



Asynchronous Distributed Multiplexed **REPLs**



REPL



```
python /...
>>> 1 + 1
2
>>> |
```

REPL

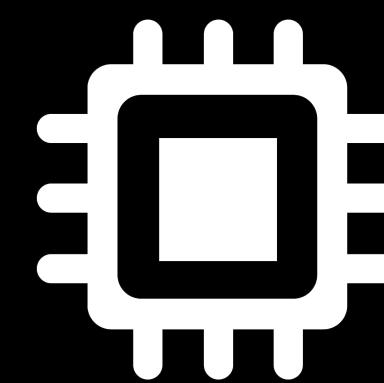
read



A screenshot of a terminal window titled "python /...". It shows the command `>>> 1 + 1` followed by the output `2`. The cursor is at the end of the line `>>> |`.

REPL

print



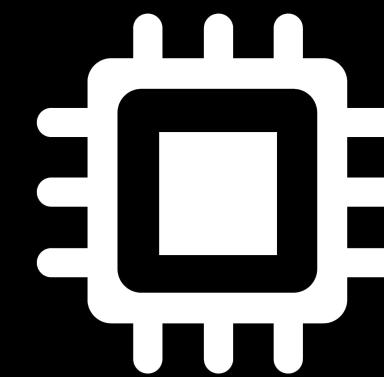
eval



```
python /...
>>> 1 + 1
2
>>> |
```

read

REPL



eval

print



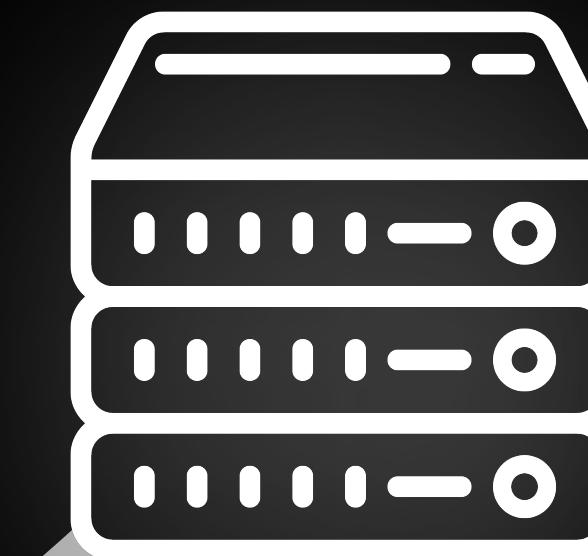
```
python /...
>>> 1 + 1
2
>>> |
```

read

REPL



print



eval

The diagram illustrates the REPL cycle with three main components: **read**, **eval**, and **print**. A central **REPL** icon is connected to a cloud icon labeled **read** on the top right, which points to a code editor window showing Clojure code. The code editor has tabs for **Welcome** and **emacros.cljs**, with **emacros.cljs** being the active tab. The code itself includes macros like `(defmacro numeric [&keys [keyboard] &num]` and `(defmacro opt [&keys [keyboard replace-all!]] &opts)`. Arrows point from the **read** cloud to the code editor and from the code editor to the **eval** component. The **eval** component is represented by a stack of three server icons. Arrows point from the **eval** component to the **print** cloud and from the **print** cloud back to the code editor. The code editor also features icons for **Vim**, **Erlang**, and **VS Code**.

```
8  (defmacro numeric [&keys [keyboard] &num]
9    ... (when (and (:shift keyboard)
10       .... (:control keyboard)
11       .... (= keyboard))
12       .... (swap! num inc))
13      ... (when (and (:shift keyboard)
14         .... (:control keyboard)
15         .... (- keyboard))
16         .... (swap! num dec)))
17     ... @num)
18
19
20 (defmacro opt [&keys [keyboard replace-all!] &opts]
21   (when (and (:shift keyboard)
22             (:control keyboard)
23             ("z" keyboard))
24     (replace-all! ; rotate opts
25       (apply concat ((juxt drop take) 1 opts))))
26   (if (seq opts))
```

read

REPL

print

eval

state of the art

read-eval-print-loop

sync request/response

1:1

ad-hoc editor integration

exception handling

state of the art

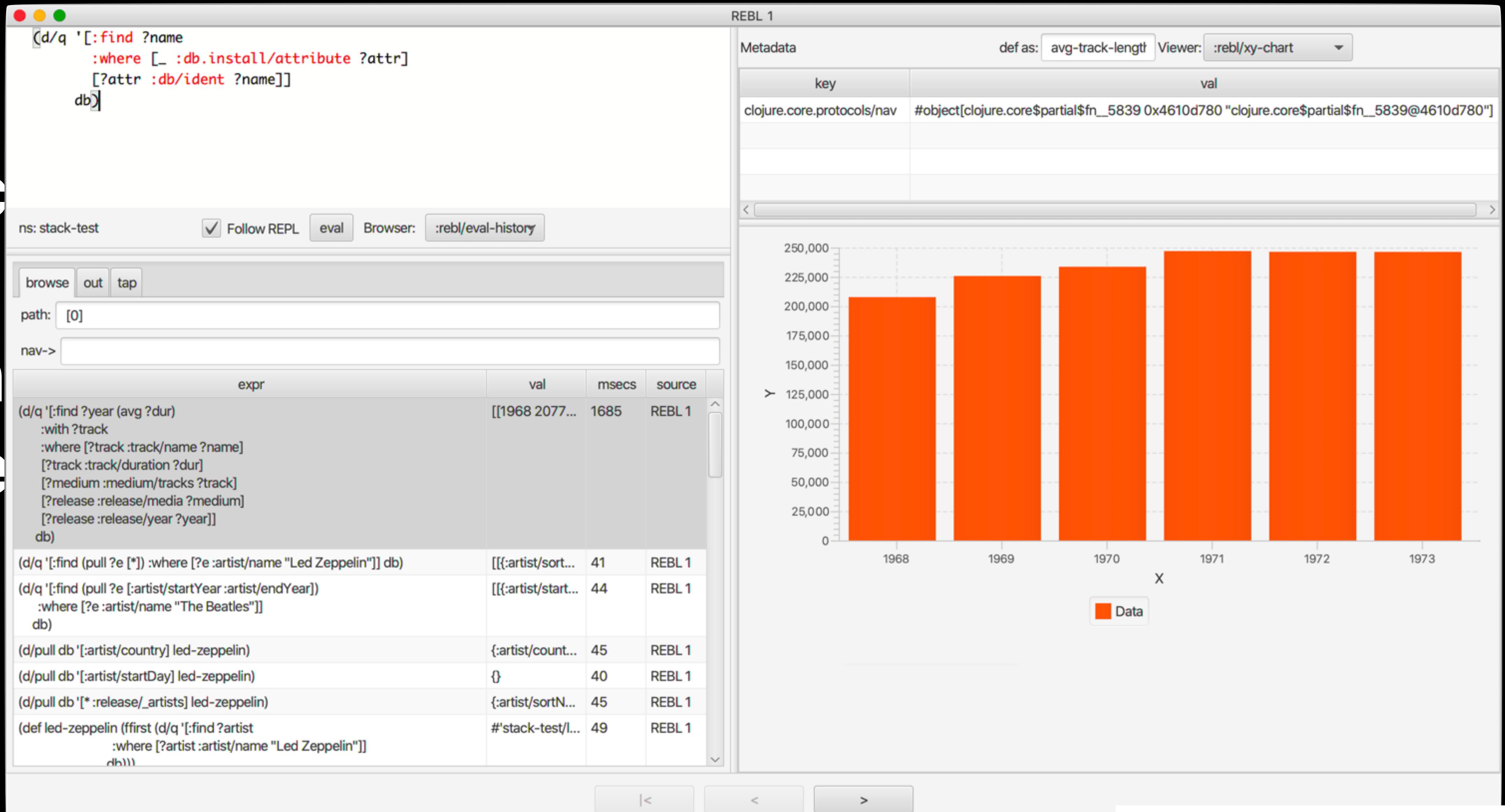
The image is a collage of screenshots from the Glamorous Toolkit environment, showcasing its modular development and analysis platform. It includes:

- Tour:** A treemap visualization of classes and packages.
- Playground:** A live code editor showing examples of GtMondrian and CompiledMethod usage.
- GtMondrian:** A visual interface for creating and manipulating Mondrian-like diagrams.
- Inspector:** A detailed view of a CompiledMethod's source code and bytecode representations.
- Coder:** A code editor for GToolkit-Mondrian examples.
- Slideshow:** A collection of slides titled "glamorous toolkit", "one rendering tree", and "Pharo 101".
- Docs:** A collection of documentation pages including "Glamorous Toolkit", "Inspector", "Playground", "Coder", "Documenter", "Visualizer", "Modifiable development", "Examples", and "More docs ...".
- File Reference:** A file browser showing a directory structure.

Glamorous Toolkit, Smalltalk (Girba+)

state of the art

read
sync
1:1
ad-h
exe



state of the art

```
lem--eval "(lem-lisp-mode:start-listener)"  
CL-USER> (defun hello (name)  
             (print name))  
HELLO  
CL-USER> (hello)  
; No value  
CL-USER> (hello)
```

```
clj::fresh-reagent.handler=>  
(defn leet->1337 [s]  
  (-> s  
        (clojure.string/replace #"l", "1")  
        (clojure.string/replace #"e", "3") => "133t"  
        (clojure.string/replace #"t", "7"))))  
#'fresh-reagent.handler/leet->1337  
clj::fresh-reagent.handler=>  
(leet->1337 "leet")
```

Calva, Clojure (Strömberg+)

```
* *lisp-repl* (lisp-repl listener) (8, 0) [CL-USER sbcl:SELF] All  
invalid number of arguments: 0  
  [Condition of type SB-INT:SIMPLE-PROGRAM-ERROR]  
  
Restarts:  
0: [REPLACE-FUNCTION] Call a different function with the same arguments  
1: [CALL-FORM] Call a different form  
2: [RETRY] Retry SLIME REPL evaluation request.  
3: [ABORT] Return to slldb level 1.  
4: [REPLACE-FUNCTION] Call a different function with the same arguments  
5: [CALL-FORM] Call a different form  
6: [RETRY] Retry SLIME REPL evaluation request.  
7: [*ABORT] Return to SLIME's top level.  
8: [ABORT] abort thread (#<THREAD "repl-thread" RUNNING {1006710413}>)
```

```
Backtrace:  
0: (HELLO) [external]  
1: (SB-INT:SIMPLE-EVAL-IN-LEXENV (HELLO))  
2: (EVAL (HELLO))  
--more--
```

SLIME, Common Lisp (Eller+)

state of the art

missing

read-eval-print
sync req/res

1:1

ad-hoc editor
exceptions

pull, as-you-type
async streams

1:n

explicit textual state
fault handling
stable time basis

missing

**pull, as-you-type
async streams**

1:n

explicit textual state

fault handling

stable time basis

side effect safety

what-if overlay

(ocap) security

data viz

transactions

contribution

**pull, as-you-type
async streams**

1:n

**explicit textual state
fault handling
stable time basis**

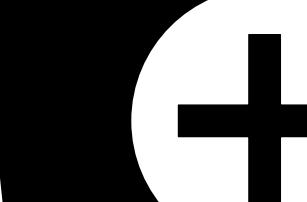
tickets

context

contribution

tickets

context



contribution

(do-something-now)!

bang

!(eval-this-on-every-keystroke)

sigils

probe

(? (+ 1 2) 3)

updates

editor macros

Can Programs be User Interfaces for Databases?
Freeform interactive s-expressions via edit-time macros



Search

Users > albertzak > Git > fun > src > ext > emacros.cljs

(+ 1] 2)

contribution

**pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis**

A screenshot of a Clojure development environment, likely Calva, running in a dark-themed browser window. The top bar includes standard OS X window controls (red, yellow, green) and a search field. The main area shows a file path: Users > albertzak > Git > fun > src > ext > emacros.cljs. The code editor displays the following Clojure code:

```
(+ 1] 2)  ⇒  3
```

The number '1' is highlighted in yellow, and the number '2' is highlighted in pink. The output '3' is shown to the right of the expression. The bottom status bar shows various tool icons and the current context: Ln 69, Col 2, Spaces: 2, UTF-8, LF, Clojure, Go Live, pprint, and a lambda symbol.

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

Calva

A screenshot of a Clojure code editor interface. At the top, there are window control buttons (red, yellow, green) and a search bar labeled "Search". To the right of the search bar are icons for minimizing, maximizing, and closing the window. Below the search bar, the file path "Users > albertzak > Git > fun > src > ext > emacros.cljs" is displayed. The main editor area shows the following code:

```
(+ 1] 2)  
3
```

The number "1]" is highlighted with a pink selection bar. In the bottom right corner of the editor area, there is a white box with the word "Calva" in bold black font.

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

A screenshot of a dark-themed code editor window. The title bar shows the file name "emacros.cljs". The status bar at the bottom indicates the cursor is at "Ln 69, Col 2" with "Spaces: 2" and "UTF-8" encoding, and the language is "Clojure". The main editor area contains the following Clojure code:

```
(? (+ 1] 2) 3)
```

contribution

**pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis**

A screenshot of a dark-themed code editor window. The title bar shows the file name "emacros.cljs". The status bar at the bottom indicates the current position is "Ln 69, Col 2". The main editor area contains the following Clojure code:

```
(? (+ 2[ 2) 4)
```

contribution

**pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis**

contribution

pull, as-you-type async streams

1:n

explicit state fault handling stable time basis



contribution

(read co2-sensor-livingroom)

Calva

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

Search

Users > albertzak > Git > fun > src > ext > emacros.cljs

(read co2-sensor-livingroom) → Promise< ~ >

Calva

REPL 🔍 clojure-lsp

Ln 69, Col 2 Spaces: 2 UTF-8 LF Clojure Go Live pprint [λ] ⚡

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

A screenshot of a Clojure code editor interface. The title bar shows the file name "emacros.cljs". The status bar at the bottom indicates the current line (Ln 69), column (Col 2), and character count (Spaces: 2). The code in the editor is:

```
(→  
  (read co2-sensor-livingroom)  
  (then (fn [v] (println v))))
```

The word "read" is highlighted in orange. A cursor is positioned at the end of the second line, after the closing parenthesis of the first function. A large white box with the word "Calva" in black text is overlaid on the bottom right of the editor window.

contribution
pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

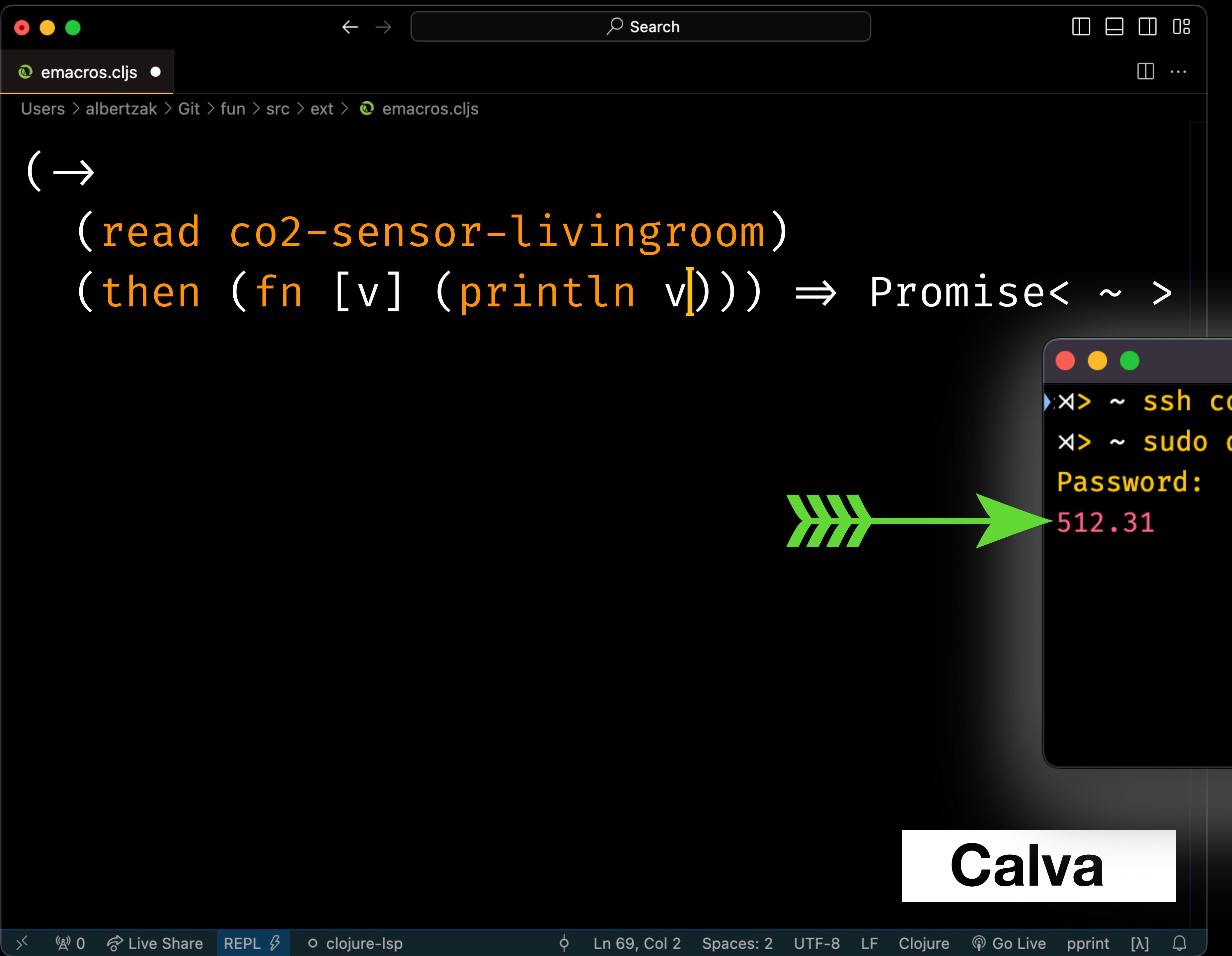
A screenshot of a Clojure code editor interface. The title bar shows the file name "emacros.cljs". The status bar at the bottom indicates the current line and column (Ln 69, Col 2), character encoding (UTF-8), and file type (Clojure). A search bar is located at the top center. The main code area displays the following Clojure code:

```
(→  
(read co2-sensor-livingroom)  
(then (fn [v] (println v))) => Promise< ~ >
```

The word "read" is highlighted in orange, indicating it is a macro or special form. The code uses the Calva Clojure editor, as evidenced by the "Calva" watermark in the bottom right corner.

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis



contribution

pull, as-you-type async streams

1:n

Explicit state

fish /Users/albertzak

۷۸

fish /Users/albertzak

۷۸

```
✖> ~ ssh co2-sensor-node
✖> ~ sudo docker logs -f fdcdab7ca6a7
Password:
→ 512.31
```

Calva

A screenshot of a macOS application window titled "emacros.cljs". The window contains a terminal-style interface with the following content:

```
(read co2-sensor-livingroom)
```

The window has standard OS X window controls (red, yellow, green buttons) at the top left, a search bar with placeholder "Search" at the top center, and a menu bar with icons for window control and time at the top right.

The title bar shows the file path: "Users > albertzak > Git > fun > src > ext > emacros.cljs".

The bottom of the window shows standard macOS system status icons: battery level, signal strength, volume, and a bell icon.

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

A screenshot of a Clojure REPL window. The title bar shows 'emacros.cljs'. The file path is 'Users > albertzak > Git > fun > src > ext > emacros.cljs'. The code in the buffer is:

```
(read co2-sensor-livingroom)
(?) (:value (ticket "e2d9j9d30")) :?:/pending)
```

The code uses the `read` macro to read from a CO2 sensor in the living room. It then uses the `ticket` macro to handle the value, specifying a ticket ID of "e2d9j9d30". The `:?:/pending` part indicates that the ticket is pending.

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

emacros.cljs

Users > albertzak > Git > fun > src > ext > emacros.cljs

```
(read co2-sensor-livingroom)
(?) (:value (ticket "e2d9j9d30")) 530.13)
```

A screenshot of a code editor window titled "emacros.cljs". The file path is "Users > albertzak > Git > fun > src > ext > emacros.cljs". The code shown is:

```
(read co2-sensor-livingroom)
(?) (:value (ticket "q0pznicd2")) :?/pending
(?) (:value (ticket "e2d9j9d30")) 530.13
```

The code uses the `read` macro from `co2-sensor-livingroom` and two ticket-based asynchronous streams. The first stream uses the `:?:pending` pattern, and the second stream returns a value of 530.13.

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

tickets

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

tickets

A screenshot of a Clojure REPL window titled "emacros.cljs". The window shows the following code:

```
(read co2-sensor-livingroom)
(?) (:value (ticket "q0pznicd2")) 538.90
(?) (:value (ticket "e2d9j9d30")) 530.13
```

The code uses the `read` macro to read from a `co2-sensor-livingroom` source. It then uses two `(?)` forms to pull values from a ticket-based stream. The first ticket is "q0pznicd2" and its value is 538.90. The second ticket is "e2d9j9d30" and its value is 530.13.

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

tickets

```
(read co2-sensor-livingroom)
(?) (:value (ticket "q0pznicd2")) 538.90
(?) (:value (ticket "e2d9j9d30")) 530.13

(?) (ticket "e2d9j9d30")
{:value 530.13
 :requested-at #inst "2023-11-03 12:13:01"
 :requested-by "3440d4j901"
 :resolved-at #inst "2023-11-03 12:13:04"
 :resolved-by "cijoeqqd20f"})
```

contribution

pull, as-you-type
async streams
1:n
explicit state
fault handling
stable time basis

tickets

```
(read co2-sensor-livingroom)
(?) (:value (ticket "q0pznicd2")) 538.90
(?) (:value (ticket "e2d9j9d30")) 530.13

(?) (ticket "e2d9j9d30")
{:value 530.13
 :requested-at #inst "2023-11-03 12:13:01"
 :requested-by "3440d4j901"
 :resolved-at #inst "2023-11-03 12:13:04"
 :resolved-by "cijoeqqd20f"})
```

```
(read co2-sensor-livingroom)
```

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

contribution

(read co2-sensor)

tickets

context

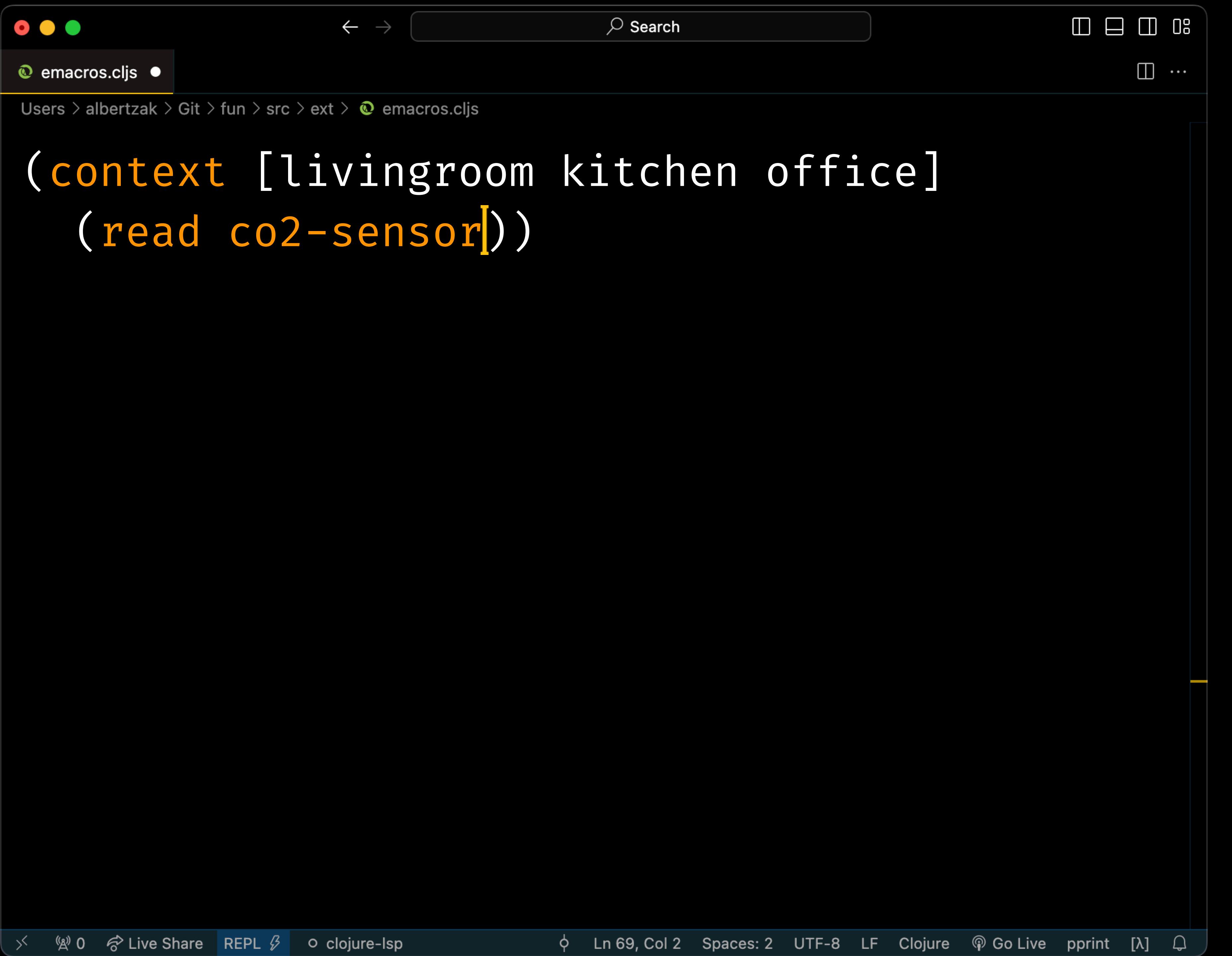
pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context



contribution

pull, as-you-type async streams

1:n

explicit state fault handling stable time basis

tickets

context

contribution

pull, as-you-type
async streams

1:n
explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (read co2-sensor))
(?) (:value (ticket "s3de0d"))
[ :?:/pending :?:/pending :?:/pending])
```

contribution

pull, as-you-type
async streams

1:n
explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (read co2-sensor))
(?) (:value (ticket "s3de0d"))
[ :?:/pending :?:/pending 404.32])
```

contribution

pull, as-you-type
async streams

1:n
explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (read co2-sensor))
(?
  (:value (ticket "s3de0d"))
  [569.21 :?:pending 404.32])
```

contribution

pull, as-you-type
async streams

1:n
explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (read co2-sensor))
(? (:value (ticket "s3de0d"))
  [569.21 801.91 404.32])
```

A screenshot of a code editor window titled "emacros.cljs". The file path is "Users > albertzak > Git > fun > src > ext > emacros.cljs". The code itself is:

```
(context [livingroom kitchen office]
  (read co2-sensor))
(→
  (? (:value (ticket "s3de0d")))
  [569.21 801.91 404.32])
(? average] 591.81)
```

The code uses Clojure's context and ticket mechanisms to handle multiple sensor readings.

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (→ (read co2-sensor)
      (? average] 591.89)))
(→
  (? (:value (ticket "s3de0d"))
    [569.21 801.91 404.32])
  (? average 591.81))
```

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (→ (read co2-sensor)
      (? average] 593.14)))
(→
  (? (:value (ticket "s3de0d"))
    [569.21 801.91 404.32])
  (? average 591.81))
```

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (→ (read co2-sensor)
      (? average] 600.12)))
(→
  (? (:value (ticket "s3de0d"))
    [569.21 801.91 404.32])
  (? average 591.81))
```

contribution

pull, as-you-type
async streams

1:n

explicit state
fault handling
stable time basis

tickets

context

```
(context [livingroom kitchen office]
  (→ (read co2-sensor)
      (? average] 586.74)))
(→
  (? (:value (ticket "s3de0d"))
    [569.21 801.91 404.32])
  (? average 591.81))
```

future work

sigil-probe/eval arbitrary subexpressions

"under construction" sigil (+theming)

security: object capability emacros

nondestructive preview overlays: history, what-if



(thank-you)!

read

On REPLs

print

eval

