Reading Inside Out An algorithm to indent S-expressions by reading them inside out.

2024-05-06 | Alex Vear <<u>alex@vear.uk</u>> | ELS 2024 Lightning talk

(defn distance $(\p\ where})$ and $(\q\)$ of any dimension." (math/sqrt (reduce +

p q)))

"Calculate the Euclidean distance between two points

(map (fn [p' q'] (math/pow (- p' q') 2))



(defn distance $(\p\ where})$ and $(\q\)$ of any dimension." (math/sqrt (reduce + (map (fn [p' q'] p q))))

"Calculate the Euclidean distance between two points

(math/pow (- p' q') 2))



How can we do this efficiently?

Build a special-purpose Clojure reader inside of Vim.

Constraints

- Vim script (the implementation language) is painfully slow.
 - Loop iterations especially so!
 - Vim 9 script and Lua?

- The algorithm is invoked separately on each line to be indented. State cannot be shared between invocations.

• Cannot assume that all code in the file is valid.

∧-->

(read-string "(defn foo [x]\n (* x\n 42))")

The base algorithm

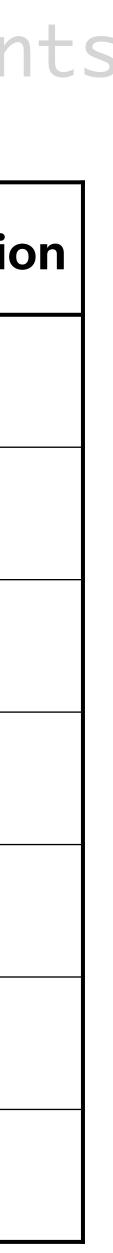
1	(defn distance
2	"Calculate the Euclide
3	$(\p\) of a$
4	[p q]
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q']
9	(math/pow (- p' q') 2))
0	p q))))



1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q']
9	(math/pow (- p' q') 2))
0	p q))))

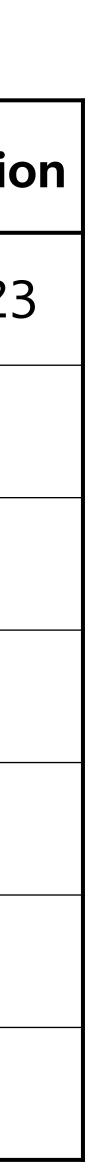
1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q ¹
9	(math/pow (- p' q') 2))
10	p q))))

Locati



1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q [*]
9	(math/pow (- p' q') 2))
10	p q))))

Token	Locati
]	8,2



1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q [*]
9	(math/pow (- p' q') 2))
10	p q))))

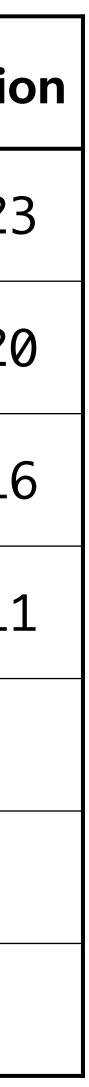
Token	Locati
	8,2



(defn distance
"Calculate the Euclide
$(\p\) and \q) of c$
(math/sqrt
(reduce +
(map (fn [p'
q ¹
(math/pow (- p' q') 2))
p q))))

Token	Locati
	8,2
	7,2
(7, 1
(7, 1





1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q [*]
9	(math/pow (- p' q') 2))
10	p q))))

Token	Locati
	8,2
Ľ	7,2
(7, 1
(7, 1

1	t	S
		1
0	n	

3	
0	
6	
1	

1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q [*]
9	(math/pow (- p' q') 2))
10	p q))))

Token	Locati
(7, 1
(7, 1



1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q '
9	(math/pow (- p' q') 2))
10	p q))))

Token (Locatio 7, 16 7, 11

on	
6	
1	

1 (defn distance "Calculate the Euclidean distance between two points $(\p\ und\ uq)$ of any dimension." 3 (math/sqrt 5 (reduce + 6 (map (fn [p' 7 8 a ' (math/pow (- p' q') 2)) 9 10 p a))))

Token	Locati
(7, 1
(7, 1

on	
6	
1	

1	(defn distance
2	"Calculate the Euclide
3	$(\p\) and \q\) of c$
4	
5	(math/sqrt
6	(reduce +
7	(map (fn [p'
8	q '
9	(math/
10	p q))))

ean distance between two points any dimension."

/pow (- p' q') 2))

Token	Locati
(7, 1
(7, 1

ion	
L6	
1	
	-
	_
	-
	_



Escaped characters

- While scanning for tokens, if any are preceded by an odd number of escape characters, we ignore those tokens.
 - Backslash is the escape character in Clojure.
 - It was either a Clojure character literal, or an escape code in a string.

Semicolon comments

• Semicolon comments are unlike the rest of the syntax as they are not S-expressions, instead throwing away everything until the next newline.

1 (defn distance "Calculate the Euclidean distance between two points 3 (\"p\" and \"q\") of any dimension." (math/sqrt 5 (reduce + 6 (map (fn [p' 8 a'] 10 (math/pow (- p' q') 2)) 11 p a))))

Token	Locatio





1 (defn distance "Calculate the Euclidean distance between two points $(\p\ und\ \q\) of any dimension."$ 3 [p q] (math/sqrt 5 (reduce + (map (fn [p' 9 10 (math/pow (- p' q') 2)) 11 p a))))

Token	Locatio
]	9, 23





1 (defn distance "Calculate the Euclidean distance between two points 3 (\"p\" and \"q\") of any dimension." 5 (math/sqrt (reduce + 6 (map (fn [p' ;; (8 **a'** 9 10 (math/pow (- p' q') 2)) 11 p a))))

Token	Location
]	9, 23
Ľ	8,26
(8, 24
• •	8, 22
• •	8, 21





1 (defn distance "Calculate the Euclidean distance between two points 3 (\"p\" and \"q\") of any dimension." p q 5 (math/sqrt (reduce + 6 (map (fn [p' 8 ;; (**a'** 9 10 (math/pow (- p' q') 2)) 11 p q)))

	Token	Location
		9, 23
		8,26
	(8, 24
	• •	8, 22
	• •	8, 21





1 (defn distance "Calculate the Euclidean distance between two points 3 (\"p\" and \"q\") of any dimension." [p q] (math/sqrt 5 (reduce + 6 (map (fn [p' 8 9 **a'** 10 (math/pow (- p' q') 2)) 11 p a))))

Token	Locatio
]	9, 23





Multiline strings

- Huge pain to deal with.
 - same.

- - additional checks is minimal.

Made worse due to opening and closing string delimiters being the

• If we detect a lone string delimiter on a line, we activate the "maybe in a multiline string" mode and apply special rules to figure it out for sure.

• Fairly uncommon (except as docstrings) so the performance hit of the



Performance improvements

Code version	200 lines of Clojure	600 lines of EDN
Original	0.96s	4.6s (let clojure_maxlines = 0)
New	0.33s	2.6s
New (Vim 9)	0.06s	0.46s

Other benefits

- Much simpler and more configurable with no hacks.
 - >400 LOC → ~300 LOC.

- Not reliant on syntax highlighting.
 - Works with Treesitter.

Reusable for other Lisps with Clojure-like syntax.

Find out more

- <u>https://github.com/clojure-vim/clojure.vim/pull/31</u>

- Email: <u>alex@vear.uk</u>
- WWW: <u>https://www.alexvear.com</u>

• Will be merged soon(ish), just a few more small changes to make.