-01"),
  hits: 342,                          // usage counter
  user: "auth0|abc123",               // optional creator ID
  kept: { tokenId, network, ... } // optional NFT mint record
}
```

The storage API (`store-kidlisp.mjs`) provides five query modes over a single REST endpoint:

1. **POST** — store a program, deduplicate by SHA-256 hash, generate a short code via `nanoid`, and optionally sync to ATProto (Bluesky).
2. **GET `?code=xyz`** — retrieve a single program with user handle resolved via `$lookup` on the `@handles` collection.
3. **GET `?codes=a,b,c`** — batch retrieval (up to 50 codes per request) for embedded layer resolution.
4. **GET `?recent=true`** — paginated feed sorted by creation time or hit count, with optional `handle` filter for per-user feeds.
5. **GET `?stats=functions`** — function usage analytics across the entire corpus, returning weighted counts by piece popularity.

Short codes are generated by a source-aware algorithm: the system first attempts to derive a meaningful code from the program's content (e.g., extracting dominant function names or color keywords), falling back to random `nanoid` generation if all inferred codes are taken.

Media pipeline

The oven service (`oven.aesthetic.computer`) converts recorded sessions into shareable media. When a user records a KidLisp session, the session server uploads a ZIP of PNG frames and optional WAV audio. The oven:

1. Extracts frames and reads `timing.json` for per-frame durations
2. Generates a thumbnail (middle frame, 3x nearest-neighbor upscale, fitted to 512x512)
3. Encodes to H.264/AAC MP4 via FFmpeg with nearest-neighbor upscaling to preserve pixel art aesthetics
4. Uploads to DigitalOcean Spaces CDN (`art-aesthetic-computer.sfo3.cdn.digitaloceanspaces.com`)
5. Stores bake metadata in the `oven-bakes` MongoDB collection and notifies subscribers via WebSocket

The oven also generates Open Graph preview images for social sharing. iOS crawlers require direct image serving (they won't follow 301 redirects), so the oven caches generated PNGs and serves them at stable URLs like `/kidlisp-og.png`.

Performance

The evaluator implements multi-level caching (RAM → IndexedDB → network) for embedded program resolution, buffer pooling (up to 8 reusable buffers per resolution), and an auto-density system that scales pixel density between 0.5x and 4x to maintain 30–55 FPS on varying hardware.

IDE

The KidLisp IDE at `kidlisp.com` provides a Monaco-based editor with custom syntax highlighting, a live preview iframe running the AC runtime, a console panel, and playback controls. A “slide mode” allows dragging numeric literals to adjust values in real time. The editor supports a stage mode for performance contexts that hides all chrome and overlays the editor directly on the canvas.

Example: Composition through embedding

A key feature is the composition graph. The program below embeds two other user-created programs as layers:

```
(wipe black)
(ink rainbow 64)
(repeat 200 i (circle (wiggle width) (wiggle height) (wiggle
   20)))
($27z)
($cow)
```

`$27z` and `$cow` are resolved from MongoDB at runtime, rendered into offscreen buffers, and composited onto the canvas. This enables collaborative layering — users build on each other’s work without copying source code.

Research applications

The corpus of 16,000+ programs with hit counters, timestamps, and author attribution constitutes a dataset for studying creative coding practices. The `?stats=functions` API endpoint already provides function usage analytics weighted by program popularity, enabling quantitative analysis of which primitives users gravitate toward. The embedding graph (which programs reference which others) offers a social network of creative influence amenable to graph analysis.

The platform’s chaos mode — which produces deterministic visual output from arbitrary text input — represents an unusual approach to error handling in creative environments that may interest programming language researchers.

AI usage disclosure

The KidLisp evaluator was primarily written by the author. Claude (Anthropic) was used as a development assistant for debugging, test writing, and documentation. This paper was drafted with AI assistance.

Acknowledgements

Thanks to the 59 KidLisp authors on Aesthetic Computer for creating the corpus and providing feedback on language design.

References

Aaron, Samuel. 2016. "Sonic Pi – Performance in Education, Technology and Art." In *Proceedings of the International Conference on Live Coding.*

Felleisen, Matthias, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2018. "Racket, the Programming Language." PLT Inc. https://racket-lang.org.

Griffiths, Dave. 2007. "Fluxus: A Rapid Prototyping Tool for Live Coding Audio Visual Performances." http://www.pawfal.org/fluxus/.

Jack, Olivia. 2019. "Hydra: Live Coding Networked Visuals." https://hydra.ojack.xyz.

McCarthy, Lauren. 2015. "P5.js." https://p5js.org.

Reas, Casey, and Ben Fry. 2007. "Processing: A Programming Handbook for Visual Designers and Artists." In. Cambridge, MA: MIT Press.