Introduction to Functional Programming with ClojureScript

David Eisenberg

Terminology

Clojure

A dialect of Lisp that runs on the JVM

Terminology

- Clojure
  - A dialect of Lisp that runs on the JVM
- ClojureScript
  - A dialect of Clojure targeting JavaScript
  - The compiler

http://leiningen.org

Leiningen

for automating Clojure projects without setting your hair on fire

1. Download the `lein` script (or on Windows `lein.bat`)
2. Place it on your `$PATH` where your shell can find it (eg. `~/bin`)
3. Set it to be executable (`chmod a+x ~/bin/lein`)
4. Run it (`lein`) and it will download the self-install package

You can check your package manager as well, but be sure you get version 2.x. There's also an installer for Windows users.

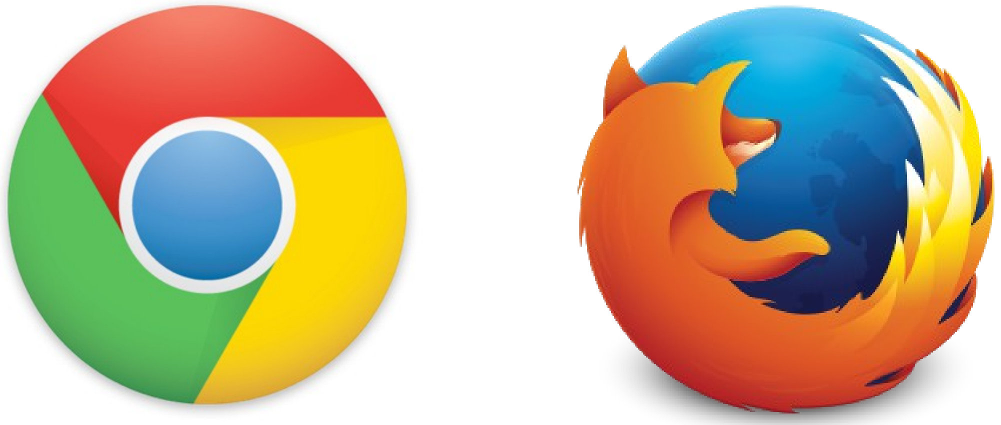
Vim

Emacs

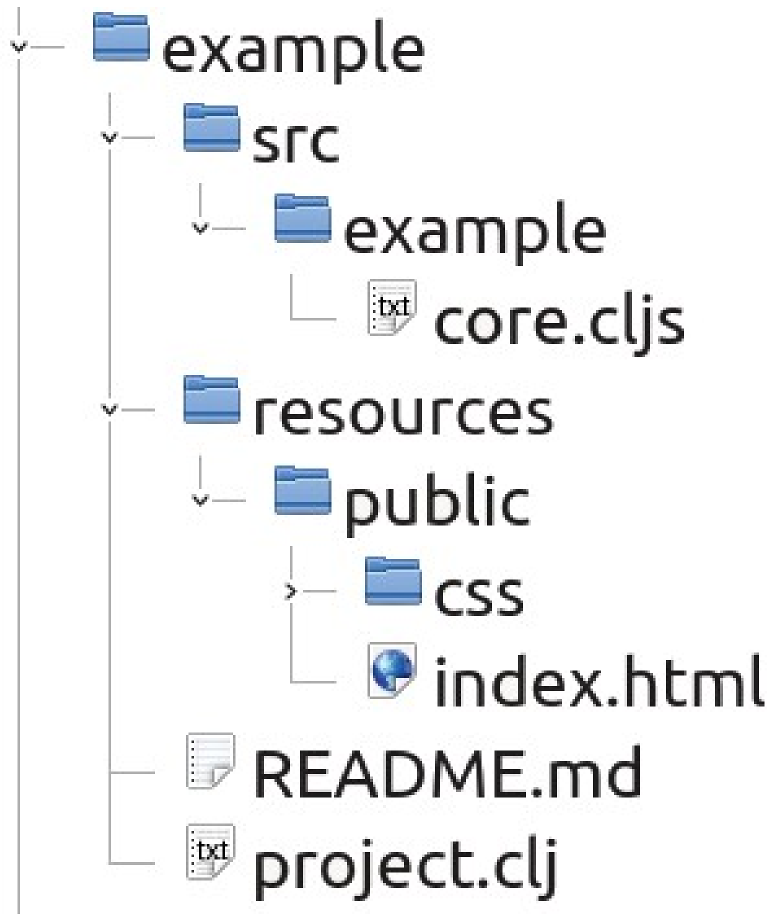
IntelliJ IDEA

jE

Notepad++



```
lein new figwheel example

cd example
```



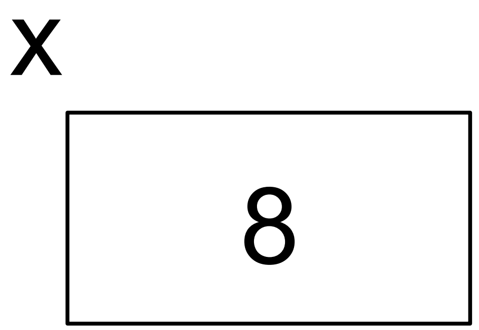
```
example
  src
    example
      core.cljs
  resources
    public
      css
      index.html
  README.md
  project.clj
```

- Read
- Evaluate
- Print
- Loop

```
lein figwheel

rlwrap lein figwheel
```

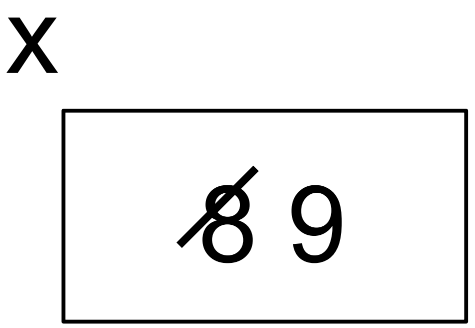
Variables

```
x = 8;
```



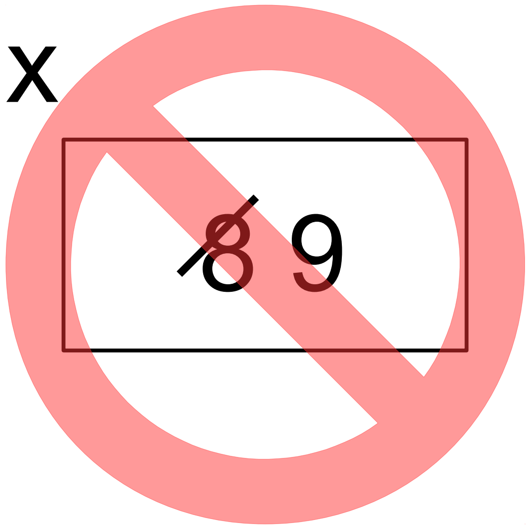
Variables

```
x = 8;
x = 9;
```



Variables

```
x = 8;
x = 9;
```



Variables

$$x = 8$$

Variables

$$x = 8$$

$$x = 9$$

Variables

$$x = 8$$

$$x = 9$$

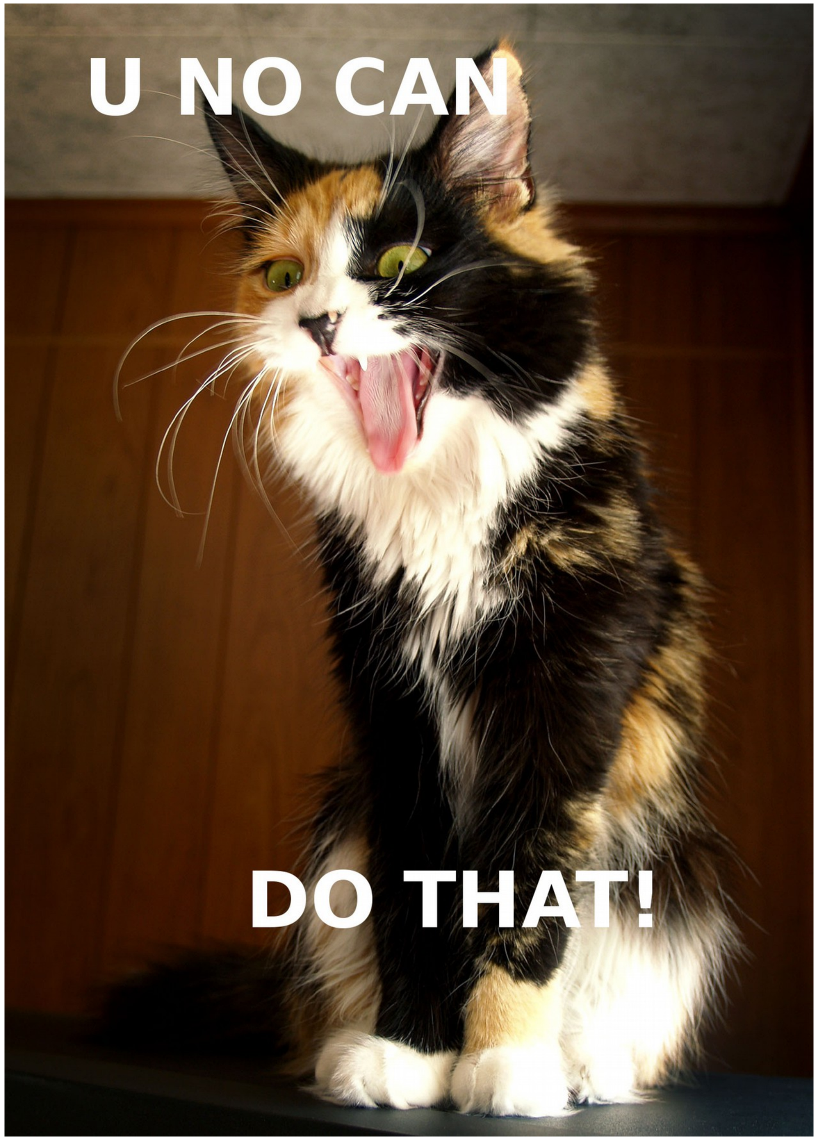
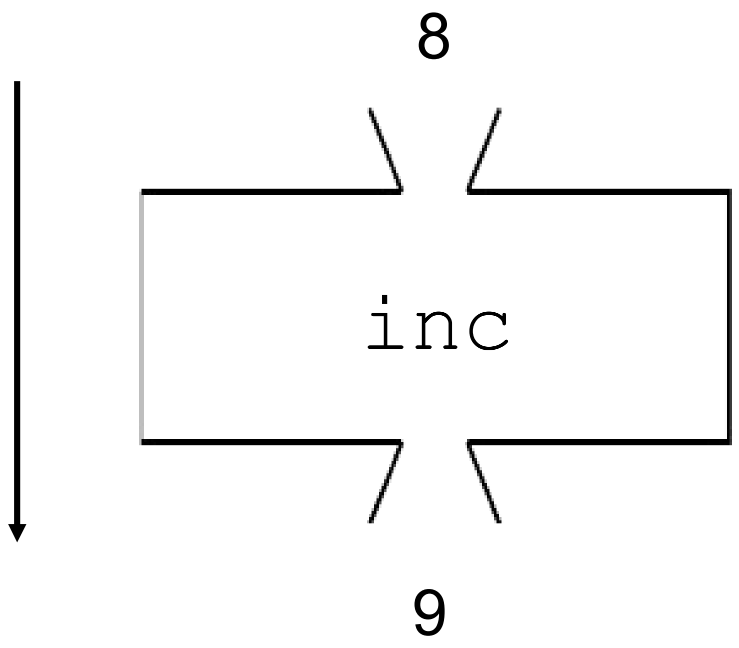


Image by xvoltagex from freeimages.com

Variables

```
(def x 8)
```

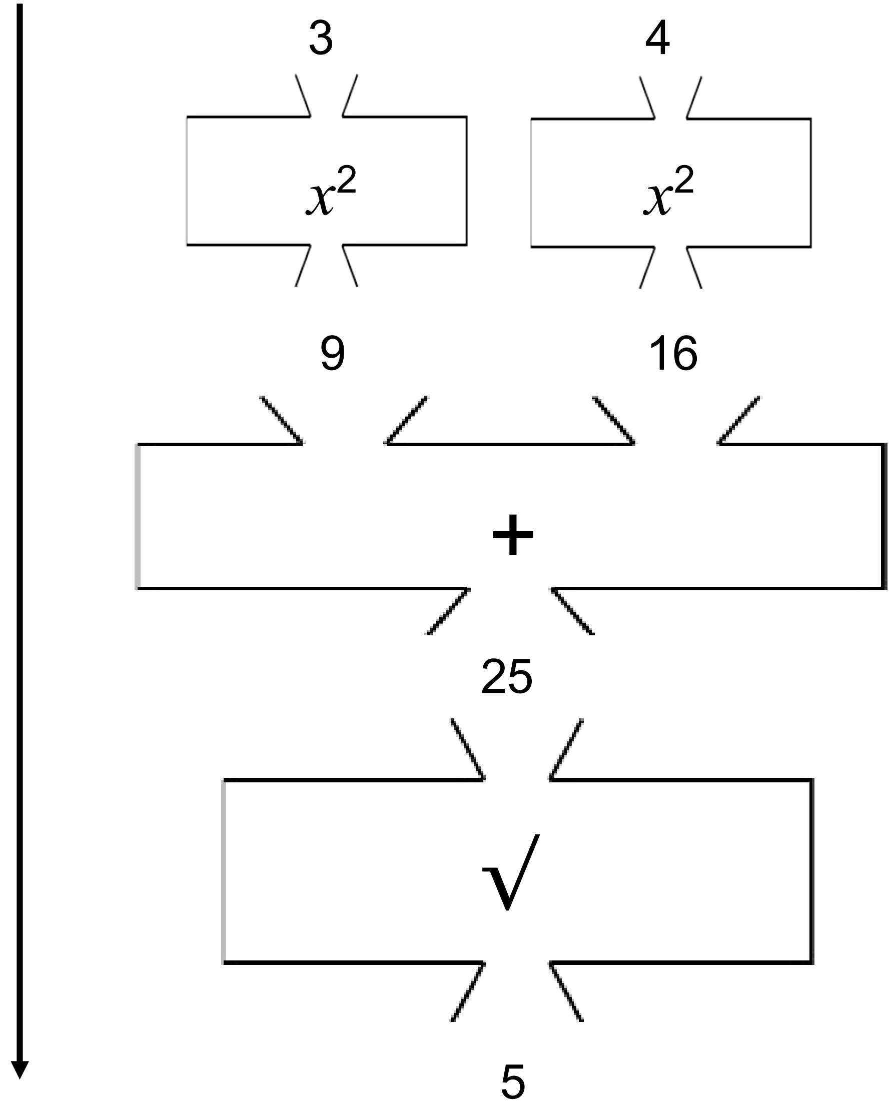
Functions



Functional Programming

Functions transform data.

Functions



Functions

```
(def name (fn [params] body))
```

Functions

```
(def name (fn [params] body))

(def square (fn [x] (* x x)))
```

Functions

```
(defn name [params] body)

(defn cube [a] (* a a a))
```

Accessing JavaScript

(.*method object args*)

```
(.sqrt js/Math 2)
```

Accessing JavaScript

Full version	Shortcut
`(.sqrt js/Math 2)`	`(js/Math.sqrt 2)`

`(.parseFloat js/window "12.34")`	`(js/window.parseFloat "12.34")`
	`(js/parseFloat "12.34")`

Accessing JavaScript

```
(.-property object)

(.-PI js/Math)
```

Accessing JavaScript

```
(class. args)

(js/Date.)
(js/Date. 0)
```

Functions

```
(get-value! id-string)

(to-number string)
```

Functions

```
(defn get-value! [id-string]
  (.-value
    (js/document.getElementById id-string)))
```

Conditionals with *if*

```
(if condition true-expr
              false-expr)

(if (< x 0) (- x) x)
```

Functions

```
(defn to-number [str]
  (if (js/isNaN (js/parseFloat str))
    0
    (js/parseFloat str)))
```

Let’s use *let*

```
(let [symbol1 value1
      symbol2 value2]
  (expr)...
  (expr))
```

Functions

```
(defn to-number [str]
  (let [n (js/parseFloat str)]
    (if (js/isNaN n) 0 n)))
```

def vs. let

```
(def x 8)
```

–The symbol *x* is available to all functions in the namespace.

```
(let [y 42]
  (expr))
```

–The symbol *y* is available only to expressions within the `(let...)`

Collections

List

```
'(1 2 3)
(list 1 2 3)
```

Collections

List

```
'(1 2 3)
(list 1 2 3)
```

Vector

```
["a" "b" "c"]
(vector "a" "b" "c")
```

Collections

List

```
'(1 2 3)
(list 1 2 3)
```

Vector

```
["a" "b" "c"]
(vector "a" "b" "c")
```

Map

```
{"quantity" 29
 "price" 3.75}
```

Collections

List

```
'(1 2 3)
(list 1 2 3)
```

Vector

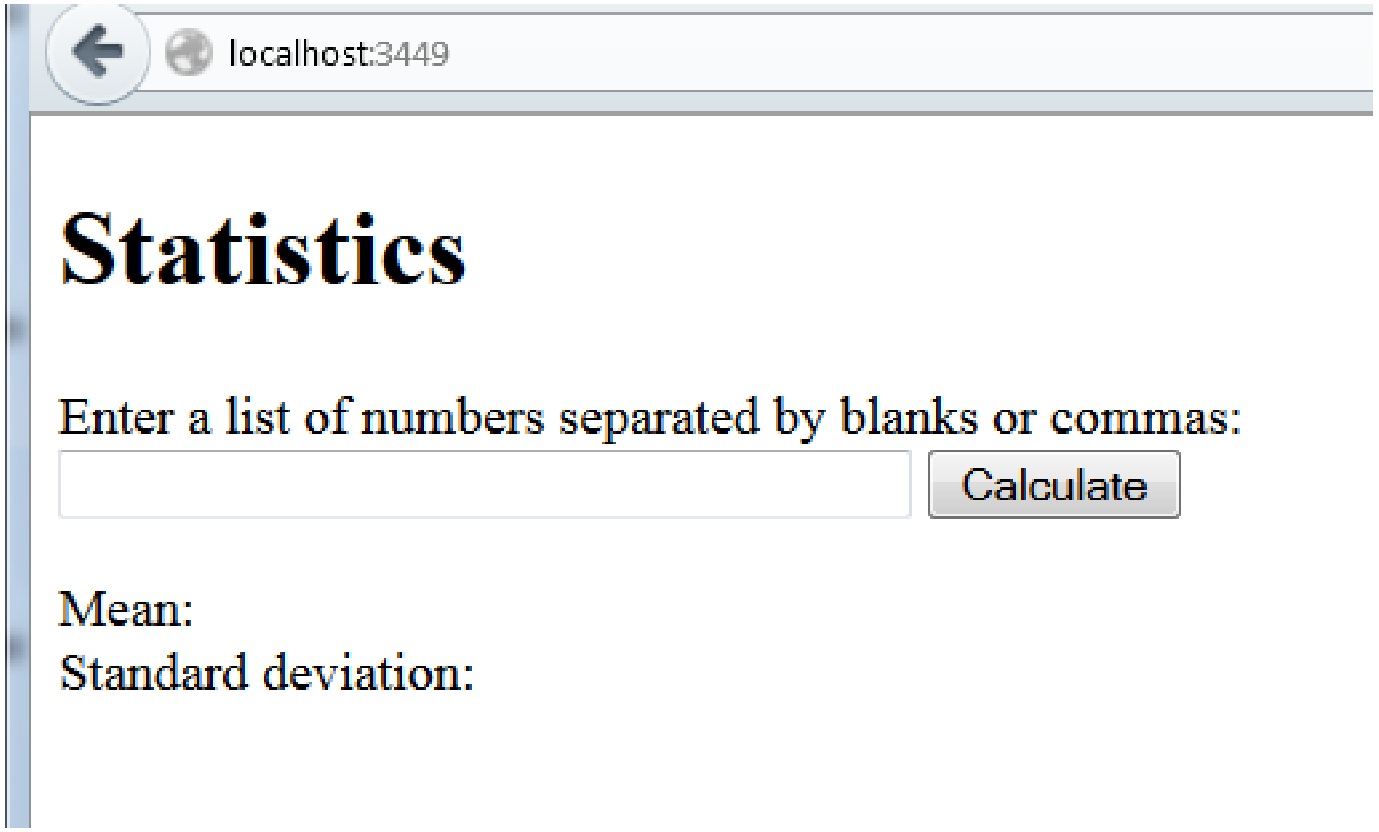
```
["a" "b" "c"]
(vector "a" "b" "c")
```

Map

```
{"quantity" 29
 "price" 3.75}
```

Set

```
#{3 5 1 7 2}
```


localhost:3449

Statistics

Enter a list of numbers separated by blanks or commas:

Mean:
Standard deviation:

To acquire data:

$$\begin{tikzcd}
\text{Take comma-separated input string} \arrow[d] & \text{"1, 3, x, 7, 5"} \arrow[d] \\
\text{Split into individual items} \arrow[d] & \text{["1" "3" "x" "7" "5"]} \arrow[d] \\
\text{Convert to numbers} \arrow[d] & \text{[1 3 NaN 7 5]} \arrow[d] \\
\text{Eliminate non-numerics} & \text{[1 3 7 5]}
\end{tikzcd}$$

Mean	$\frac{\sum x}{n}$
Standard Deviation	$\sqrt{\frac{\sum x^2 - \frac{\left(\sum x\right)^2}{n}}{n-1}}$

To calculate mean and standard deviation:

Get number of items in list	[1 3 7 5] $\rightarrow$ 4
Reduce list to a sum	[1 3 7 5] $\rightarrow$ 16
Reduce list to a sum of squares	[1 3 7 5] $\rightarrow$ 84

To acquire data:

Take comma-separated input string	"1, 3, x, 7, 5"
↓	↓
Split into individual items	["1" "3" "x" "7" "5"]
↓	↓
Convert to numbers	[1 3 NaN 7 5]
↓	↓
Eliminate non-numerics	[1 3 7 5]

```
(map f [x₀ x₁ x₂ ... xₙ])
```

$$\begin{array}{ccccccc} [& x_0 & x_1 & x_2 & \ldots & x_n &] \\ & \downarrow & \downarrow & \downarrow & & \downarrow & \\ [& f(x_0) & f(x_1) & f(x_2) & \ldots & f(x_n) &] \end{array}$$

This is a function that
takes one argument

$$\texttt{(map f [}\mathtt{x}_0\ \mathtt{x}_1\ \mathtt{x}_2\ \texttt{. . .}\ \mathtt{x}_n\texttt{])}$$

$$\begin{array}{cccccccc}
[& x_0 & x_1 & x_2 & \dots & x_n &] \\
 & \downarrow & \downarrow & \downarrow & & \downarrow & \\
[& f(x_0) & f(x_1) & f(x_2) & \dots & f(x_n) &]
\end{array}$$

```
(defn square [x]
  (* x x))

(map square [12  4  1.5  9])
```

$$\begin{tikzcd}
\text{[} & 12 \arrow[d] & 4 \arrow[d] & 1.5 \arrow[d] & 9 \arrow[d] & \text{]} \\
 & \text{(square 12)} \arrow[d] & \text{(square 4)} \arrow[d] & \text{(square 1.5)} \arrow[d] & \text{(square 9)} \arrow[d] & \\
\text{(} & 144 & 16 & 2.25 & 81 & \text{)}
\end{tikzcd}$$

This is a function that takes one argument and returns *true* or *false*.

(filter f [x_0 x_1 x_2 . . . x_n])

```
(filter even? [3 16 22 7 4])
```

```
[     3            16            22            7            4        ]
(even? 3)    (even? 16)    (even? 22)    (even? 7)    (even? 4)
  false          true          true          false        true
                  ↓             ↓                          ↓
(                16            22                          4         )
```

Mean $$\frac{\sum x}{n}$$

Standard Deviation $$\sqrt{\frac{\sum x^2 - \frac{\left(\sum x\right)^2}{n}}{n-1}}$$

To calculate mean and standard deviation:

Get number of items in list	[1 3 7 5] → 4
Reduce list to a sum	[1 3 7 5] → 16
Reduce list to a sum of squares	[1 3 7 5] → 84

```
(count numbers)
```

```
(reduce sum 0 data)

         (def data [  5       4        6   ])
```

```
(reduce sum 0 data)

        (def data [  5       4         6    ])
```

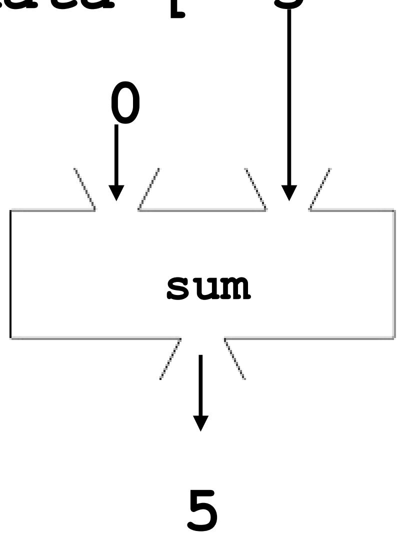
```
(reduce sum 0 data)

        (def data [  5       4         6    ])
```

```
(reduce sum 0 data)

          (def data [  5         4           6    ])
```

```
(defn sum [acc x] (+ acc x))
(reduce sum 0 data)

          (def data [  5      4       6   ])
```

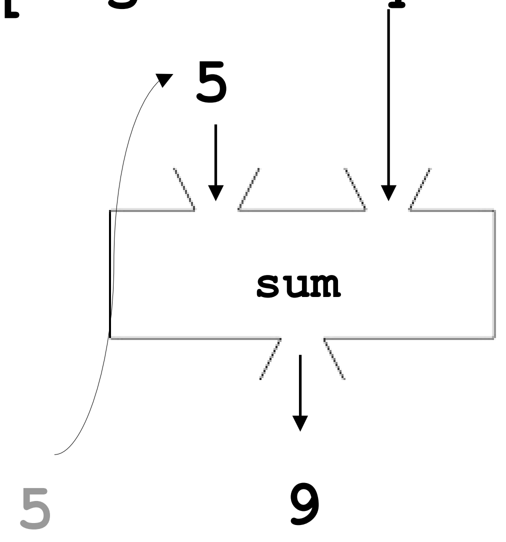
```
(defn sum [acc x] (+ acc x))
(reduce sum 0 data)
```

```
(def data [  5       4          6   ])
```



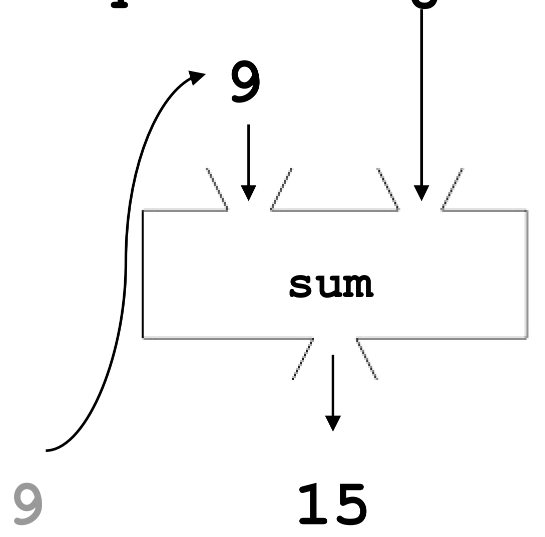
```
(defn sum [acc x] (+ acc x))
(reduce sum 0 data)
```

```
(def data [  5      4        6   ])
```



```
(defn sum [acc x] (+ acc x))
(reduce sum 0 data)
```

```
(def data [  5      4        6   ])
```



```
(defn sum [acc x] (+ acc x))
(reduce sum 0 data)
```

```
          (def data [  5      4        6   ])
```

$\boxed{15}$

```
(reduce + 0 data)

        (def data [  5        4          6   ])
```

$\boxed{15}$

Where to go from here

You can think functionally in JavaScript.

You can use `map`, `reduce`, and `filter` in JavaScript.

Where to go from here

DOM manipulation

– Google “closure” library
– dommy
– Domina
– Enfocus
– Reagent (if you are using React)

Where to go from here

Documentation

– https://github.com/clojure/clojurescript/wiki
– http://funcool.github.io/clojurescript-unraveled/

Somewhat more advanced topics
– https://github.com/cljsinfo/cljs-api-docs/tree/catalog
– http://clojurescriptmadeeasy.com/

Further exercises
– https://github.com/jdeisenberg/etudes-for-clojurescript