Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

Tom White

@dribnet

Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

Tom White
@dribnet

What is ClojureScript / JavaScript Data Interop?

Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

Tom White
@dribnet

```
(ns demo)

(def extent {:width 800 :height 300})
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))

/*
demo.extent.toString()
demo.extent.width
Object.keys(demo.extent)
cljs.core.clj__GT_js(demo.extent)

demo.alphabet.toString()
demo.alphabet.length
cljs.core.clj__GT_js(demo.alphabet)[0]
*/
```

Elements Resources Network Sources Timeline Profiles Audits Console

```
> demo.extent.toString()
"{:height 300, :width 800}"
> demo.extent.width
undefined
> Object.keys(demo.extent)
["meta", "keys", "strobj", "update_count", "__hash",
"cljs$lang$protocol_mask$partition1$", "cljs$lang$protocol_mask$partition0$"]
> demo.alphabet[0]
undefined
>
```

clj-demo(clj-demo.html) All Errors Warnings Logs Debug

Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

Tom White
@dribnet

```
(ns demo
  (:require [mrhyde]))

(mrhyde/bootstrap)
(def extent {:width 800 :height 300})
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))
```

```
/*
demo.extent.toString()
demo.extent.width
Object.keys(demo.extent)
for (var key in demo.extent) console.log("extent[" + key + "] = " + demo.extent[key])
cljs.core.map_QMARK_(demo.extent)

demo.alphabet.toString()
demo.alphabet[0]
demo.alphabet.length
demo.alphabet[100]
for(var i=0; i<demo.alphabet.length; i+=8) console.log(demo.alphabet[i])
```

Elements Resources Network Sources Timeline Profiles Audits Console

```
> demo.extent.toString()
"{:height 300, :width 800}"
> demo.extent.width
800
> Object.keys(demo.extent)
["height", "width"]
> demo.alphabet[0]
"a"
>
```

clj-demo2(clj-demo2.html) All Errors Warnings Logs Debug

Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

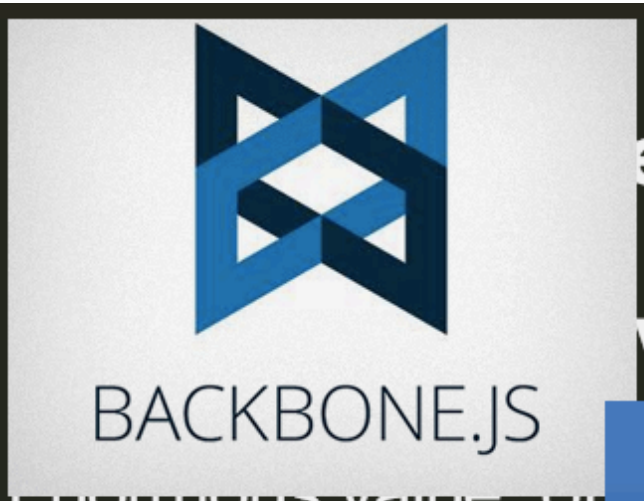
Tom White

@dribnet

Libraries are like instruments

Lots of exciting JavaScript libraries.

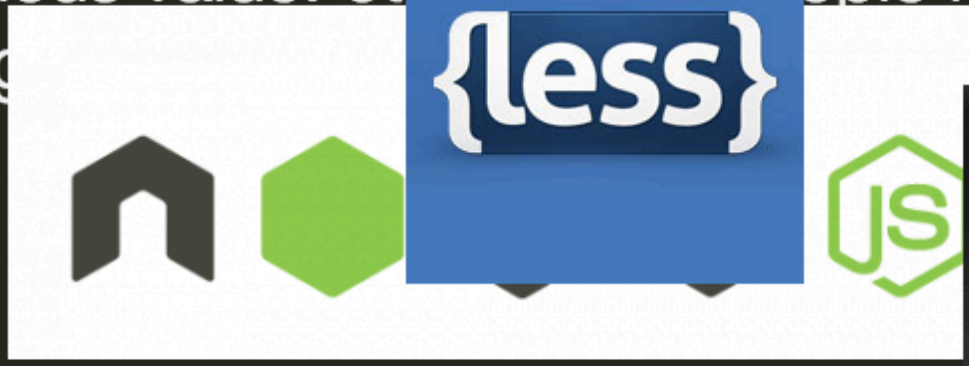
Enormous value: other smart people making, fixing, and refining JavaScript libraries.



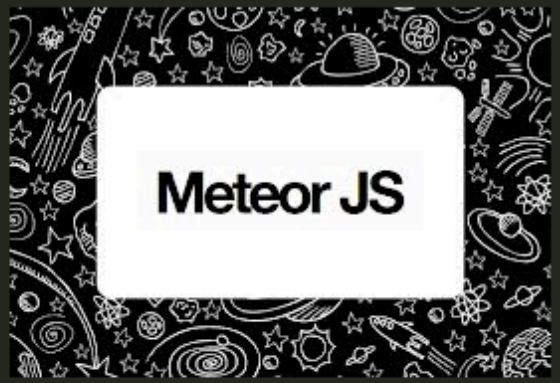
e instruments

vaScript libraries.

Enormous value. Ot pple making, fixing, and refining



Leaflet



‘Let's use the best JavaScript libraries and strive to make it pleasant to do so.”

clj->js and js->clj translations are heavyweight solutions and clutter the code

This infrastructure is meant to be "horizontally scalable" across JavaScript libraries.

Each library has its own personality. Investigate what the JS API is trying to do with your data.

Potential Disappointments:

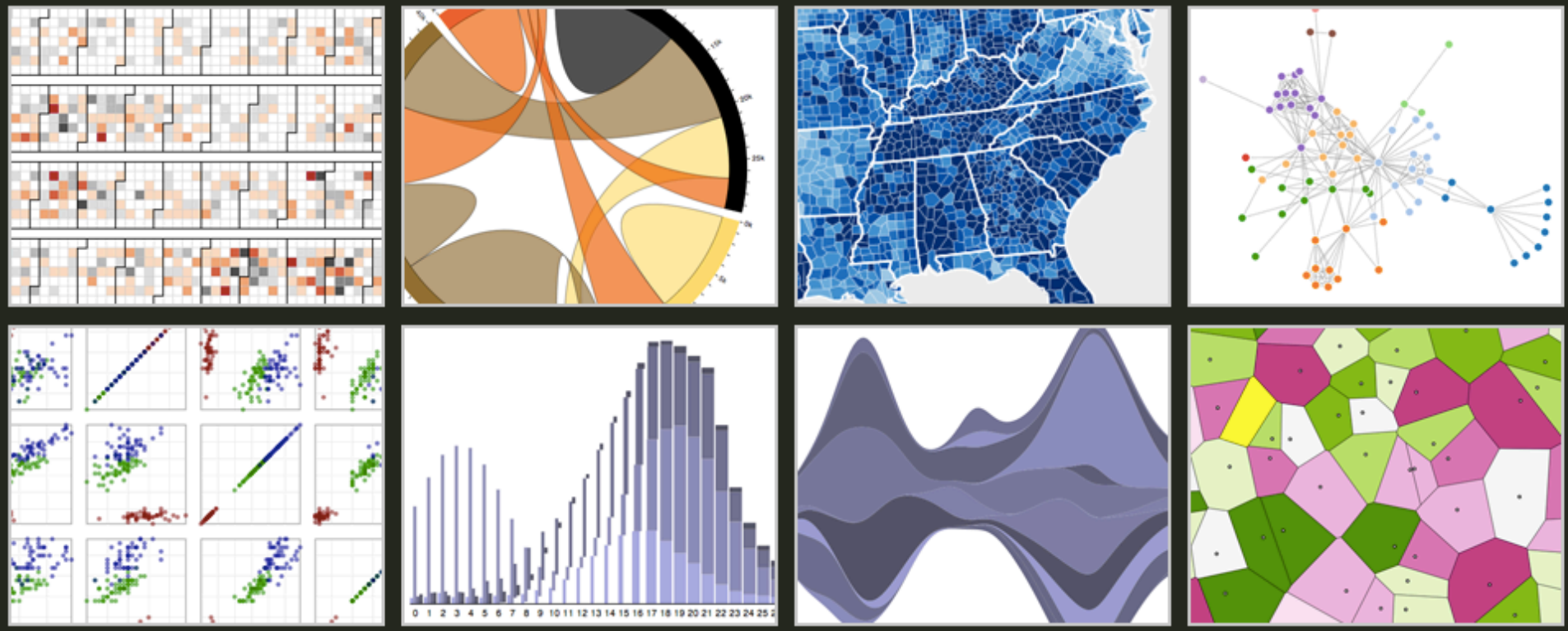
* Fast? Well, faster than clj->js marshalling

* All browsers? Chrome, Safari, Firefox

* Simple? I think so, but jury is still out

* Thorough? For self-imposed constraints

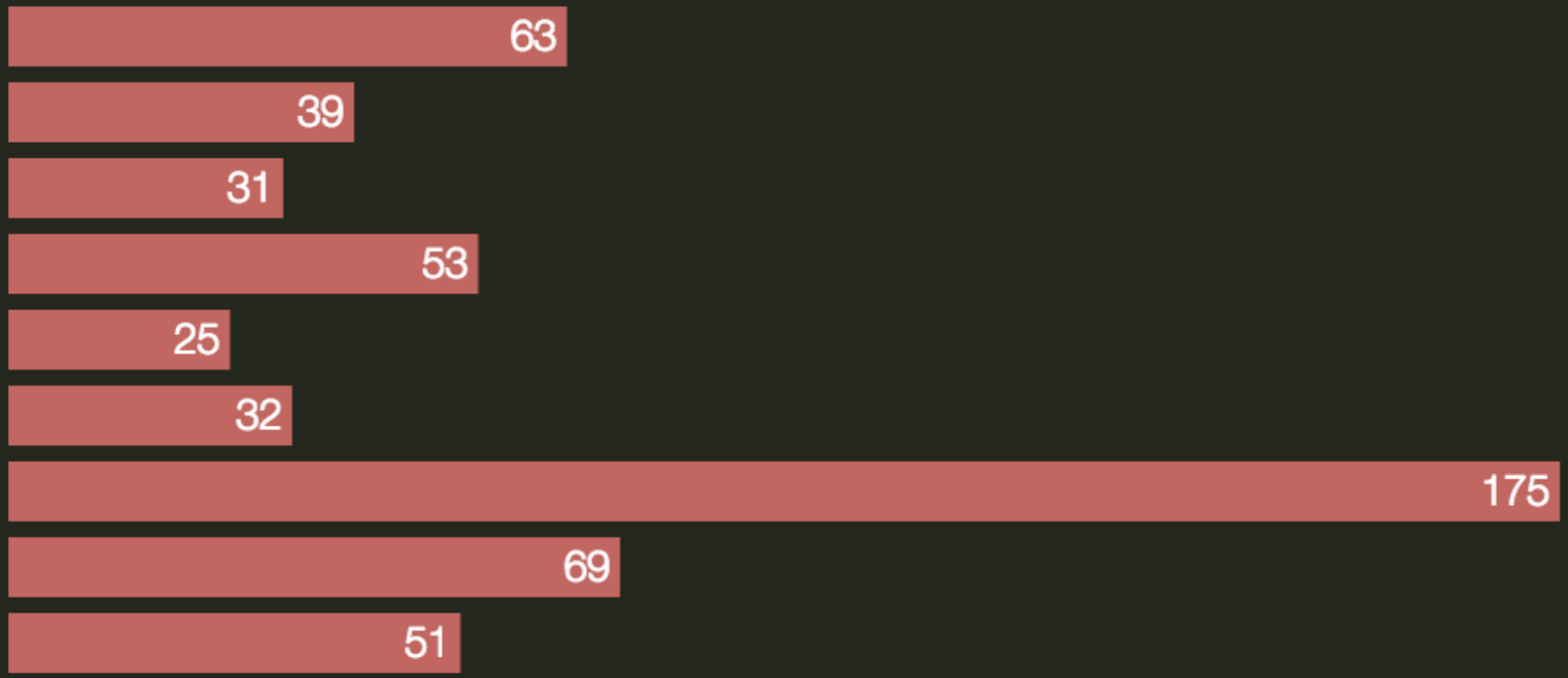
D3.js: Data-Driven Documents



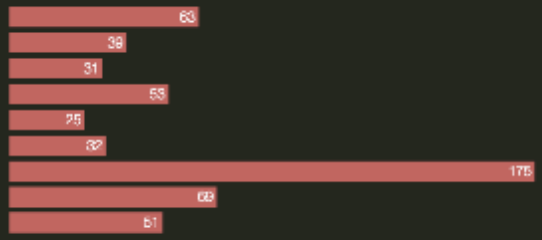
rect: data ↦ width, index ↦ y

63	39	31	53	25	32	175	69	51

rect: data ↦ width, index ↦ y

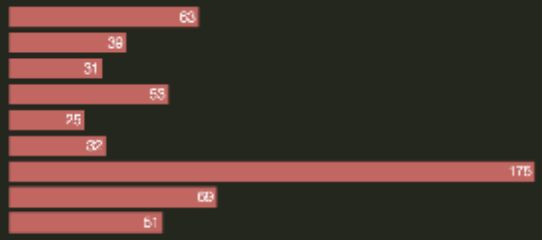


rect: data ↦ width, index ↦ y




```
var data = [63, 39, 31, 53, 25, 32, 175, 69, 51];

// data ↦ width
var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, w]);

// index ↦ y
var y = d3.scale.ordinal()
    .domain(d3.range(data.length))
    .rangeRoundBands([0, h], 0.2);

// Data ↦ Element
var rect = svg.selectAll("rect")
    .data(data);

// Data Attributes ↦ Element Attributes
rect.enter().append("rect")
    .attr("width", function(d, i) { return x(d); })
    .attr("y", function(d, i) { return y(i); })
    .attr("height", y.rangeBand());
```

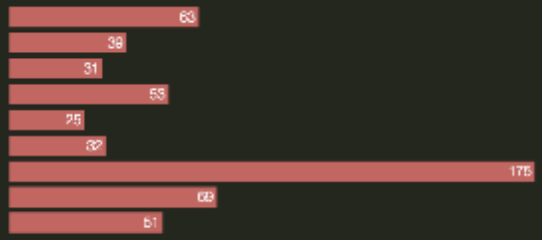
rect: data ↦ width, index ↦ y




```
var data = [63, 39, 31, 53, 25, 32, 175, 69, 51];

// data ↦ width
var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, w]);

// index ↦ y
var y = d3.scale.ordinal()
    .domain(d3.range(data.length))
    .rangeRoundBands([0, h], 0.2);

// Data ↦ Element
var rect = svg.selectAll("rect")
    .data(data);

// Data Attributes ↦ Element Attributes
rect.enter().append("rect")
    .attr("width", function(d, i) { return x(d); })
    .attr("y", function(d, i) { return y(i); })
    .attr("height", y.rangeBand());
```

```
(def data [63 39 31 53 25 32 175 69 51])

; data ↦ width
(def x (-> d3 .-scale (.linear)
  (.domain (array 0 (apply max data)))
  (.range (array 0 w))))

; y is a fn: index ↦ y
(def y (-> d3 .-scale (.ordinal)
  (.domain (apply array (range (count data))))
  (.rangeRoundBands (array 0 h) 0.2)))

; Data ↦ Element
(def rect (-> svg (.selectAll "rect")
   (.data (clj->js data))))

; Data Attributes ↦ Element Attributes
(-> rect (.enter) (.append "rect")
    (.attr (clj->js {:width  #(x %)
                     :y      #(y %2)
                     :height (.rangeBand y)})))
```

rect: data ↦ width, index ↦ y




```
var data = [63, 39, 31, 53, 25, 32, 175, 69, 51];

// data ↦ width
var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, w]);

// index ↦ y
var y = d3.scale.ordinal()
    .domain(d3.range(data.length))
    .rangeRoundBands([0, h], 0.2);

// Data ↦ Element
var rect = svg.selectAll("rect")
    .data(data);

// Data Attributes ↦ Element Attributes
rect.enter().append("rect")
    .attr("width", function(d, i) { return x(d); })
    .attr("y", function(d, i) { return y(i); })
    .attr("height", y.rangeBand());
```

```
(def data [63 39 31 53 25 32 175 69 51])

; data ↦ width
(def x (-> d3 .-scale (.linear)
  (.domain [0 (apply max data)])
  (.range [0 w])))

; index ↦ y
(def y (-> d3 .-scale (.ordinal)
  (.domain (vec (range (count data))))
  (.rangeRoundBands [0 h] 0.2)))

; Data ↦ Element
(def rect (-> svg (.selectAll "rect")
    (.data data)))

; Data Attributes ↦ Element Attributes
(-> rect (.enter) (.append "rect")
    (.attr {:width  #(x %)
            :y      #(y %2)
            :height (.rangeBand y)}))
```

Approach: Metaprogramming Polyfill

Polyfill: a small piece of bootstrap code that "fills in holes" so that a runtime platform will have the capabilities it needs to interact with the real code the follows.

Have the ClojureScript program manipulate its data structures so that JavaScript can interact with the data structures directly and idiomatically.

These capabilities are provided by the *mrhyde* library.

Examples

Approach: Metaprogramming Polyfill

Polyfill: a small piece of bootstrap code that "fills in holes" so that a runtime platform will have the capabilities it needs to interact with the real code the follows.

Have the ClojureScript program manipulate its data structures so that JavaScript can interact with the data structures directly and idiomatically.

These capabilities are provided by the *mrhyde* library.

xs / s / M / L

```
(ns venn
  (:require [mrhyde :refer [d3]]))

(mrhyde/bootstrap)

(def extent {:width 800 :height 300})

(def svg (-> d3 (.select "#venn") (.append "svg")
    (.attr extent)))

(-> svg (.append "circle")
    (.attr {:cx 340 :cy 100 :r 100 :class "left"}))

(-> svg (.append "circle")
    (.attr {:cx 460 :cy 100 :r 100 :class "right"}))

(-> svg (.append "circle")
    (.attr {:cx 400 :cy 200 :r 100 :class "bottom"}))
```

abcdefghijklmnopqrstuvwxyz

```
(ns gup3
  (:require [mrhyde :refer [d3]]))

(mrhyde/bootstrap)

; 26 characters in a vec
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))

(def width 960)
(def height 500)

(def svg (-> d3 (.select "body") (.append "svg")
      (.attr {:width width :height height})
    (.append "g")
      (.attr {:transform (str "translate(32," (/ height 2) ")")})))

(defn update [data]
  ; DATA JOIN
  ; Join new data with old elements, if any.
  (let [text (-> svg (.selectAll "text") (.data data identity))]
    ; UPDATE
    ; Update old elements as needed
    (-> text (.attr {:class "update"})
      (.transition)
        (.duration 750)
        (.attr {:x #(* %2 32)})))

    ; ENTER
    ; Create new elments as needed
    (-> text (.enter) (.append "text")
        (.attr {:class "enter"
                :x     #(* %2 32)
                :y     -60
                :dy    ".35em"})
        (.style {:fill-opacity 1e-6})
        (.text identity)
      (.transition)
        (.duration 750)
        (.attr {:y 0})
        (.style {:fill-opacity 1}))

    ; EXIT
    ; Remove old elements as needed.
    (-> text (.exit)
        (.attr {:class "exit"})
      (.transition)
        (.duration 750)
        (.attr {:y 60})
        (.style {:fill-opacity 1e-6})
        (.remove))))

; The initial display = all 26 letters
(update alphabet)

; Grab a random sample of letters from the alphabet,
;   in alphabetical order.
(.setInterval js/window (fn []
  (->> alphabet
    shuffle
    (take (rand-int 26))
    sort
    vec
    update))
  ; 1.5 seconds between swaps
  1500)
```

aceijlmopsuwxz

```
; ENTER
; Create new elements as needed
(-> text (.enter) (.append "text")
    (.attr {:class "enter"
            :x     #(* %2 32)
            :y     -60
            :dy    ".35em"})
    (.style {:fill-opacity 1e-6})
    (.text identity)
  (.transition)
    (.duration 750)
    (.attr {:y 0})
    (.style {:fill-opacity 1})))
```

horsv

```
; 26 characters in a vec
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))

;; ... later ...

; The initial display - all 26 letters
(update alphabet)

; Grab a random sample of letters from the alphabet,
;   in alphabetical order.
(.setInterval js/window (fn []
  (->> alphabet
    shuffle
    (take (rand-int 26))
    sort
    vec
    update))
  ; 1.5 seconds between swaps
  1500)
```

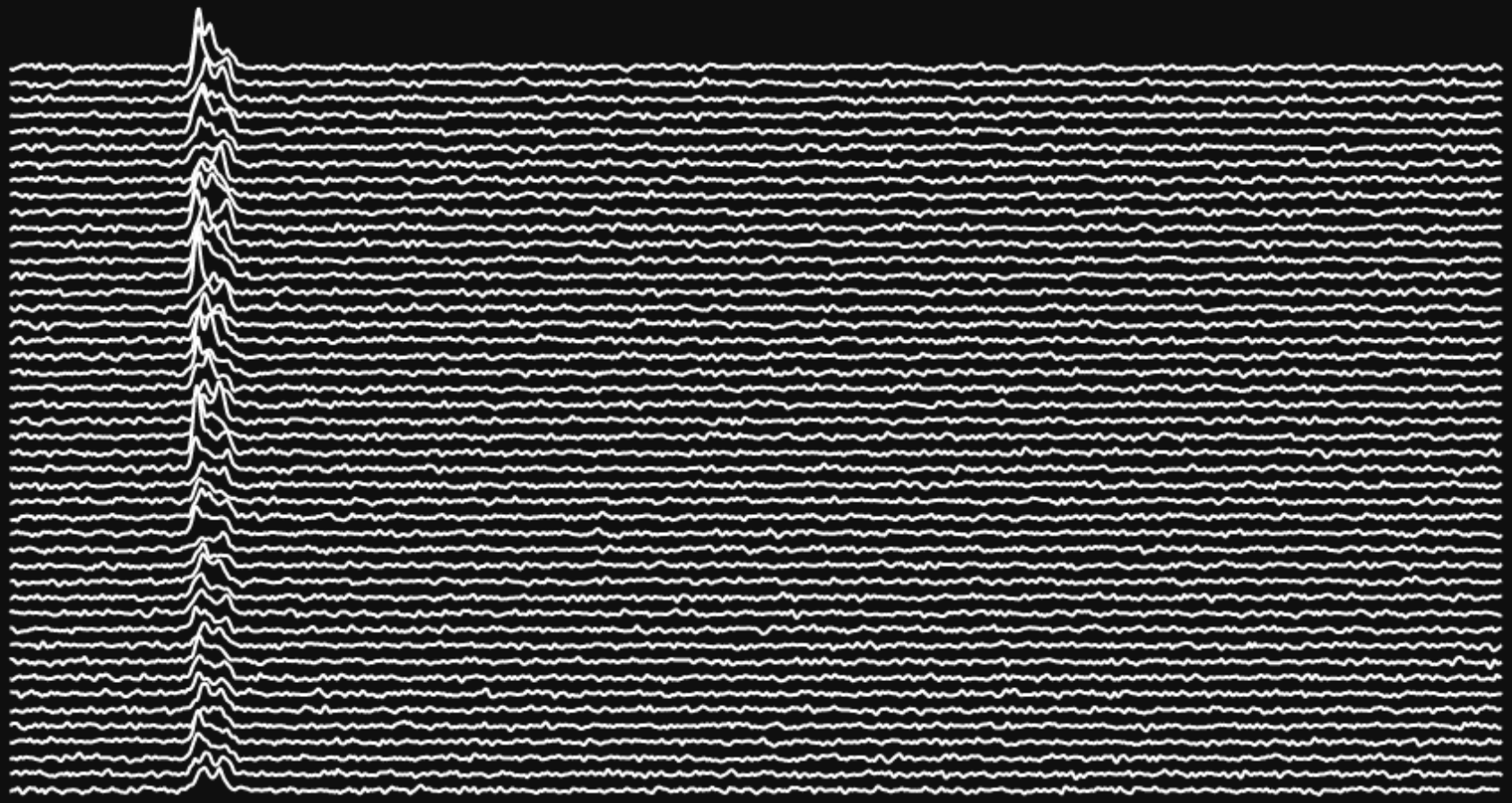
abcdeghjlopqruwxy

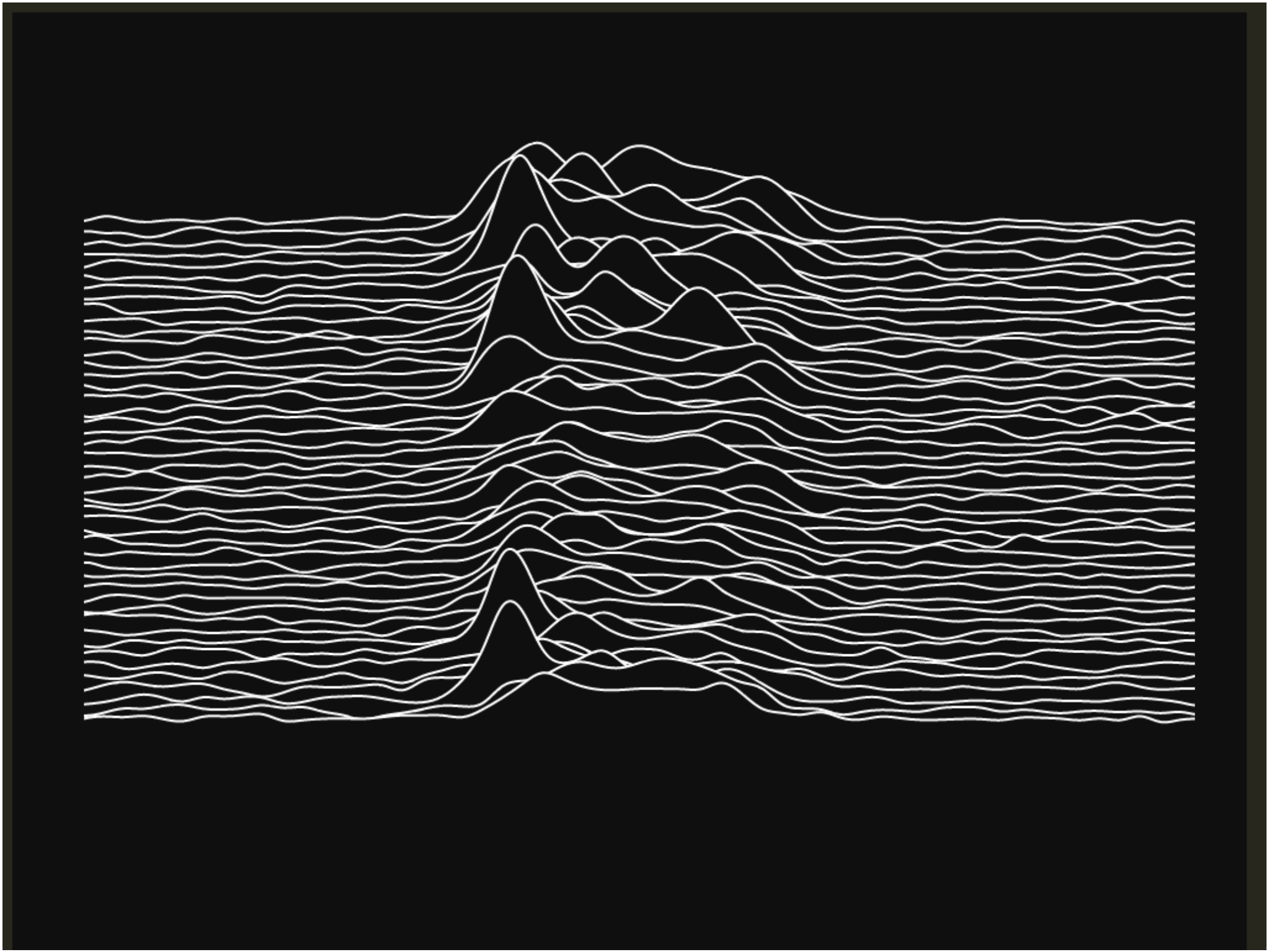
```clojure
; 26 characters in a vec
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))

;; ... later ...

; The initial display - all 26 letters
(update alphabet)

; Grab a random sample of letters from the alphabet,
;   in alphabetical order.
(.setInterval js/window (fn []
  (->> alphabet
    shuffle
    (take (rand-int 26))
    sort
    vec
    update))
  ; 1.5 seconds between swaps
  1500)
```

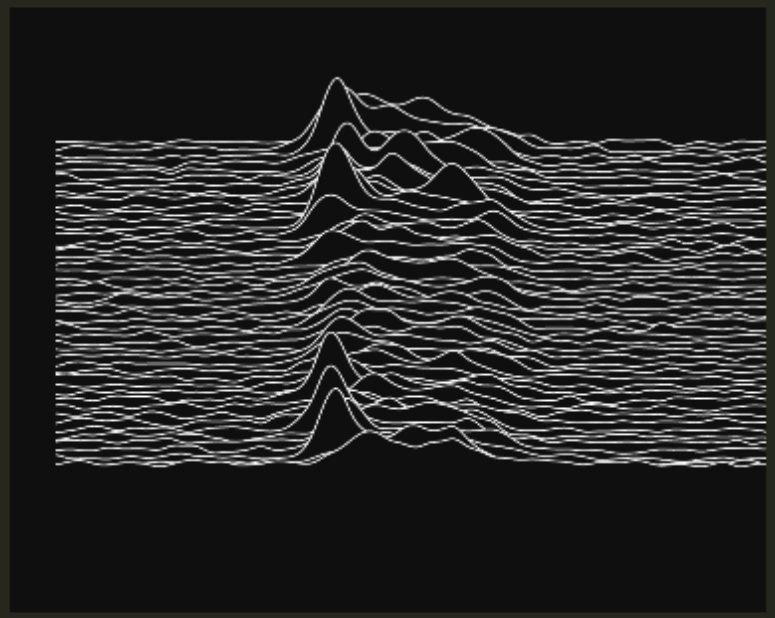
```javascript
// Grab a random sample of letters from the alphabet,
//   in alphabetical order.
setInterval(function() {
  update(shuffle(alphabet)
      .slice(0, Math.floor(Math.random() * 26))
      .sort());
}, 1500);

// Shuffles the input array.
function shuffle(array) {
  var m = array.length, t, i;
  while (m) {
    i = Math.floor(Math.random() * m--);
    t = array[m], array[m] = array[i], array[i] = t;
  }
  return array;
}
```

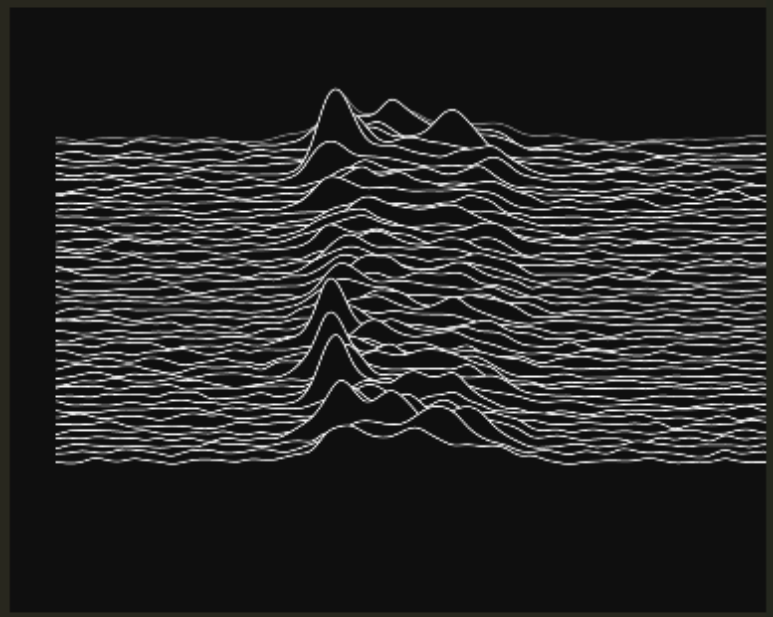
```clojure
(ns b3
  (:require [clojure.string]
            [mrhyde :refer [d3]]))

(mrhyde/bootstrap)

(def span (-> d3 (.select "#hostspan")))

(def titledata [{:name "title" :pos [0 0] :scale 1.0}])

(def pagedata (atom [titledata]))

; destructuring the parameter directly confuses d3's apply somehow
(defn pos-scale-to-str [d i]
  (let [{:keys [pos scale] :or {pos [0 0] scale 1.0}} d]
    ; (.log js/console (str "pos and scale: " pos "," scale))
    (str "-webkit-transform: translate3d(" (first pos) "px,"
      (second pos) "px,0px) scale(" scale ");z-index:" i ";")))

(def curpage (atom 0))
(def stage-exit 2000)
(def curflow (atom stage-exit))
(def timestr (str (.now js/Date)))

(defn pos-scale-to-str-birth [d i]
  (pos-scale-to-str (update-in d [:pos 0] + @curflow) i))

(defn pos-scale-to-str-death [d i]
  (pos-scale-to-str (update-in d [:pos 0] - @curflow) i))

(defn update [data]
  ; DATA JOIN
  ; Join new data with old elements, if any.
  (let [pages (-> span (.selectAll "iframe") (.data data #(:name %)))]
    ; UPDATE
    ; Update old elements as needed
    (-> pages
      (.transition)
        (.duration 750)
        (.attr {:style pos-scale-to-str}))

    ; ENTER
    ; Create new elments as needed
    (-> pages (.enter) (.append "iframe")
      (.attr {:id #(:name %)
              :width 1024
              :height 968
              :src #(str (:name %) ".html?" timestr)
              :style pos-scale-to-str-birth})
      (.transition)
        (.duration 750)
        (.attr {:style pos-scale-to-str}))

    ; EXIT
    ; Remove old elements as needed.
    (-> pages (.exit)
      (.transition)
        (.duration 750)
        (.attr {:style pos-scale-to-str-death})
        (.remove))))

(defn step [n]
  (if (> n 0) (reset! curflow stage-exit))
  (if (< n 0) (reset! curflow (- 0 stage-exit)))
  (swap! curpage #(mod (+ % n) (count @pagedata)))
  (-> js/window .-location .-hash (set! (str @curpage)))
  (update (nth @pagedata @curpage)))
  ;(.log js/console (str "page is " @curpage))

(defn key-fn []
  (let [key (-> d3 .-event .-keyCode)]
    (cond
      (or (= key 188) (= key 37)) (step -1) ; key , or <-
      (or (= key 190) (= key 39)) (step 1)  ; key . or ->
      )))
      ; :else
      ;   (.log js/console (str "The key is " key)))))

(-> d3 (.select "body") (.on "keyup" key-fn))

(update titledata)

(strokes/fetch-edn (str "slides.edn?" timestr) (fn [error, root]
  (if-not (or (zero? error) (nil? error))
    (.log js/console (str "error: " error))
    (do
      (swap! pagedata concat root)
      (let [pageid (clojure.string/replace (-> js/window .-location .-hash) "#" "")]
        (if pageid
          (step (int pageid))
          (step 1)))))))
```

```clojure
[
  [ {:name "title" :pos [-350 -250] :scale 0.3}
    {:name "clj-demo" :pos [250 0] :scale 1.0} ]
  [ {:name "title" :pos [-350 -250] :scale 0.3}
    {:name "clj-demo2" :pos [250 0] :scale 1.0} ]
  [ {:name "title" :pos [0 0] :scale 1.0} ]
  [ {:name "make" :pos [0 0] :scale 1.0} ]
  [ {:name "use" :pos [0 0] :scale 1.0} ]
  [ {:name "constraints" :pos [0 0] :scale 1.0} ]
  [ {:name "victory/index" :pos [0 0] :scale 1.0} ]
  [ {:name "dt" :pos [0 0] :scale 1.0} ]
  [ {:name "bar" :pos [0 0] :scale 1.0} ]
  [ {:name "bar" :pos [-280 0] :scale 0.45}
    {:name "bar-js-code" :pos [380 0] :scale 0.9} ]
  [ {:name "bar" :pos [-280 -200] :scale 0.45}
    {:name "bar-js-code" :pos [-250 200] :scale 0.5}
    {:name "bar-strokeless-src" :pos [390 0] :scale 0.9} ]
  [ {:name "bar" :pos [-280 -200] :scale 0.45}
    {:name "bar-js-code" :pos [-250 200] :scale 0.5}
    {:name "bar-clj-code" :pos [390 0] :scale 0.9} ]
  [ {:name "approach" :pos [0 0] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0}
    {:name "gup-src-end" :pos [320 150] :scale 0.0}
    {:name "gup-src-middle" :pos [-20 120] :scale 0.28}
    {:name "gup-src" :pos [320 150] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0}
    {:name "gup-src-end" :pos [320 150] :scale 0.0}
    {:name "gup-src" :pos [120 150] :scale 0.5}
    {:name "gup-src-middle" :pos [50 150] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0}
    {:name "gup-src-middle" :pos [-20 120] :scale 0.28}
    {:name "gup-src-end" :pos [30 390] :scale 0.38}
    {:name "gup-src" :pos [320 150] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0}
    {:name "gup-src-end" :pos [0 200] :scale 0.8} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0}
    {:name "gup-src-end" :pos [-150 50] :scale 0.64}
    {:name "gup-src-end-js" :pos [250 200] :scale 0.64} ]
  [ {:name "b3-src" :pos [0 0] :scale 1.0} ]
  [ {:name "hightlight-src" :pos [0 0] :scale 1.0} ]
  [ {:name "slides-edn-src" :pos [0 0] :scale 1.0} ]
  [ {:name "vect-to-array" :pos [0 0] :scale 1.0} ]
  [ {:name "vect-to-array" :pos [-500 -300] :scale 0.2}
    {:name "gup/index" :pos [0 100] :scale 0.9} ]
  [ {:name "vect-to-array" :pos [-500 -300] :scale 0.2}
    {:name "gup/index" :pos [-400 300] :scale 0.3}
    {:name "gup-src" :pos [200 0] :scale 1.0} ]
  [ {:name "vect-to-array" :pos [-500 -300] :scale 0.2}
    {:name "minigup" :pos [100 0] :scale 0.9} ]
  [ {:name "vect-to-array" :pos [0 0] :scale 1.0} ]
  [ {:name "map-to-obj" :pos [0 0] :scale 1.0} ]
  [ {:name "venn-src" :pos [0 0] :scale 1.0} ]
  [ {:name "map-to-obj" :pos [0 0] :scale 1.0} ]
  [ {:name "quadtree-src" :pos [0 0] :scale 1.0} ]
  [ {:name "keywords" :pos [0 0] :scale 1.0} ]
  [ {:name "fun-patching" :pos [0 0] :scale 1.0} ]
  [ {:name "gup/index" :pos [0 0] :scale 1.0} ]
  [ {:name "gup-src" :pos [0 0] :scale 1.0} ]
  [ {:name "gup-compare" :pos [0 0] :scale 1.0} ]
  [ {:name "rvoronoi/index" :pos [0 0] :scale 1.0} ]
  [ {:name "rv-compare" :pos [0 0] :scale 1.0} ]
  [ {:name "factual" :pos [0 0] :scale 1.0} ]
  [ {:name "mobster-shot" :pos [0 0] :scale 1.0} ]
  [ {:name "factual-src" :pos [0 0] :scale 1.0} ]
  [ {:name "semantics" :pos [0 0] :scale 1.0} ]
  [ {:name "mutate-demo" :pos [0 0] :scale 1.0} ]
  [ {:name "histo-grow" :pos [0 0] :scale 1.0} ]
  [ {:name "gram/index" :pos [0 0] :scale 1.0} ]
  [ {:name "others" :pos [0 0] :scale 1.0} ]
  [ {:name "arraylike/arraylike" :pos [0 0] :scale 1.0} ]
  [ {:name "blade" :pos [0 0] :scale 1.0} ]
  [ {:name "acute/tutorial/index" :pos [0 0] :scale 1.0} ]
  [ {:name "acute-src" :pos [0 0] :scale 1.0} ]
  [ {:name "acute-edn" :pos [0 0] :scale 1.0} ]
  [ {:name "acute/tutorial/index" :pos [0 0] :scale 1.0} ]
  [ {:name "closing" :pos [0 0] :scale 1.0} ]
  [ {:name "questions" :pos [0 0] :scale 1.0} ]
]
```









Guts of mrhyde

MrHyde1:
Persistent maps => read-only JavaScript Objects

```
(defn install-js-get-property [obj nam getfn]
  (aset reusable-descriptor "get" getfn)
  (.defineProperty js/Object obj nam reusable-descriptor))

(defn gen-map-getter [k]
  (fn []
    (this-as t
        (get t k))))

(defn patch-map-object [m]
  ; hide all existing keys
  (hide-properties m (.keys js/Object m))
  (doseq [k (keys m)]
    (if (and (keyword? k) (not (goog.object.containsKey m (name k))))
      (install-js-get-property m (name k) (gen-map-getter k))))
  m)

(defn patch-core-map-type [corename]
  (let [orig-fn (aget (.-core js/cljs) corename)
        orig-keys (.keys js/Object orig-fn)
        new-fn  (fn [& args]
                  (let [nargs (apply array (cons nil args))
                        binder (js/Function.prototype.bind.apply orig-fn nargs)
                        that  (new binder)]
                    (patch-map-object that)
                    that))]

    (doseq [k orig-keys]
      (aset new-fn k (aget orig-fn k)))
    (aset js/cljs.core corename new-fn)))

(defn patch-map-type [type corename]
 (omitted-light-bookeeping-fn type)
 (patch-core-map-type corename))

(patch-map-type cljs.core.ObjMap "ObjMap")
```

MrHyde2:
Persistent vectors => read-only javascript Arrays

```
(defn install-js-get-property [obj nam getfn]
  (aset reusable-descriptor "get" getfn)
  (.defineProperty js/Object obj nam reusable-descriptor))

(defn gen-seq-getter [n]
  (fn []
    (this-as t
      (nth t n js/undefined))))

(defn hyde-array-map [f]
  (this-as ct
    (doall (map
      #(.call f js/undefined % %2 ct) (seq ct) (range)))))

(defn patch-prototype-as-array [p]
  (install-js-get-prop p "length" #(this-as t (count t)))

  (dotimes [n MAX]
    (install-js-get-prop p n (gen-seq-getter n)))

  (aset p "map" hyde-array-map)
)

(patch-prototype-as-array
  (aget cljs.core.PersistentVector "prototype"))
```

Different Semantics?

ClojureScript

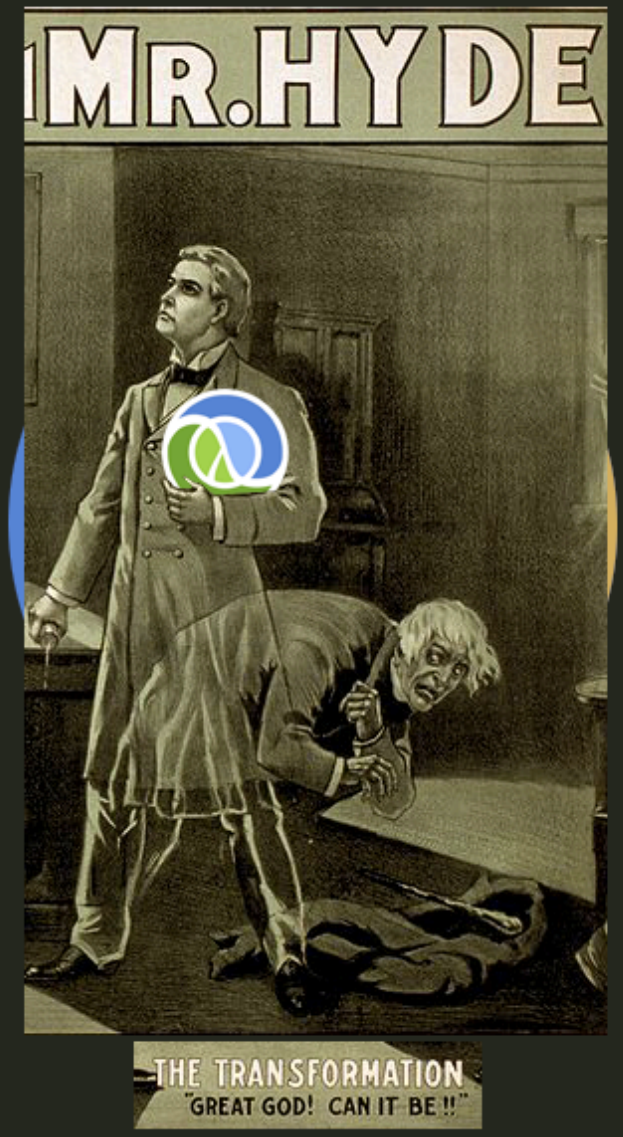
JavaScript

```
a.pop();
a[3] = 100;
```

```
a.mutate = true;
```

Different Semantics?

ClojureScript



JavaScript

```
a.pop();
a[3] = 100;
```

```
a.mutate = true;
```

Different Semantics?

ClojureScript

JavaScript

```
a.pop();
a[3] = 100;
a.mutate = true;
```

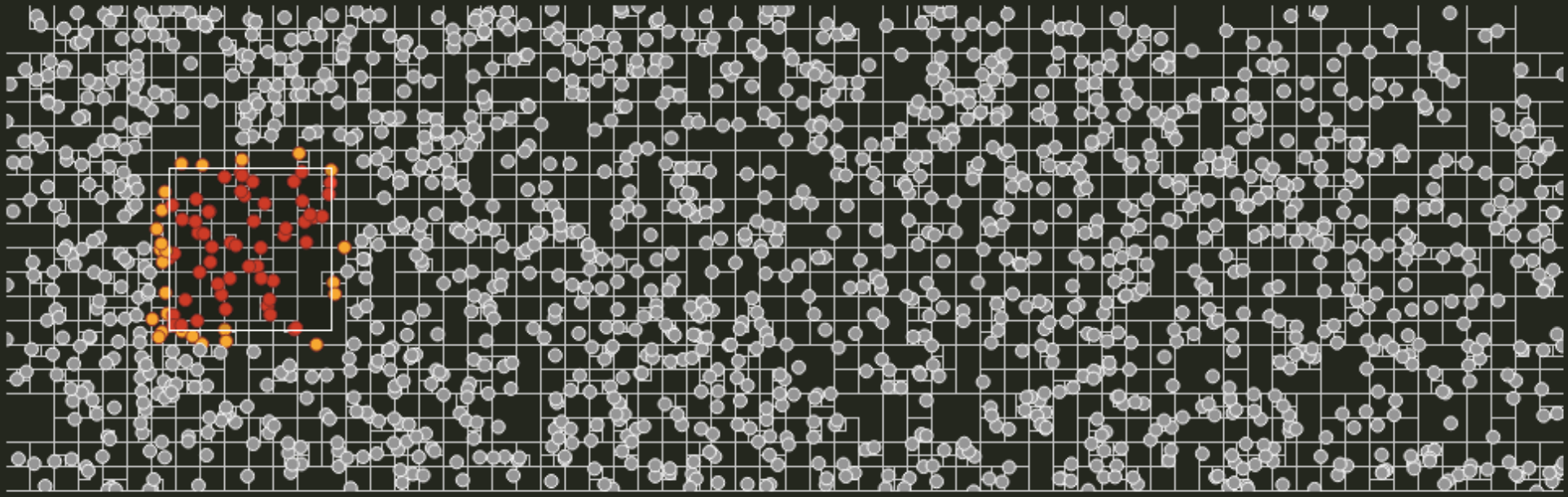
```
(ns demo
  (:require [mrhyde]))

(mrhyde/bootstrap)

; 26 characters in a vec
(def alphabet (vec "abcdefghijklmnopqrstuwvxyz"))

(def extent {:width 800 :height 300})

(defn clj-get [m k]
  (get m (if (string? k) (keyword k) k)))
```

```
/*
demo.clj_get(demo.alphabet, 0)
demo.alphabet[0]
demo.alphabet[0] = "X"
demo.clj_get(demo.alphabet, 0)
*/
```

Elements Resources Network Sources Timeline Profiles Audits Console

```
> demo.alphabet[0]
"a"
> demo.alphabet[0] = "X"
"X"
> demo.alphabet[0]
"X"
> demo.clj_get(demo.alphabet, 0)
"a"
>
```

mutate-demo(mutate-demo.ht▼ All Errors Warnings Logs Debug



```
(-> svg (.selectAll ".node")
  (.data (nodes @quadtree))
(.enter) (.append "rect")
  (.attr "class" "node")
  (.attr "x" #(:x %))
  (.attr "y" #(:y %))
  (.attr "width" #(:width %))
  (.attr "height" #(:height %)))
```

ClojureScript CLJS-451

keywords should be implemented as JavaScript functions

Log In

Views

Details

Type:	Defect	Status:	
Priority:	Minor	Resolution:	
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

People

Assignee:	Unassigned
Reporter:	tom white
Vote (0)	Watch (0)

Dates

Created:	30/Dec/12 5:47 AM
Updated:	
Resolved:	

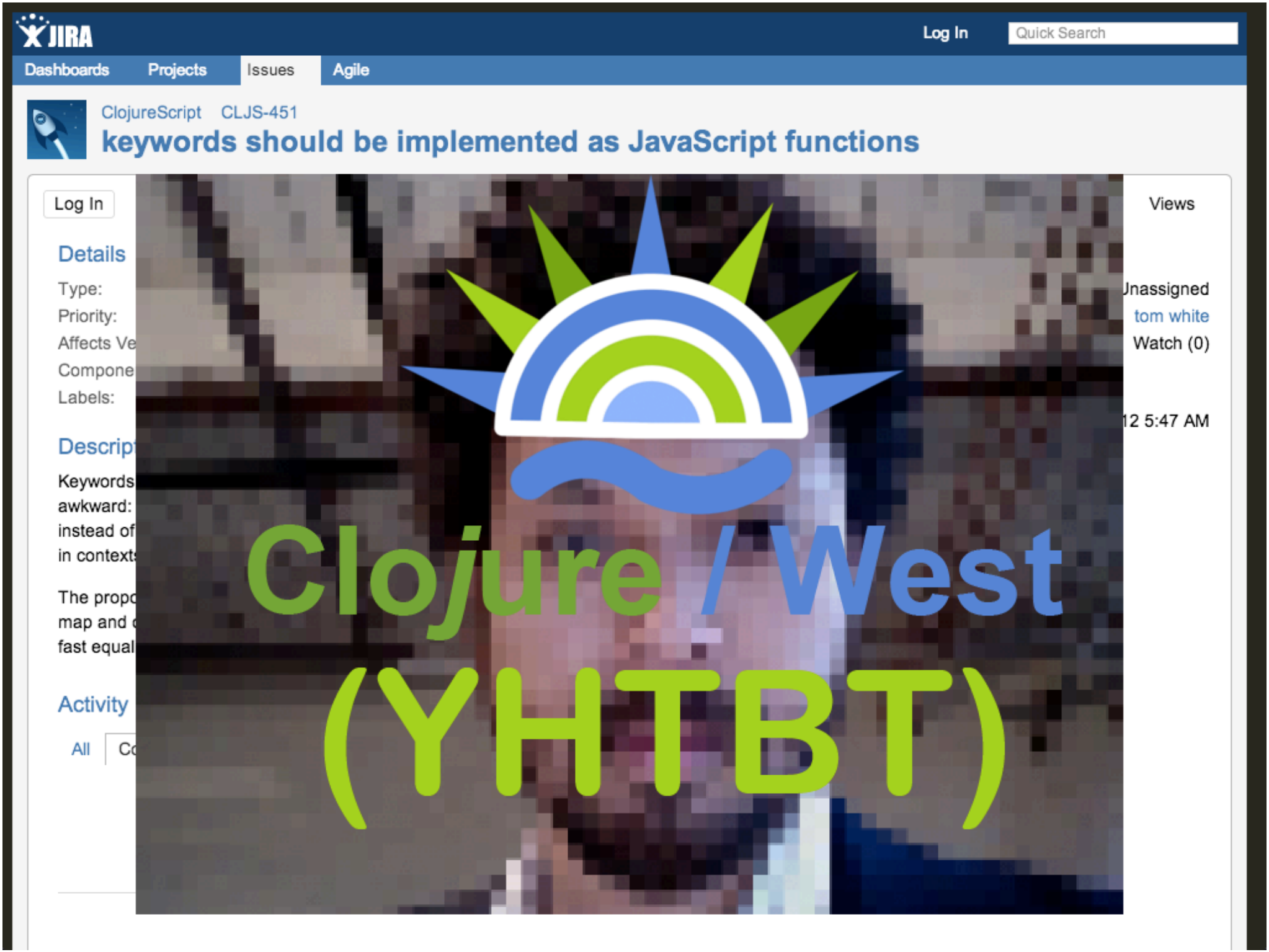
Description

Keywords are currently implemented as JavaScript strings. This makes certain JavaScript interop situations awkward: to provide a javascript api a function that can pull data from a map, one must use #(:foo %) instead of the more natural :foo. This is confusing because ClojureScript functions are otherwise fully usable in contexts expecting JavaScript functions.

The proposed alternative is to have keywords instead be implemented as JavaScript functions that accept a map and optional default value, assuming this doesn't otherwise screw up their core functionality of providing fast equality tests, etc.

Activity

All | Comments | History | Activity


JIRA
Log In
Quick Search
Dashboards
Projects
Issues
Agile
ClojureScript CLJS-451
keywords should be implemented as JavaScript functions
Log In
Details
Type:
Priority:
Affects Ve
Compone
Labels:
Descript
Keywords
awkward:
instead of
in context
The propo
map and
fast equal
Activity
All
Co
Views
Unassigned
tom white
Watch (0)
12 5:47 AM
Clojure / West
(YHTBT)

ClojureScript CLJS-451

keywords should be implemented as JavaScript functions

Log In

Views

Details

Type:	Defect	Status:	Resolved
Priority:	Minor	Resolution:	Declined
Affects Version/s:	None	Fix Version/s:	None
Component/s:	None		
Labels:	None		

People

Assignee:	Unassigned
Reporter:	tom white
Vote (0)	Watch (0)

Dates

Created:	30/Dec/12 5:47 AM
Updated:	04/Jan/13 4:52 PM
Resolved:	04/Jan/13 4:52 PM

Description

Keywords are currently implemented as JavaScript strings. This makes certain JavaScript interop situations awkward: to provide a javascript api a function that can pull data from a map, one must use #(:foo %) instead of the more natural :foo. This is confusing because ClojureScript functions are otherwise fully usable in contexts expecting JavaScript functions.

The proposed alternative is to have keywords instead be implemented as JavaScript functions that accept a map and optional default value, assuming this doesn't otherwise screw up their core functionality of providing fast equality tests, etc.

Activity

All Comments History Activity

David Nolen added a comment - 04/Jan/13 4:52 PM

Keywords & Symbols are likely to become a proper ClojureScript datatypes. The benefit of keywords & symbols as implemented as functions is small.

MrHyde3:
Function Patching - interop duct tape

```
(defn get-store-cur-js-fn [o field-name]
  (let [cur-fn (aget o field-name)
        js-fn-name (str "_js_" field-name)
        root-fn (aget o js-fn-name)]
    ; first store original (if we have not already done so)
    (if (= js/undefined root-fn)
      (aset o js-fn-name cur-fn))
    cur-fn))

; patch a js function convert specified keyword args to functions
(defn patch-args-keyword-to-fn [o field-name & fields]
  (let [orig-fn (get-store-cur-js-fn o field-name)
        arg-filter (if (empty? fields) #(identity true) (set fields))]
    (aset o field-name
      (fn [& args]
        (let [nargs (map (fn [c x]
                           (if (and (arg-filter c) (keyword? x)) #(x %) x))
                         (range) args)]
          (this-as ct (.apply orig-fn ct nargs)))))))

(patch-args-keyword-to-fn (-> d3 .-selection .-prototype) "attr" 1)
```

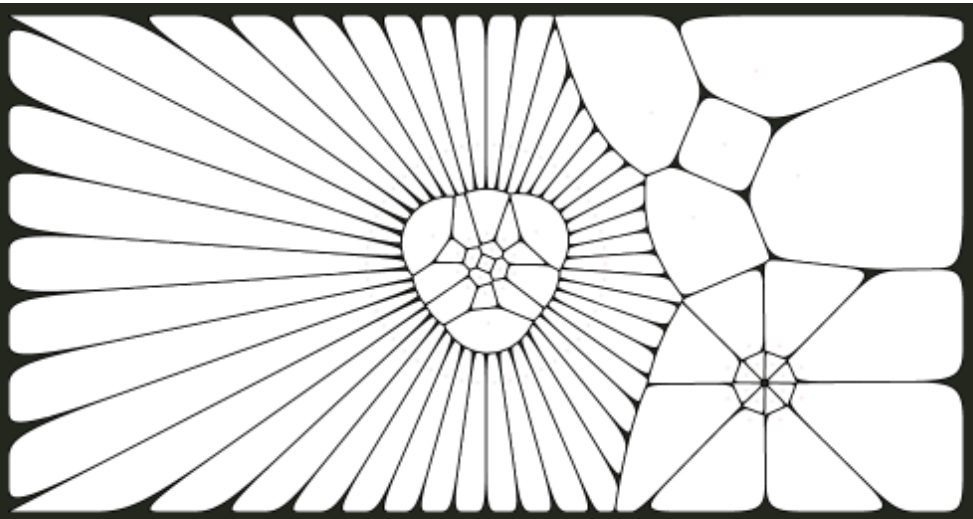
strokes = mrhyde + d3

Examples

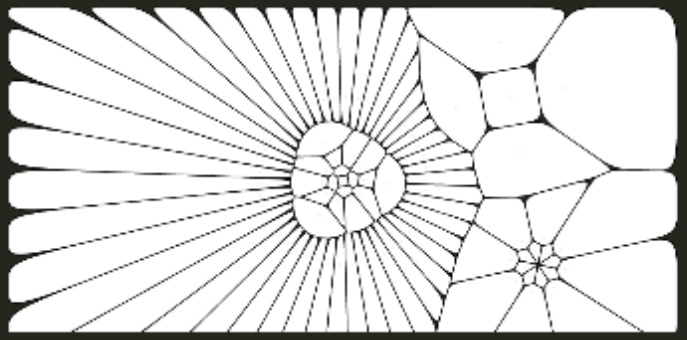
MrHyde3:
Function Patching - interop duct tape

```
(defn get-store-cur-js-fn [o field-name]
  (let [cur-fn (aget o field-name)
        js-fn-name (str "_js_" field-name)
        root-fn (aget o js-fn-name)]
    ; first store original (if we have not already done so)
    (if (= js/undefined root-fn)
      (aset o js-fn-name cur-fn))
    cur-fn))

; patch a js function convert specified keyword args to functions
(defn patch-args-keyword-to-fn [o field-name & fields]
  (let [orig-fn (get-store-cur-js-fn o field-name)
        arg-filter (if (empty? fields) #(identity true) (set fields))]
    (aset o field-name
      (fn [& args]
        (let [nargs (map (fn [c x]
                           (if (and (arg-filter c) (keyword? x)) #(x %) x))
                         (range) args)]
          (this-as ct (.apply orig-fn ct nargs)))))))

(patch-args-keyword-to-fn (-> d3 .-selection .-prototype) "attr" 1)
```

CLJS + D3 = BFF



```
function resample(points) {
  var i = -1,
      n = points.length,
      p0 = points[n - 1], x0 = p0[0], y0 = p0[1], p1, x1, y1,
      points2 = [];
  while (++i < n) {
    p1 = points[i], x1 = p1[0], y1 = p1[1];
    points2.push(
      [(x0 * 2 + x1) / 3, (y0 * 2 + y1) / 3],
      [(x0 + x1 * 2) / 3, (y0 + y1 * 2) / 3],
      p1
    );
    p0 = p1, x0 = x1, y0 = y1;
  }
  return points2;
}
```



```
function resample(points) {
  var i = -1,
      n = points.length,
      p0 = points[n - 1], x0 = p0[0], y0 = p0[1], p1, x1, y1,
      points2 = [];
  while (++i < n) {
    p1 = points[i], x1 = p1[0], y1 = p1[1];
    points2.push(
      [(x0 * 2 + x1) / 3, (y0 * 2 + y1) / 3],
      [(x0 + x1 * 2) / 3, (y0 + y1 * 2) / 3],
      p1
    );
    p0 = p1, x0 = x1, y0 = y1;
  }
  return points2;
}
```

```
(defn three-polate [[[x0 y0] [x1 y1]]]
  [ [x0 y0]
    [(/ (+ (* 2 x0) x1) 3), (/ (+ (* 2 y0) y1) 3)]
    [(/ (+ (* 2 x1) x0) 3), (/ (+ (* 2 y1) y0) 3)] ])

(defn resample [pts]
  (let [pairs (map vector (cons (last pts) pts) pts)]
    (mapcat three-polate pairs)))
```

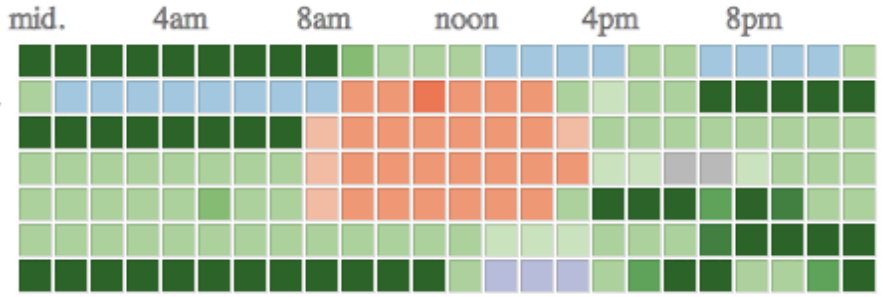
factual™

Factual

la.prod.factual.com/

6 categories

ID	KEY
Social > Arts > Art Dealers and Galleries	
Shopping > Fashion > Clothing & Accessories	
Business & Professional Services	
Food & Beverage > Restaurants	
Shopping > Fashion > Jewelry & Watches	
Real Estate & Home Improvement	

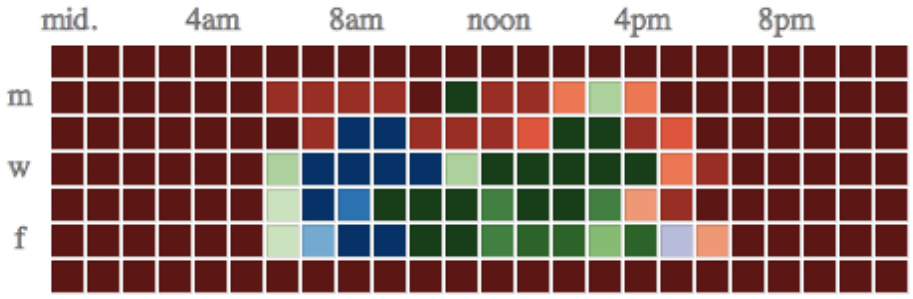


Factual

la.prod.factual.com/

6 categories

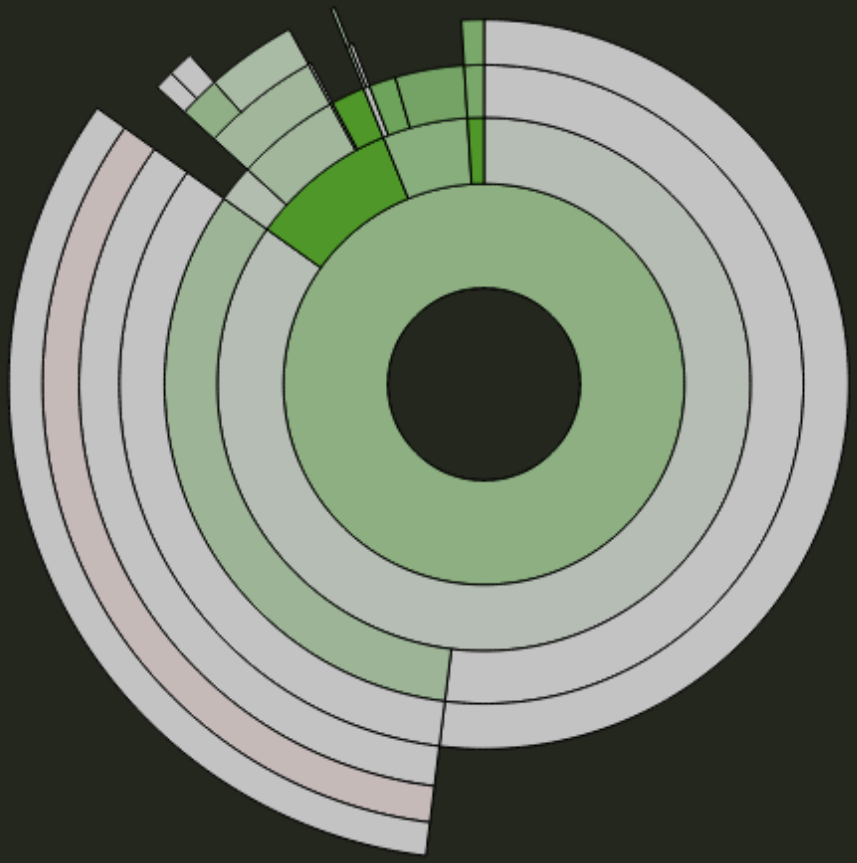
ID
Social > Zoos, Aquariums and Wildlife Sanctuaries
Business & Professional Services
Business & Professional Services > Equipment, Sup
Business & Professional Services > Advertising & M
Business & Professional Services > Equipment, Sup
Legal & Financial > Banking & Financing



d3 + hiccups

```
; https://github.com/teropa/hiccups
(:require-macros [hiccups.core :as hiccups])

; and then later...
(-> tbody
  (.selectAll "tr")
  (.data (vec main-cats))
  (.enter)
  (.append "tr")
  (.html
    (fn [d i]
      (hiccups/html
        [:td d]
        (let [hue (hue-names
                        (min
                          (dec (count hue-names))
                          (inc i)))]
          [:td
            [:table
              [:tr (map
                     (fn [x]
                       [:td {:class (str "stripped " hue "-q" x "-9")}])
                     (range 8))]]])))))
```

duration

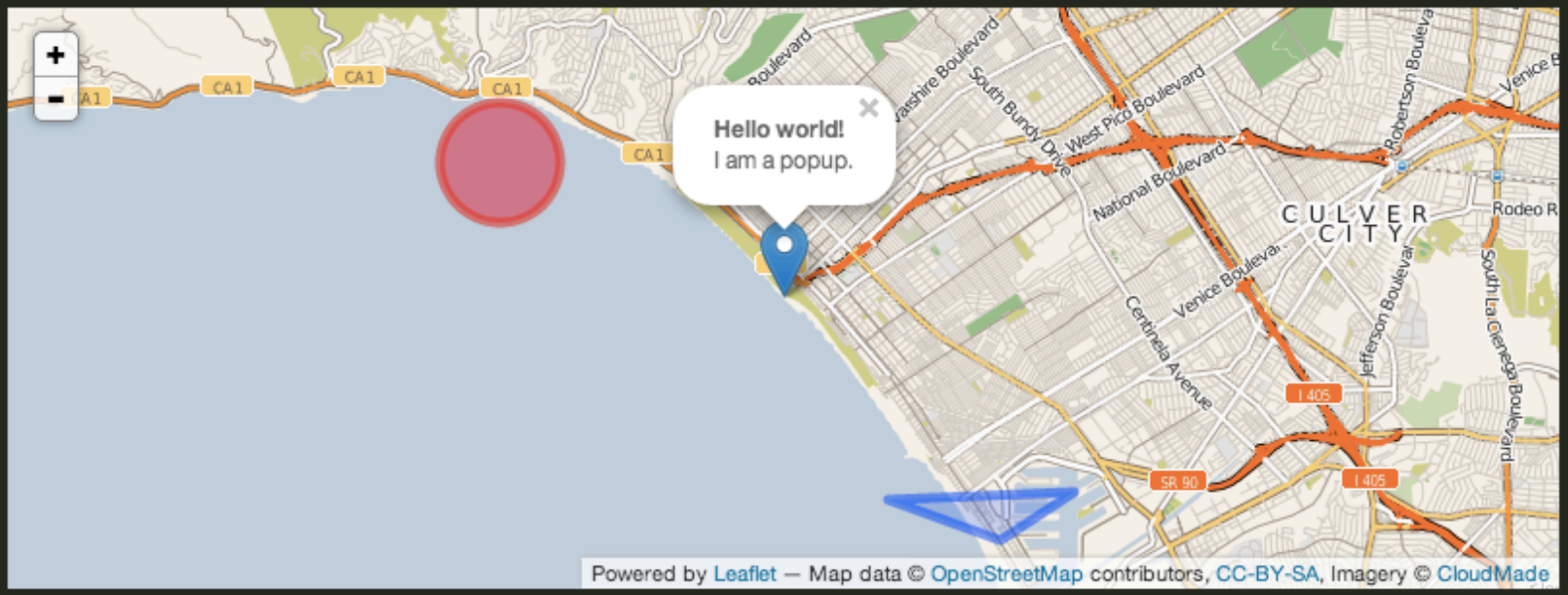


datastore-api-server:services:summaries:generate-summary-diffs

Does mrhyde work with other JavaScript libraries?

d3 trusts arguments are arrays without inspection

```
<!-- this polyfill provides ArrayLike.js compatibility -->
<script type="text/javascript" src="ArrayLikeIsArray.js"></script>
<script type="text/javascript" src="leaflet.js"></script>
<script type="text/javascript" src="mrhyde-program.js"></script>
```




```
(-> L (.circle [34.0286 -118.5486] 1000 {
           :color "red"
           :fillColor "#f03"
           :fillOpacity "0.5"})
      (.addTo mappy)
      (.bindPopup "I am a circle."))

(-> L (.polygon [
        [33.979 -118.48]
        [33.973 -118.46]
        [33.98 -118.447]])
      (.addTo mappy)
      (.bindPopup "I am a <del>polygon</del> triangle"))
```

Search:

motor

Sort by:

Newest

Motorola XOOM™ with Wi-Fi

The Next, Next Generation Experience the future with Motorola XOOM with Wi-Fi, the world's first tablet powered by Android 3.0 (Honeycomb).

MOTOROLA XOOM™

The Next, Next Generation Experience the future with MOTOROLA XOOM, the world's first tablet powered by Android 3.0 (Honeycomb).

MOTOROLA ATRIX™ 4G

MOTOROLA ATRIX 4G the world's most powerful smartphone.

DROID™ 2 Global by Motorola

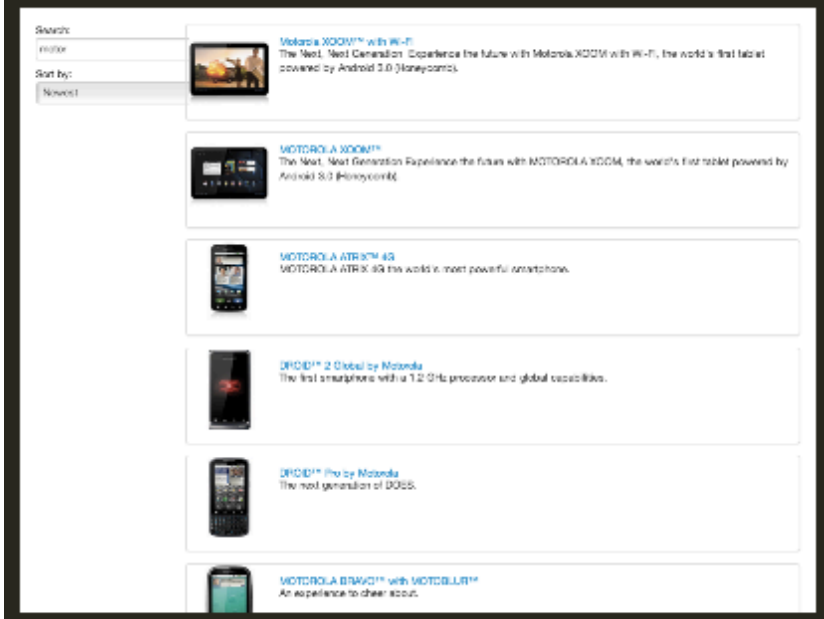
The first smartphone with a 1.2 GHz processor and global capabilities.

DROID™ Pro by Motorola

The next generation of DOES.

MOTOROLA BRAVO™ with MOTOBLUR™

An experience to cheer about.



app.cljs

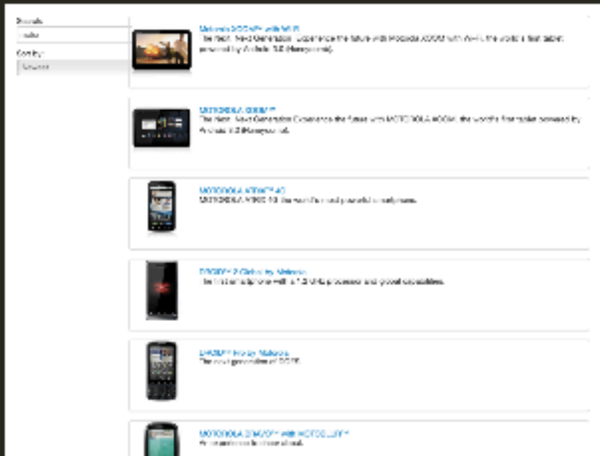
```
(ns acute.tutorial.app
  (:require [cljs.reader :refer [read-string]]
            [acute :refer [angular bootstrap]]
            [acute.tutorial.controllers :refer [PhoneListCtrl PhoneDetailCtrl]]))

(bootstrap)

(let [app (-> angular
              (.module "phonecat" ["phonecatFilters"]))]
  (-> app
    (.config
      ["$routeProvider" (fn [$routeProvider]
        (-> $routeProvider
          (.when "/phones" {:templateUrl "partials/phone-list.html"
                            :controller PhoneListCtrl})
          (.when "/phones/:phoneId" {:templateUrl "partials/phone-detail.html"
                                     :controller PhoneDetailCtrl})
          (.otherwise {:redirectTo "/phones"})))]))
  (-> app
    (.config
     ["$httpProvider" (fn [$httpProvider]
        (set! (-> $httpProvider .-defaults .-transformResponse)
          (fn [x]
            ; given a string, is it edn?
            (if (and (string? x)
                     (or (re-find #"^\{\:" x) (re-find #"^\[\{" x)))
              (read-string x)
              x))))]))))
```

controllers.cljs

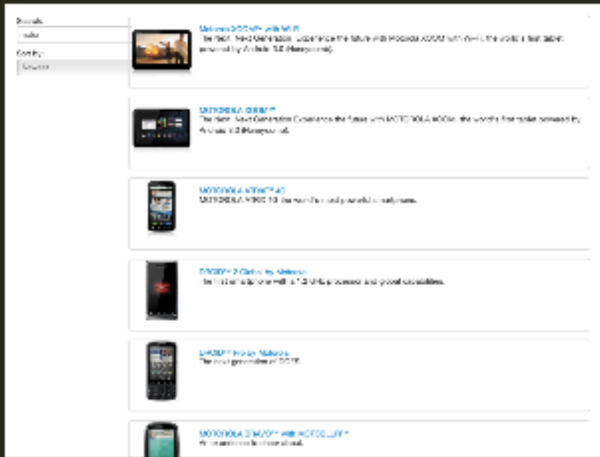
```
(ns acute.tutorial.controllers
  (:require [cljs.reader :refer [read-string]]
            [acute :refer [angular]]))

(defn PhoneListCtrl [$scope, $http]
  (-> $http (.get "phones/phones.edn")
    (.success
      (fn [data]
        (aset $scope "phones" data))))
  (aset $scope "orderProp" "age"))

(defn PhoneDetailCtrl [$scope, $routeParams, $http]
  (-> $http (.get (str "phones/" (aget $routeParams "phoneId") ".edn"))
    (.success
      (fn [data]
        (aset $scope "phone" data)
        (aset $scope "mainImageUrl" (first (:images data))))))
  (aset $scope "setImage" #(aset $scope "mainImageUrl" %)))
```

filters.cljs

```
(ns acute.tutorial.filters
  (:require [acute :refer [angular]]))

(-> angular
  (.module "phonecatFilters" [])
  (.filter "checkmark" (fn [] #(if % "\u2713" "\u2718"))))
```



Who needs JSON?



app.cljs

controllers.cljs

filters.cljs

motorola-xoom.edn

```
{:camera {:features ["Flash" "Video"], :primary "5.0 megapixels"},
:sizeAndWeight
{:dimensions ["249.0 mm (w)" "168.0 mm (h)" "12.7 mm (d)"],
 :weight "726.0 grams"},
:display
{:screenResolution "WXGA (1200 x 800)",
 :screenSize "10.1 inches",
 :touchScreen true},
:storage {:flash "32000MB", :ram "1000MB"},
:images
["img/phones/motorola-xoom.0.jpg"
 "img/phones/motorola-xoom.1.jpg"
 "img/phones/motorola-xoom.2.jpg"],
:name "MOTOROLA XOOM™",
:battery
{:standbyTime "336 hours",
 :talkTime "24 hours",
 :type "Other (3250 mAH)"},
:connectivity
{:bluetooth "Bluetooth 2.1",
 :cell "CDMA 800 /1900 LTE 700, Rx diversity in all bands",
 :gps true,
 :infrared false,
 :wifi "802.11 a/b/g/n"},
:availability ["Verizon"],
:hardware
{:accelerometer true,
 :audioJack "3.5mm",
 :cpu "1 GHz Dual Core Tegra 2",
 :fmRadio false,
 :physicalKeyboard false,
 :usb "USB 2.0"},
:android {:os "Android 3.0", :ui "Android"},
:additionalFeatures
```

Search:

motor

Sort by:

Newest

Motorola XOOM™ with Wi-Fi

The Next, Next Generation Experience the future with Motorola XOOM with Wi-Fi, the world's first tablet powered by Android 3.0 (Honeycomb).

MOTOROLA XOOM™

The Next, Next Generation Experience the future with MOTOROLA XOOM, the world's first tablet powered by Android 3.0 (Honeycomb).

MOTOROLA ATRIX™ 4G

MOTOROLA ATRIX 4G the world's most powerful smartphone.

DROID™ 2 Global by Motorola

The first smartphone with a 1.2 GHz processor and global capabilities.

DROID™ Pro by Motorola

The next generation of DOES.

MOTOROLA BRAVO™ with MOTOBLUR™

An experience to cheer about.

Caveats / Wishlist / Future:

* At mercy of ClojureScript Compiler
    + constructor interception
        - cljs maps -> js objects
        - vectors with upper bounds
        - advanced compilation
    + non-enumerated members
    + key collisions -> "keys"
* Parts of array interface currently stubbed
* Configurable: Turn off mutation or allow lazy seqs
* Embrace and extend pattern
* Inversion: JavaScript for ClojureScript libraries

Go find the best JavaScript libraries and enjoy them...

without writing a line of JavaScript.

Metaprogramming Polyfill:
Feed Clojure Data to your JavaScript Libraries

Thanks!

Tom White
@dribnet