@Nikitonsky

js.cljs — datascript

js.cljs

```clojure
(ns ^:no-doc datascript.js
  (:refer-clojure :exclude [filter])
  (:require
   [cljs.reader]
   [goog.object :as go]
   [clojure.walk :as walk]
   [datascript.core :as d]))

;; Conversions

(defn- keywordize [s]
  (if (and (string? s) (= (subs s 0 1) ":"))
    (keyword (subs s 1))
    s))

(defn- schema->clj [schema]
  (->> (js->clj schema)
       (reduce-kv
         (fn [m k v] (assoc m k (walk/postwalk keywordize v))) {})))

(declare entities->clj)

(defn- entity-map->clj [e]
  (walk/postwalk
    (fn [form]
      (if (and (map? form) (contains? form ":db/id"))
        (-> form
            (dissoc ":db/id")
            (assoc  :db/id (get form ":db/id")))
        form))
    e))
```

Line 8, Column 1 master 2 UTF-8 Unix Spaces: 2 ClojureC

— Font

— Color scheme

— Clojure grammar

— Clojure formatting

— ~~REPL~~

What is this talk about?

Many small ideas unified by a common theme:
“Tools for programming in Clojure”

Won’t help you write better code. Might make you suffer less during coding though.

This talk is about ideas, not implementation.

Is this a design talk?

Yes.

In a sense.

Part I. The Font

FIRA CODE

FIRA CODE

Font: Fira Code

A font with ligatures

Font: Fira Code

~~A font with ligatures~~
Font aware of context: programming

Idea: ligatures

e take look at figure 1 at the bo

↓

e take look at figure 1 at the bo

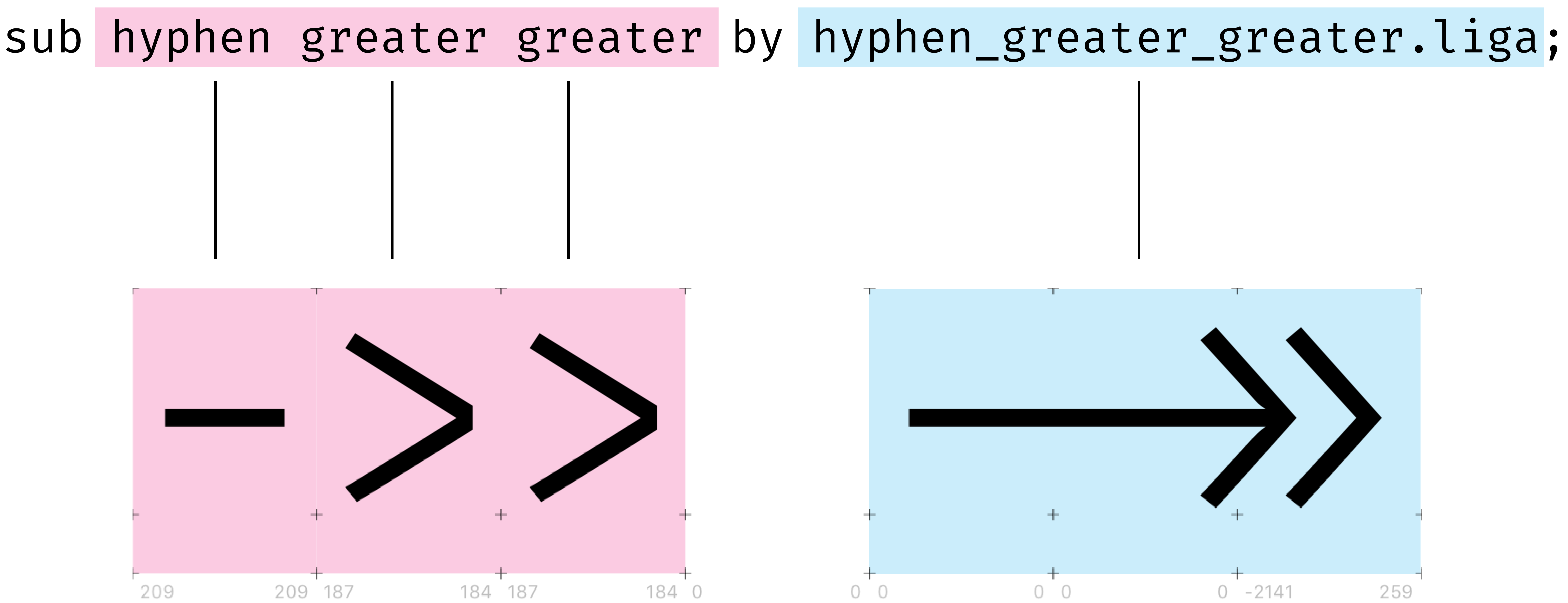
Idea: ligatures

```
(->> coll (filter #(>= % 0)))
```

↓ ↓ ↓

```
(⟶» coll (filter #( ≥ % 0)))
```

Idea: ligatures

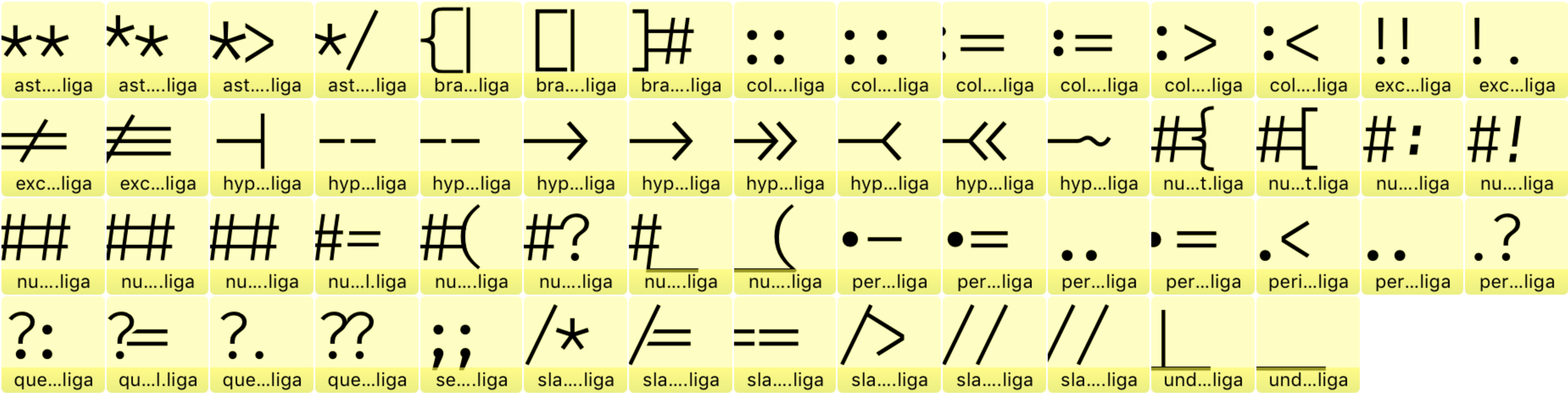


Clojure ligatures

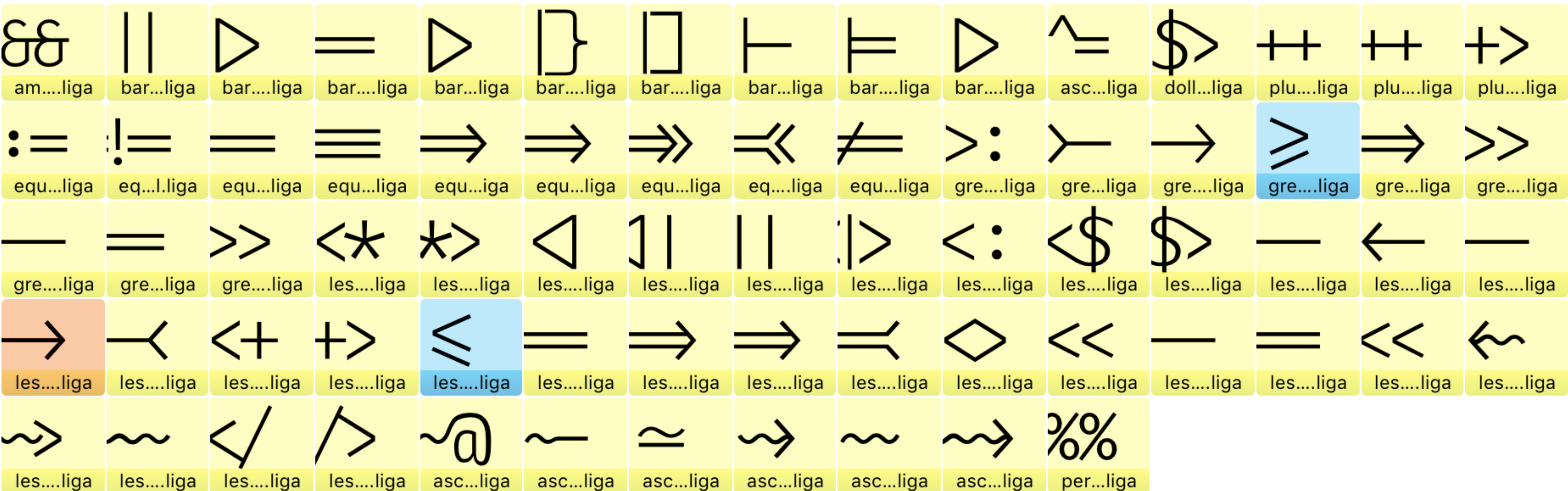
```
->   ->>
#[   #{   #(
#_   #?   ~@
::   .-   ..
<=   >=   ==
```

Idea: language agnostic

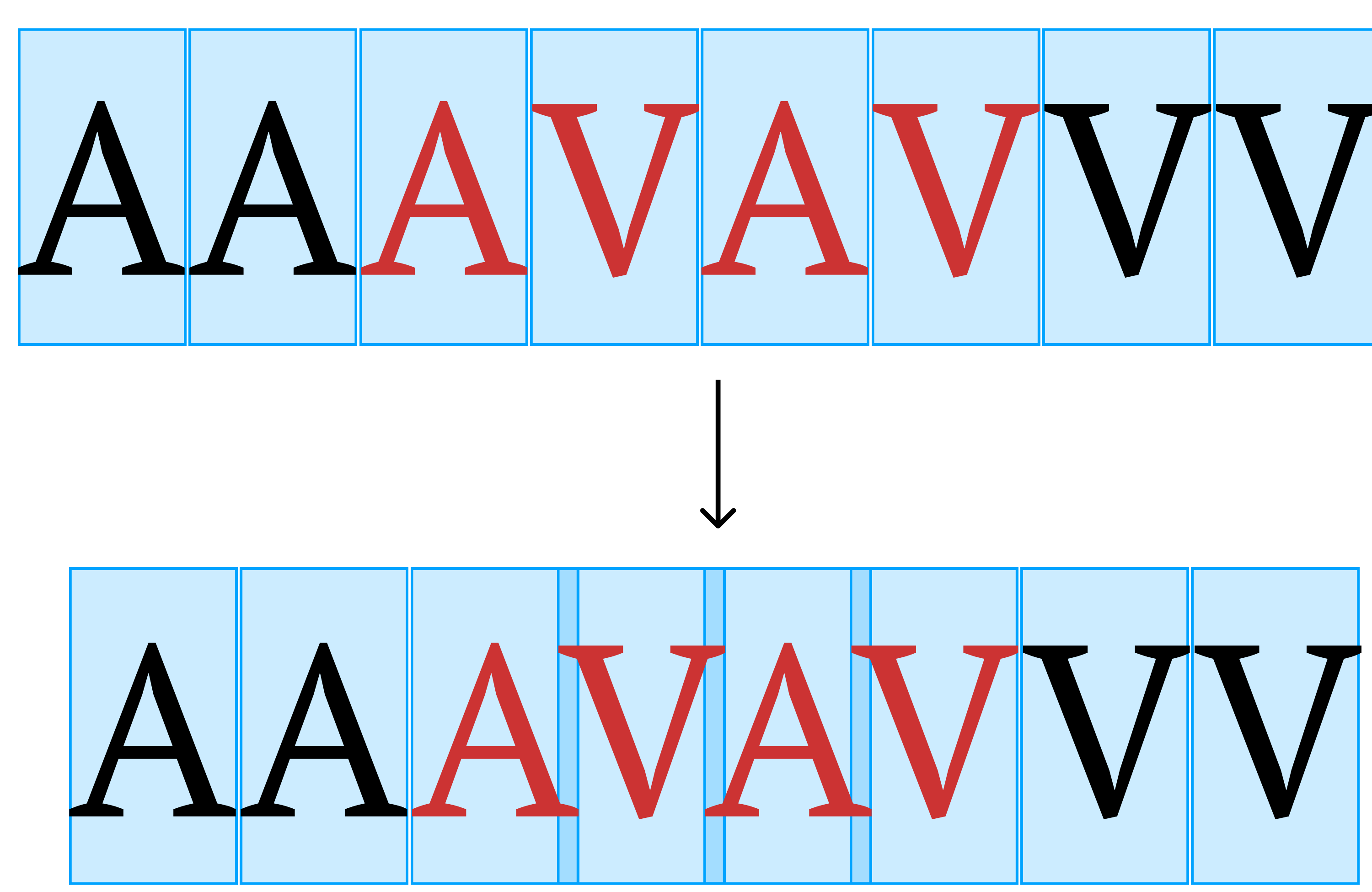
Punctuation



Symbol



Idea: fix spacing



Idea: fix spacing

AAAVAVVV

↓

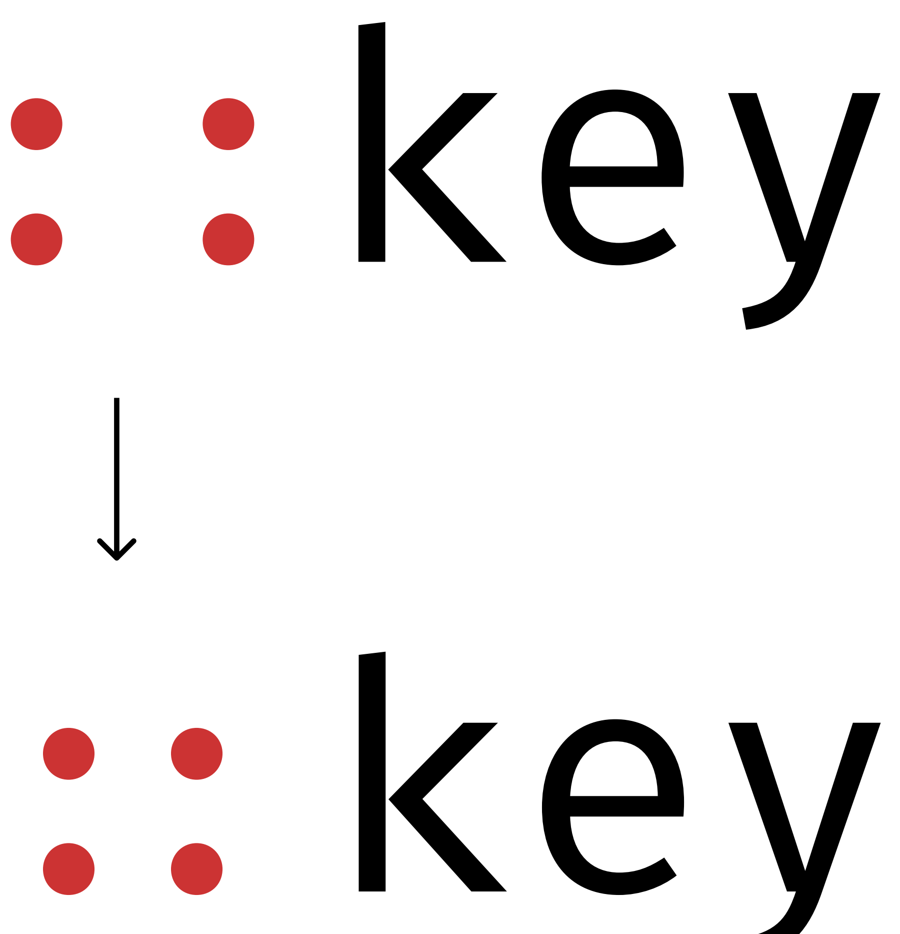
AAAVAVVV

Idea: fix spacing

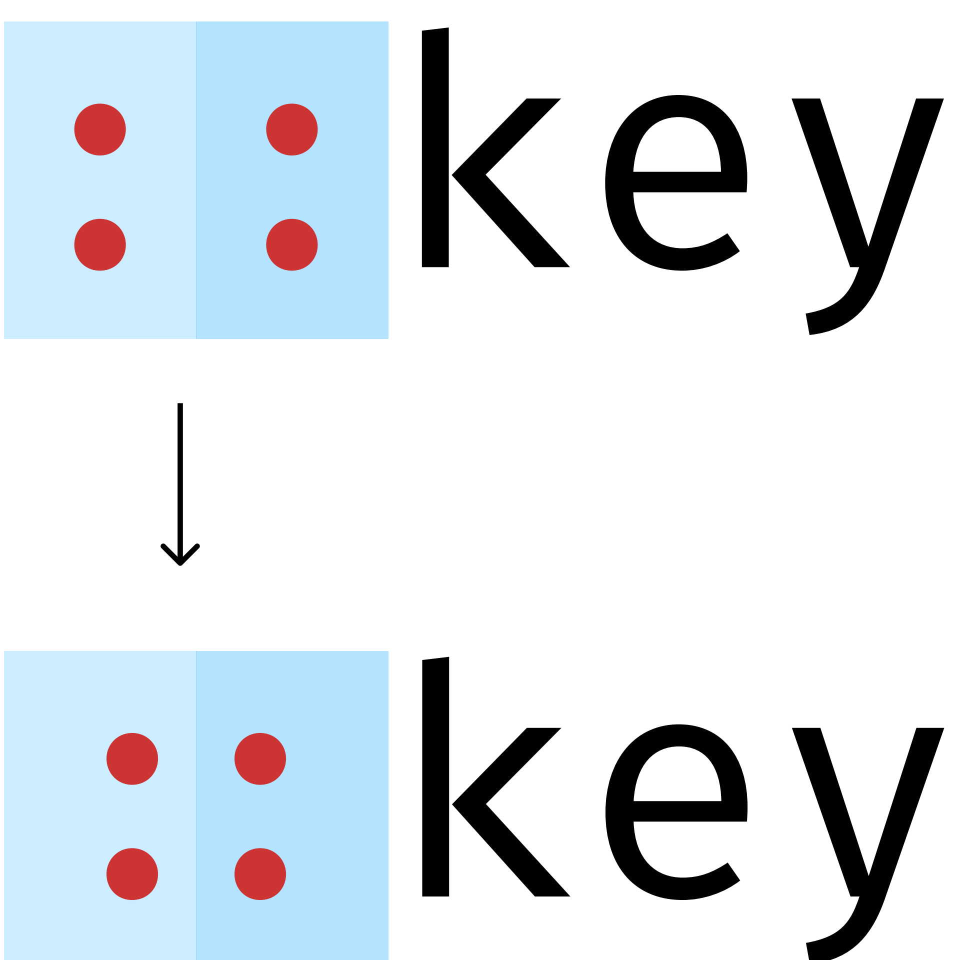
```
-A  -C  -G  -J  -O  -Q  -S  -T  -V  -W  -X  -Y  -c  -d  -e  -g  -o  -q  -s  -x  -Æ  -Ø  -Œ  -Ø  -œ  -Å  A'  A-  AT  AU  AV  AW  AY  A'  Ac  Ad  Ae  Af  Ao
D.  DA  DV  DW  DY  D'  D"  D,  D„  D"  DÅ  F'  F,  F-  F.  F:  F;  FA  Fa  Fe  Fi  Fo  Fr  Fu  Fy  F«  F‹  F›  F,  F„  F"  F»  FÆ  FØ  Fœ  FÅ  G'  G,  G-
KØ  KŒ  KØ  Kœ  KÅ  L'  L-  LT  LU  LV  LW  LY  L'  Ly  L"  L"  M,  M.  M:  M;  M'  M"  N,  N.  N:  N;  N'  N"  N,  N„  O'  O,  O-  O.  O:  O;  OX  OY  O,
RW  RY  R'  Ry  R"  R"  S,  S-  S.  T,  T-  T.  T:  T;  TA  Ta  Tc  Te  Ti  To  Tr  Ts  Tu  Tw  Ty  T«  T‹  T›  T,  T„  T»  TÆ  TØ  Tœ  TÅ  U,  U-  U.  U:
W,  W-  W.  W:  W;  WA  Wa  We  Wi  Wo  Wr  Wu  Wy  W«  W‹  W›  W,  W„  W"  W»  WÆ  WØ  Wœ  WÅ  X-  XA  XC  XO  X'  Xe  X"  X«  X‹  X,  X„  XØ  XŒ  XÅ  Y,
'T  'V  'W  'X  'Y  'b  'd  'g  'h  'k  'l  'q  'r  'v  'w  'y  'Æ  'Ø  'Œ  'Å  a'  a'  a"  a"  b'  b-  b'  b"  b,  b„  b"  e-  ex  f'  f,  f.  f'  f"  f«
p'  p"  p,  p„  p"  q,  q„  r'  r,  r-  r.  r«  r‹  r,  r„  r"  s-  s'  s"  t'  t'  t"  t,  t„  t"  u'  u"  v'  v,  v.  v,  v„  v"  w'  w,  w-  w.  w'  w"
"w  "y  "Æ  "Ø  "Œ  "Å  «J  «T  «V  «W  «Y  «Æ  ‹J  ‹T  ‹V  ‹W  ‹Y  ‹Æ  ›A  ›J  ›T  ›U  ›V  ›W  ›X  ›Y  ›Å  fi'  fi"  ,A  ,C  ,G  ,J  ,O  ,Q  ,T  ,U  ,V  ,W
„ß  „Å  »A  »J  »T  »U  »V  »W  »X  »Y  »Å  Æ'  Æ-  Æ"  Ø'  Ø,  Ø-  Ø.  Ø:  Ø;  ØX  ØY  Ø,  Ø„  Ø"  Æ-  Æx  Ø-  Ø'  Øx  Ø"  œ-  œx  ß-  ß'  ß"  Å'  Å-  ÅT
```

```
-Ø  -œ  -Å  A'  A-  AT  AU  AV  AW  AY  A'  Ac  Ad  Ae  Af  Ao  Aq  At  Au  Av  Aw  Ay  A"  A«  A‹  Afi  Afl  A,  A„  A"  AØ  Aœ  B'  B,  B-  B.  BV  BW  BY  B,  B„  B"  C'  C-  C"  D'  D,  D-
Fu  Fy  F«  F‹  F›  F,  F„  F"  F»  FÆ  FØ  Fœ  FÅ  G'  G,  G-  G.  G"  J'  J,  J-  J.  J:  J;  JA  J,  J„  J"  JÅ  K'  K-  KA  KC  KO  KU  KW  KY  K'  Ke  Ko  Ku  Ky  K"  K«  K‹  K,  K„  K"
N.  N:  N;  N'  N"  N,  N„  O'  O,  O-  O.  O:  O;  OX  OY  O,  O„  O"  P'  P,  P-  P.  P:  P;  PA  P'  Pa  Pe  Po  Ps  P"  P«  P‹  P›  P,  P„  P"  P»  PÆ  PØ  Pœ  PÅ  Q,  Q-  Q.  R'  RT  RV
Tw  Ty  T«  T‹  T›  T,  T„  T»  TÆ  TØ  Tœ  TÅ  U,  U-  U.  U:  U;  UA  U«  U‹  U,  U„  UÅ  V'  V,  V-  V.  V:  V;  VA  Va  Ve  Vi  Vo  Vu  Vy  V«  V‹  V›  V,  V„  V"  V»  VÆ  VØ  Vœ  VÅ  W'
WA  X-  XA  XC  XO  X'  Xe  X"  X«  X‹  X,  X„  XØ  XŒ  XÅ  Y,  Y-  Y.  Y:  Y;  YA  YC  YO  Ya  Ye  Yi  Yo  Yu  Y«  Y‹  Y›  Y,  Y„  Y»  YØ  YŒ  YÆ  YØ  Yœ  YÅ  Z,  Z-  Z.  Z:  Z;  'A  'O  'Q
a"  b'  b-  b'  b"  b,  b„  b"  e-  ex  f'  f,  f.  f'  f"  f«  f‹  f,  f„  f"  g,  g-  g.  h'  h'  h"  h"  i'  i"  j'  j"  k-  k,  k„  m'  m'  m"  m"  n'  n'  n"  n"  o-  o'  ox  o"  p'  p-
t„  t"  u'  u"  v'  v,  v.  v,  v„  v"  w'  w,  w-  w.  w'  w"  w,  w„  w"  x-  xc  xe  xo  xØ  xœ  y'  y,  y.  y,  y„  y"  "A  "O  "Q  "T  "V  "W  "X  "Y  "b  "d  "g  "h  "k  "l  "q  "r  "v
›W  ›X  ›Y  ›Å  fi'  fi"  ,A  ,C  ,G  ,J  ,O  ,Q  ,T  ,U  ,V  ,W  ,X  ,Y  ,j  ,t  ,u  ,v  ,w  ,Æ  ,Ø  ,Œ  ,ß  ,Å  „A  „C  „G  „J  „O  „Q  „T  „U  „V  „W  „X  „Y  „j  „t  „u  „v  „w  „Æ  „Ø  „Œ
Ø„  Ø"  Æ-  Æx  Ø-  Ø'  Øx  Ø"  œ-  œx  ß-  ß'  ß"  Å'  Å-  ÅT  ÅU  ÅV  ÅW  ÅY  Å'  Åc  Åd  Åe  Åf  Åo  Åq  Åt  Åu  Åv  Åw  Åy  Å"  Å«  Å‹  Åfi  Åfl  Å,  Å„  Å"  ÅØ  Åœ  Å'  Å'  Å"  Å"
```

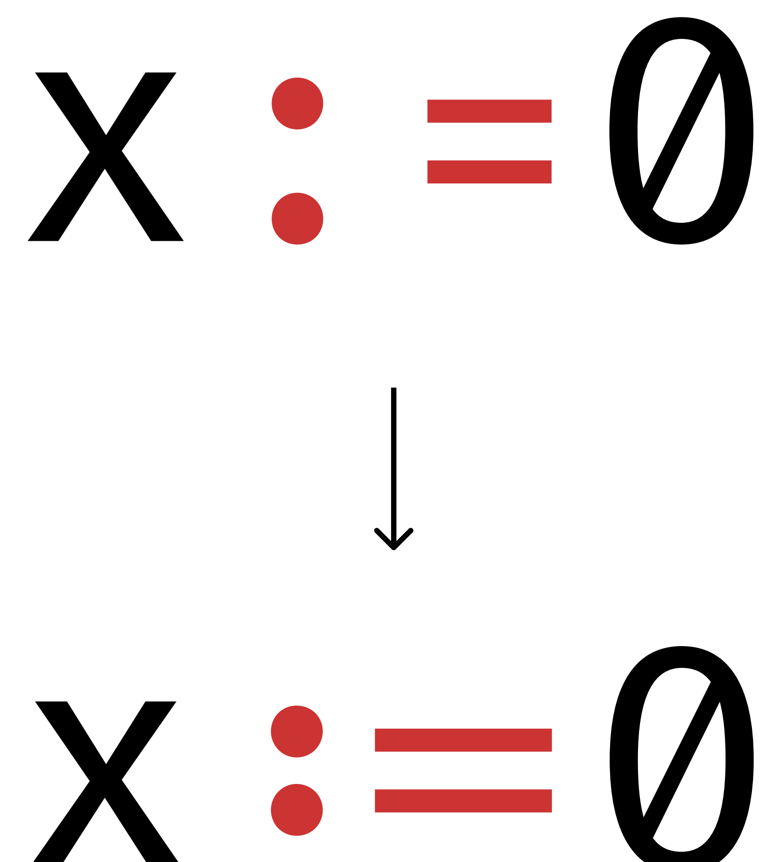
Idea: fix spacing



Idea: fix spacing



Idea: fix align



Idea: fix align

```
*ptr->x
```

Idea: contextual alignment

:= =:=
9:0 a:e
C-O c-o
*RS *rs
0xFF

→

:= =:=
9:0 a:e
C-O c-o
*RS *rs
0xFF

Idea: contextual alignment

WWW → WWW

Bonus: group parens

...)
...))
...)))
...))))
...)))))
...))))))

→

...)
...)❷
...)❸
...)❹
...)❺
...)5+

Fira Code

github.com/tonsky/FiraCode

End of Part I

Part II. The colors

Can't highlight everything

```
fun main() {
    nodejs { this: NodeJSBlock
        val downloadPath : File  by bind(File( pathname: "download"))
        if (downloadPath.exists()) downloadPath.deleteRecursively()

        val r: MemoryUsage = eval( javascript: "process.memoryUsage()")
        println("rss is ${r.rss()}, heapTotal is ${r.heapTotal()}")

        val callback : Consumer<Map<String, Any?>>  by bind(Consumer { m: Map<String, Any?> ->
            val map : DatConnection  = m.asValue().cast<DatConnection>()
            val host: String = map.host()
            val port: Short = map.port()
            val type: String = map.type()
            println("""New connection to $host:$port using $type""")
        })

        run( javascript: """
            var Dat = require('dat-node');
            D   (downloadPath.getName(), { key: "778f8d955175c92e4ced5e4f5563f69bfec0c86cc6
                                     ow err;
                console.log("Joined DAT network!");
                let network = dat.joinNetwork();    // Downloads files automatically.
```

'var' used instead of 'let' or 'const' more... (⌘F1)

Can’t highlight everything

```
fun main() {
    nodejs {
        val downloadPath by bind(File("download"))
        if (downloadPath.exists()) downloadPath.deleteRecursively()

        val r: MemoryUsage = eval("process.memoryUsage()")
        println("rss is ${r.rss()}, heapTotal is ${r.heapTotal()}")

        val callback by bind(Consumer { m: Map<String, Any?> ->
            val map = m.asValue().cast<DatConnection>()
            val host: String = map.host()
            val port: Int = map.port()
            val type: String = map.type()
            println("""New connection to $host:$port using $type""")
        })

        run("""
            let Dat = require('dat-node');
            Dat(downloadPath.getName(), { key: "778f8d955175c92e4ced5e4f5563f69bfec0c86cc6f670352c4579
                if (err) throw err;
                console.log("Joined DAT network!");
                let network = dat.joinNetwork();    // Downloads files automatically.
```

Pro-blem: Too many rules

- Editor
  - General
  - Font
  - Color Scheme
    - General
    - Language Defaults
    - Color Scheme Font
    - Console Font
    - Console Colors
    - Custom
    - Debugger
    - Diff & Merge
    - VCS
    - Java
    - Android Logcat
    - Dart
    - EditorConfig
    - Flutter Log
    - Groovy
    - HTML
    - JSON
    - Kotlin
    - Properties
    - RegExp
    - XML
    - XPath
    - XSLT
    - YAML
    - By Scope
  - Code Style
  - Inspections
  - File and Code Templates
  - File Encodings
  - Live Templates

- Annotations
  - Annotation attribute name
  - Annotation name
- Braces and Operators
  - Braces
  - Brackets
  - Comma
  - Dot
  - Operator sign
  - Parentheses
  - Semicolon
- Class Fields
  - Constant (static final field)
  - Constant (static final imported field)
  - Instance field
  - Instance final field
  - Static field
  - Static imported field
- Classes and Interfaces
  - Abstract class
  - Anonymous class
  - Class
  - Enum
  - Interface
- Comments
  - Block comment
  - JavaDoc
    - Markup
    - Tag
    - Tag value
    - Text
  - Line comment
- Keyword
- Methods
  - Abstract method
  - Constructor call
  - Constructor declaration
  - Inherited method
  - Method call
  - Method declaration

Bold Italic

Foreground

Background

Error stripe mark

Effects 909090

Underscored

Inherit values from:

Identifiers→Reassigned local variable (Language Defaults)

something *expression*

2
HOUR PARKING
6 AM - 5 AM
MON THRU FRI
PARKING
OF VEHICLES
ONLY
AUTHORIZED
PARKING
PERMITTED
ANYTIME
AFTER MIDNIGHT

Problem: to_O MUUCH dEcOrAtIoN!!!

```
updateObject(type, id, object) {
  assertNoIdsArePresent(object);
  assertApiArgumentTypes(apiCommandsSchema.update, {type, id, object}, "updateObject command");
  return makeApiCommand("data/create-or-update", {type, object: {...object, ...makeIdSpecifierObject(id)}});
},

deleteObject(type, id) {
  assertApiArgumentTypes(apiCommandsSchema.deleteObject, {type, id}, "deleteObject command");
  return makeApiCommand("data/delete", {type, object: makeIdSpecifierObject(id)});
},

attachAttribute(type, id, attributeToAttach, affectedIds) {
  assertApiArgumentTypes(apiCommandsSchema.attachOrDetachAttribute, {
    type,
    id,
    attributeToAttach,
    affectedIds,
  }, "attachAttribute command");
  console.assert(affectedIds.length > 0, "affectedIds must not be empty");
  return makeApiCommand("data/attach-attribute", {
    type,
    object: makeIdSpecifierObject(id),
    attribute: attributeToAttach,
    "attribute-objects": affectedIds.map(makeIdSpecifierObject),
  });
},
```

Problem: to_O MUUCH dEcOrAtIoN!!!

```
updateObject(type, id, object) {
  assertNoIdsArePresent(object);
  assertApiArgumentTypes(apiCommandsSchema.update, {type, id, object}, "updateObject command");
  return makeApiCommand("data/create-or-update", {type, object: {...object, ...makeIdSpecifierObject(id)}});
},

deleteObject(type, id) {
  assertApiArgumentTypes(apiCommandsSchema.deleteObject, {type, id}, "deleteObject command");
  return makeApiCommand("data/delete", {type, object: makeIdSpecifierObject(id)});
},

attachAttribute(type, id, attributeToAttach, affectedIds) {
  assertApiArgumentTypes(apiCommandsSchema.attachOrDetachAttribute, {
    type,
    id,
    attributeToAttach,
    affectedIds,
  }, "attachAttribute command");
  console.assert(affectedIds.length > 0, "affectedIds must not me empty");
  return makeApiCommand("data/attach-attribute", {
    type,
    object: makeIdSpecifierObject(id),
    attribute: attributeToAttach,
    "attribute-objects": affectedIds.map(makeIdSpecifierObject),
  });
},
```

Idea: few rules you will remember

Compile-time constants

“String-like” (string, regexps)

Declarations

;; Comments

(punctuation)

Idea: highlight comments

db.cljc — datascript

db.cljc

```
1144
1145         (map? entity)
1146         (let [old-eid (:db/id entity)]
1147           (cond+
1148             ;; :db/current-tx / "datomic.tx" ⇒ tx
1149             (tx-id? old-eid)
1150             (let [id (current-tx report)]
1151               (recur (allocate-eid report old-eid id)
1152                      (cons (assoc entity :db/id id) entities)))
1153
1154             ;; lookup-ref ⇒ resolved | error
1155             (sequential? old-eid)
1156             (let [id (entid-strict db old-eid)]
1157               (recur report
1158                      (cons (assoc entity :db/id id) entities)))
1159
1160             ;; upserted ⇒ explode | error
1161             :let [upserted-eid (upsert-eid db entity)]
1162
1163             (some? upserted-eid)
1164             (if (and (tempid? old-eid)
1165                      (contains? tempids old-eid)
1166                      (not= upserted-eid (get tempids old-eid)))
1167               (retry-with-tempid initial-report report initial-es old-eid upserted-eid)
1168               (recur (allocate-eid report old-eid upserted-eid)
1169                      (concat (explode db (assoc entity :db/id upserted-eid)) entities)))
1170
1171             ;; resolved | allocated-tempid | tempid | nil ⇒ explode
1172             (or (number? old-eid)
1173                 (nil?    old-eid)
1174                 (string? old-eid))
1175             (let [new-eid (cond
1176                             (nil? old-eid)    (next-eid db)
```

Line 1113, Column 1 | master 2 | UTF-8 | Unix | Spaces: 2 | Clojure

Idea: highlight top-level declarations

(ns cljfmt.core

```clojure
266
267 (defn- make-indenter [[key opts] alias-map]
268   (apply some-fn (map (partial indenter-fn key alias-map) opts)))
269
270 (defn- indent-order [[key _]]
271   (cond
272     (and (symbol? key) (namespace key)) (str 0 key)
273     (symbol? key) (str 1 key)
274     (pattern? key) (str 2 key)))
275
276 (defn- custom-indent [zloc indents alias-map]
277   (if (empty? indents)
278     (list-indent zloc)
279     (let [indenter (->> (sort-by indent-order indents)
280                         (map #(make-indenter % alias-map))
281                         (apply some-fn))]
282       (or (indenter zloc)
283           (list-indent zloc)))))
284
285 (defn- indent-amount [zloc indents alias-map]
286   (let [tag (-> zloc z/up z/tag)
287         gp  (-> zloc z/up z/up)]
288     (cond
289       (reader-conditional? gp) (coll-indent zloc)
290       (#{:list :fn} tag)       (custom-indent zloc indents alias-map)
291       (= :meta tag)            (indent-amount (z/up zloc) indents alias-map)
292       :else                    (coll-indent zloc))))
293
294 (defn- indent-line [zloc indents alias-map]
295   (let [width (indent-amount zloc indents alias-map)]
296     (if (> width 0)
297       (zip/insert-right zloc (whitespace width))
298       zloc)))
299
300 (defn indent
301   ([form]
302    (indent form default-indents))
303   ([form indents]
304    (transform form edit-all should-indent? #(indent-line % indents {})))
305   ([form indents alias-map]
306    (transform form edit-all should-indent? #(indent-line % indents alias-map))))
307
308 (defn reindent
309   ([form]
310    (indent (unindent form)))
311   ([form indents]
312    (indent (unindent form) indents))
313   ([form indents alias-map]
314    (indent (unindent form) indents alias-map)))
315
316 (defn root? [zloc]
317   (nil? (zip/up zloc)))
318
319 (defn final? [zloc]
320   (and (nil? (zip/right zloc)) (root? (zip/up zloc))))
```

Line 318, Column 1

Alabaster

- ✓ Sublime Text 3
- ✓ VS Code
- ✓ Vim
- ✓ IntelliJ
- ✓ Light Table

playground.clj — sublime-scheme-alabaster

playground.clj

```
(ns playground
  (:require
    [clojure.string :as str]))

(defn clojure-function [args]
  (let [string   "multiline\nstring"
        regexp   #"regexp"
        number   100.000
        booleans [false true]
        keyword  ::the-keyword]
    ;; this is comment
    (if true
      (->>
        (list [vector] {:map map} #{'set})))))

;; highlighted line
;; find highlights:
(def some-var)
(defonce another-var)
```

.* Aa “” string Find Find Prev Find

2 of 3 matches master 1 UTF-8 Unix Spaces: 2

End of Part II

Part III. The grammar

Problem: approximate

```
(def x [
  a'                   ; symbol with '
  :1                   ; keyword starting with digit
  :a:b                 ; keyword with color
  :абв                 ; cyrillic keyword
  \o377                ; octal char
  100N                 ; BigInt literal
  01/2                 ; Ratio
  #datascript/DB {}    ; namespaced reader tag
  ^:kw ^"String" sym   ; multiple metas
  (rum/defc label []) ; namespaced def*
  @ , *atom            ; whitespace between @ and symbol
  `(quote ~[])         ; quote-unquote
])
```

Problem: approximate

```
(def x [
  a'                   ; symbol with '
  :1                   ; keyword starting with digit
  :a:b                 ; keyword with color
  :абв                 ; cyrillic keyword
  \o377                ; octal char
  100N                 ; BigInt literal
  01/2                 ; Ratio
  #datascript/DB {}    ; namespaced reader tag
  ^:kw ^"String" sym   ; multiple metas
  (rum/defc label [])  ; namespaced def*
  @ , *atom            ; whitespace between @ and symbol
  `(quote ~[])         ; quote-unquote
])
```

Problem: approximate

```
(def x [
  a'                   ; symbol with '
  :1                   ; keyword starting with digit
  :a:b                 ; keyword with color
  :абв                 ; cyrillic keyword
  \o377                ; octal char
  100N                 ; BigInt literal
  01/2                 ; Ratio
  #datascript/DB {}    ; namespaced reader tag
  ^:kw ^"String" sym   ; multiple metas
  (rum/defc label []) ; namespaced def*
  (defmethod clojure.test/report :error [m]) ; def with namespaced symbol
  @ , *atom            ; whitespace between @ and symbol
  `(quote ~[])         ; quote-unquote
])
```

Idea: pedantically follow the spec

Precise, not approximate

Helpful, not confusing

Parse what should be parsed: no false negatives

Do not parse what shouldn’t: no false positives

Test thoroughly

Don’t be lazy

Idea: report errors

```
#"\t \n \r \f \a \e \cC \d \D \h \H \s \S \v \V \w \W"

#"\\ \07 \077 \0377 \xFF \uFFFF \x{0} \x{FFFFF} \x{10FFFF} \N{white smiling face}"

#"\y \x \uABC \p{Is Latin} \k<1gr> "

#"(x|(\(\||[)|])))"
```

sublime-clojure

github.com/tonsky/sublime-clojure

```
(def x [
  a'                      ; symbol with '
  :1                      ; keyword starting with digit
  :a:b                    ; keyword with color
  :абв                    ; cyrillic keyword
  \o377                   ; octal char
  100N                    ; BigInt literal
  01/2                    ; Ratio
  #datascript/DB {}       ; namespaced reader tag
  ^:kw ^"String" sym      ; multiple metas
  (rum/defc label [])     ; namespaced def*
  @ , *atom               ; whitespace between @ and symbol
  `(quote ~[])            ; quote-unquote
])
```

End of Part III

Part IV. The format

One Clojure formatter

for everyone to agree on

gofmt

De facto Go formatter

Mandatory for all published code

No knobs

Good enough

“Gofmt's style is nobody’s favorite, yet gofmt is everybody’s favorite.”

Why another formatter?

Clojure Style Guide

Why another formatter?

Clojure Style Guide

cljfmt

Why another formatter?

Clojure Style Guide

cljfmt

emacs

Why another formatter?

Clojure Style Guide

cljfmt

emacs

zprint

Why another formatter?

Clojure Style Guide

cljfmt

emacs

zprint

fipp

Problem: too vague

Optionally omit the new line between the function name and argument vector

Optionally omit the new line between the argument vector and a short function body

Consider enhancing the readability of map literals via **judicious** use of commas and line breaks.

An exception to the rule is the grouping of related defs together.

Where feasible, avoid making lines longer than 80 characters.

An exception can be made to indicate grouping of pairwise constructs as found in e.g. let and cond.

Problem: too specific

68 lines (67 sloc) | 2.05 KB

Raw | Blame | History

```
1   {alt!          [[:block 0]]
2    alt!!         [[:block 0]]
3    are           [[:block 2]]
4    as->          [[:block 2]]
5    binding       [[:block 1]]
6    bound-fn      [[:inner 0]]
7    case          [[:block 1]]
8    catch         [[:block 2]]
9    comment       [[:block 0]]
10   cond          [[:block 0]]
11   condp         [[:block 2]]
12   cond->        [[:block 1]]
13   cond->>       [[:block 1]]
14   def           [[:inner 0]]
15   defmacro      [[:inner 0]]
16   defmethod     [[:inner 0]]
17   defmulti      [[:inner 0]]
18   defn          [[:inner 0]]
19   defn-         [[:inner 0]]
20   defonce       [[:inner 0]]
21   defprotocol   [[:block 1] [:inner 1]]
22   defrecord     [[:block 2] [:inner 1]]
```

Problem: too specific

```
23  defstruct       [[:block 1]]
24  deftest         [[:inner 0]]
25  deftype         [[:block 2] [:inner 1]]
26  do              [[:block 0]]
27  doseq           [[:block 1]]
28  dotimes         [[:block [illegible]]]
29  doto            [[:block 1]]
30  extend          [[:block 1]]
31  extend-protocol [[:block 1] [:inner 1]]
32  extend-type     [[:block 1] [:inner 1]]
33  fdef            [[:inner 0]]
34  finally         [[:block 0]]
35  fn              [[:inner 0]]
36  for             [[:block 1]]
37  future          [[:block 0]]
38  go              [[:block 0]]
39  go-loop         [[:block 1]]
40  if              [[:block 1]]
41  if-let          [[:block 1]]
42  if-not          [[:block 1]]
43  if-some         [[:block 1]]
44  let             [[:block 1]]
45  letfn           [[:block 1] [:inner 2 0]]
46  locking         [[:block 1]]
47  loop            [[:block 1]]
48  match           [[:block 1]]
49  ns              [[:block 1]]
50  proxy           [[:block 2] [:inner 1]]
51  reify           [[:inner 0] [:inner 1]]
52  struct-map      [[:block 1]]
53  testing         [[:block 1]]
54  thread          [[:block 0]]
55  try             [[:block 0]]
56  use-fixtures    [[:inner 0]]
```

Problem: too specific

```
51    reify            [[:inner 0] [:inner 1]]
52    struct-map       [[:block 1]]
53    testing          [[:block 1]]
54    thread           [[:block 0]]
55    try              [[:block [illegible]]]
56    use-fixtures     [[:inner [illegible]]]
57    when             [[:block 1]]
58    when-first       [[:block 1]]
59    when-let         [[:block 1]]
60    when-not         [[:block 1]]
61    when-some        [[:block 1]]
62    while            [[:block 1]]
63    with-local-vars  [[:block 1]]
64    with-open        [[:block 1]]
65    with-out-str     [[:block 0]]
66    with-precision   [[:block 1]]
67    with-redefs      [[:block 1]]}
```

Problem: too specific

68 lines (67 sloc) | 2.05 KB

Raw | Blame | History

```
{alt!           [[:block 0]]
 alt!!          [[:block 0]]
 are            [[:block 2]]
 as->           [[:block 2]]
 binding        [[:block 1]]
 bound-fn       [[:inner 0]]
 case           [[:block 1]]
 catch          [[:block 2]]
 comment        [[:block 0]]
 cond           [[:block 0]]
 condp          [[:block 2]]
 cond->         [[:block 1]]
 cond->>        [[:block 1]]
 def            [[:inner 0]]
 defmacro       [[:inner 0]]
 defmethod      [[:inner 0]]
 defmulti       [[:inner 0]]
 defn           [[:inner 0]]
 defn-          [[:inner 0]]
 defonce        [[:inner 0]]
 defprotocol    [[:block 1] [:inner 1]]
 defrecord      [[:block 2] [:inner 1]]
 defstruct      [[:block 1]]
 deftest        [[:inner 0]]
 deftype        [[:block 2] [:inner 1]]
 do             [[:block 0]]
 doseq          [[:block 1]]
 dotimes        [[:block 1]]
 doto           [[:block 1]]
 extend         [[:block 1]]
 extend-protocol [[:block 1] [:inner 1]]
 extend-type    [[:block 1] [:inner 1]]
 fdef           [[:inner 0]]
 finally        [[:block 0]]
 fn             [[:inner 0]]
 for            [[:block 1]]
 future         [[:block 0]]
 go             [[:block 0]]
 go-loop        [[:block 1]]
 if             [[:block 1]]
 if-let         [[:block 1]]
 if-not         [[:block 1]]
 if-some        [[:block 1]]
 let            [[:block 1]]
 letfn          [[:block 1] [:inner 2 0]]
 locking        [[:block 1]]
 loop           [[:block 1]]
 match          [[:block 1]]
 ns             [[:block 1]]
 proxy          [[:block 2] [:inner 1]]
 reify          [[:inner 0] [:inner 1]]
 struct-map     [[:block 1]]
 testing        [[:block 1]]
 thread         [[:block 0]]
 try            [[:block 0]]
 use-fixtures   [[:inner 0]]
 when           [[:block 1]]
 when-first     [[:block 1]]
 when-let       [[:block 1]]
 when-not       [[:block 1]]
 when-some      [[:block 1]]
 while          [[:block 1]]
 with-local-vars [[:block 1]]
 with-open      [[:block 1]]
 with-out-str   [[:block 0]]
 with-precision [[:block 1]]
 with-redefs    [[:block 1]]}
```

```
(def zfnstyle
  {"->" :noarg1-body,
   "->>" :force-nl-body,
   ":import" :force-nl-body,
   ":require" :force-nl-body,
   "=" :hang,
   "alt" :pair-fn,
   "and" :hang,
   "apply" :arg1,
   "as->" :arg2,
   "assoc" :arg1-pair,
   "as[illegible]" :arg1
   "[illegible]nding" [illegible]g,
   "[illegible]se" :arg1-p[illegible]-body,
   "[illegible]e-nl,
   "catch" :arg2,
   "cond" :pair-fn,
   "cond-let" :pair-fn,
   "cond->" :arg1-pair-body,
   "condp" :arg2-pair,
   "def" :arg1-body,
   "defc" :arg1-mixin,
   "defcc" :arg1-mixin,
   "defcs" :arg1-mixin,
   "defmacro" :arg1-body,
   "defexpect" :arg1-body,
   "defmethod" :arg2,
   "defmulti" :arg1-body,
   "defn" :arg1-body,
   "defn-" :arg1-body,
   "defproject" [:arg2-pair {:vector {:wrap? false}}],
   "defprotocol" :arg1-force-nl,
   "defrecord" :arg2-extend,
   "deftest" :arg1-body,
   "deftype" :arg2-extend,
   "defui" :arg1-extend,
   "do" :none-body,
   "doseq" :binding,
   "dotimes" :binding,
   "doto" :arg1,
   "extend" :arg1-extend,
   "extend-protocol" :arg1-extend,
   "extend-type" :arg1-extend,
   "fdef" :arg1-force-nl,
   "filter" :arg1,
   "filterv" :arg1,
   "fn" :fn,
   "fn*" :fn,
   "for" :binding,
   "if" :arg1-body,
   "if-let" :binding,
   "if-not" :arg1-body,
   "if-some" :binding,
   "interpose" :arg1,
   "let" :binding,
   "letfn" :binding,
   "loop" :binding,
   "map" :arg1,
   "mapcat" :arg1,
   "match" :arg1-pair-body,
   "matchm" :arg1-pair-body,
   "mapv" :arg1,
   "not=" :hang,
   "ns" :arg1-body,
   "or" :hang,
   "proxy" :arg2-fn,
   "reduce" :arg1,
   "reify" :extend,
   "remove" :arg1,
   "s/def" [:arg1-body {:list {:constant-pair-min 2}}],
   "s/fdef" [:arg1-body {:list {:constant-pair-min 2}}],
   "s/and" :gt2-force-nl,
   "s/or" :gt2-force-nl,
   "some->" :force-nl-body,
   "some->>" :force-nl-body,
   "swap!" :arg2,
   "try" :none-body,
   "when" :arg1-body,
   "when-first" :binding,
   "when-let" :binding,
   "when-not" :arg1-body,
   "when-some" :binding,
   "with-bindings" :arg1,
   "with-bindings*" :arg1,
   "with-local-vars" :binding,
   "with-meta" :arg1-body,
   "with-open" :binding,
   "with-redefs" :binding,
   "with-redefs-fn" :arg1-body})
```

Problem: runtime/whitelisting

```
(when something
  (something-else))
```

but

```
(filter something
        (something-else))
```

Everyone to agree on

Now and in the future

No versions, no patches

Any runtime (jvm, js, python, rust?)

Open world

New libraries

New editors

New language features (cond->, when-some, ?#...)

Idea: remove ALL special cases

As simple as it gets

No exceptions

No whitelisting

No runtime dependency

Few rules, applied uniformly

How few?

1. Align lists starting with symbol with 2 spaces

```
(when something
··body)

(defn f [x]
··body)
```

```
(defn f
··[x]
··body)
```

2. Align rest to the bracket

```
[1 2 3 4
 5 6]

{:key 1
 :also-key 2}

#{a b c d
  e f}
```

```
([x]
 body)
([x y]
 body))

#?(:clj
   (Math/round 1))
```

Bonus: no deep nesting

```
(filter even?
········(range 1 10))
```

```
(filter even?
··(range 1 10))
```

Read more at

tonsky.me/blog/clojurefmt

End of Part IV

Part C. The Conclusion

Writing tools is fun!
Lots to improve.
Huge impact for yourself
Try it!

— Submit a patch to language grammar!
— Remap your keyboard!
— Write a color scheme!
— Add ligatures to a font!

Thank you!

github.com/tonsky
@nikitonsky