# Learning by Doing - Beginning Clojure by Solving Puzzles

•••

DJ Adams
Principal, Bluefin Solutions

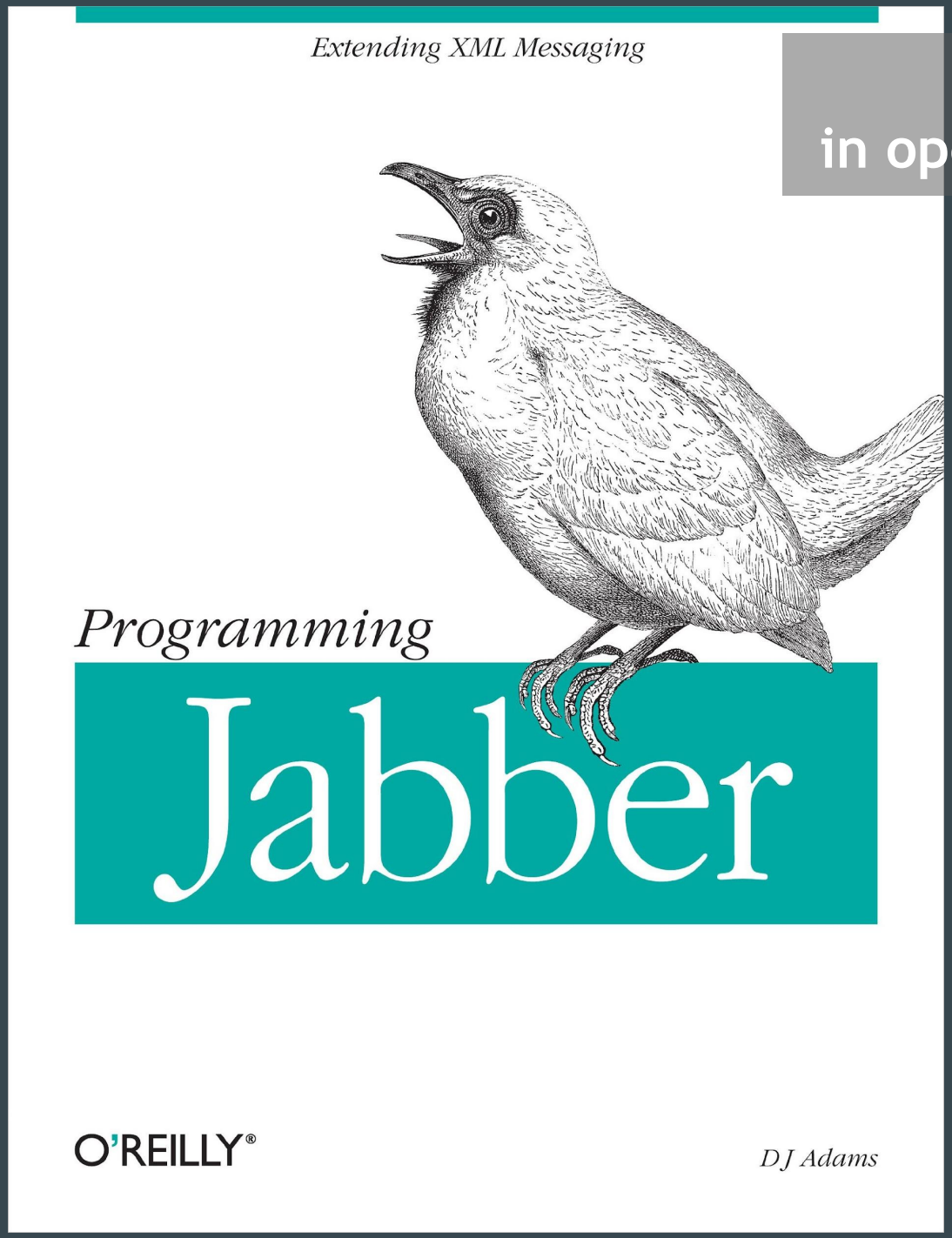
Manchester Lambda Lounge
MadLab
March 2016


# Basic Plus on Systime PDP-11

# 6502 assembler & Atom Basic on Acorn Atom


370 assembler, COBOL, Rexx, CLIST, JCL
on IBM System 370
FIRE
POINT
IBM
370

# Perl & Python
# in open source communities



Sales document | Edit | Goto | Extras | Environment | System | Help

# Display Internet Quotation 201203615: Overview

Orders

| | | | | | |
|---|---|---|---|---|---|
| Internet Quotation | 201203615 | | Net value | 1.566,25 | USD |
| Sold-To Party | 178890 | Cleveland Brothers Equipment Co. / P.O. Box 2535 / HARRISB… | | | |
| Ship-To Party | 20015679 | CLEVELAND BROTHERS / 5210 Paxton Street / Harrisburg PA … | | | |
| PO Number | ISA-08/26/2013 | | PO date | 26.08.2013 | |

Sales | Item overview | Item detail | Ordering party | Procurement | Shipping | Reason for rejection

| | | | | | |
|---|---|---|---|---|---|
| Req. deliv.date | D 26.08.2013 | | Deliver.Plant | | |
| Valid from | 24.02.2012 | | Valid to | 10.10.2013 | |
| Complete dlv. | | | Total Weight | 27,380 KG | |
| | | | Pricing date | 24.02.2012 | |
| Total amount | 1.566,25 | | Doc. Currency | USD / 1,00000 | |
| Payment terms | Y15D Due net next 10th… | | Incoterms | FCA Shipping Point | |
| Order reason | | | | | |
| Sales area | US00 / 10 / 10 USA, Metso Minerals, Metso Minerals | | | | |

All items

| Item | Material | Order Quantity | Un | S | Description | Customer Material Numb | ItCa | DGIP | HL Itm | D | First date | Plnt | Batch |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 00-611-481-391 | 18 | EA | | BOLT SQUARE HEAD 0.875"-9U… | | YAGN | | 0 | D | 26.08.2013 | US10 | |
| 20 | 07-247-319-003 | 3 | EA | | GASKET 58.75"OD X 52.2"ID X … | | YAGN | | 0 | D | 26.08.2013 | US10 | |

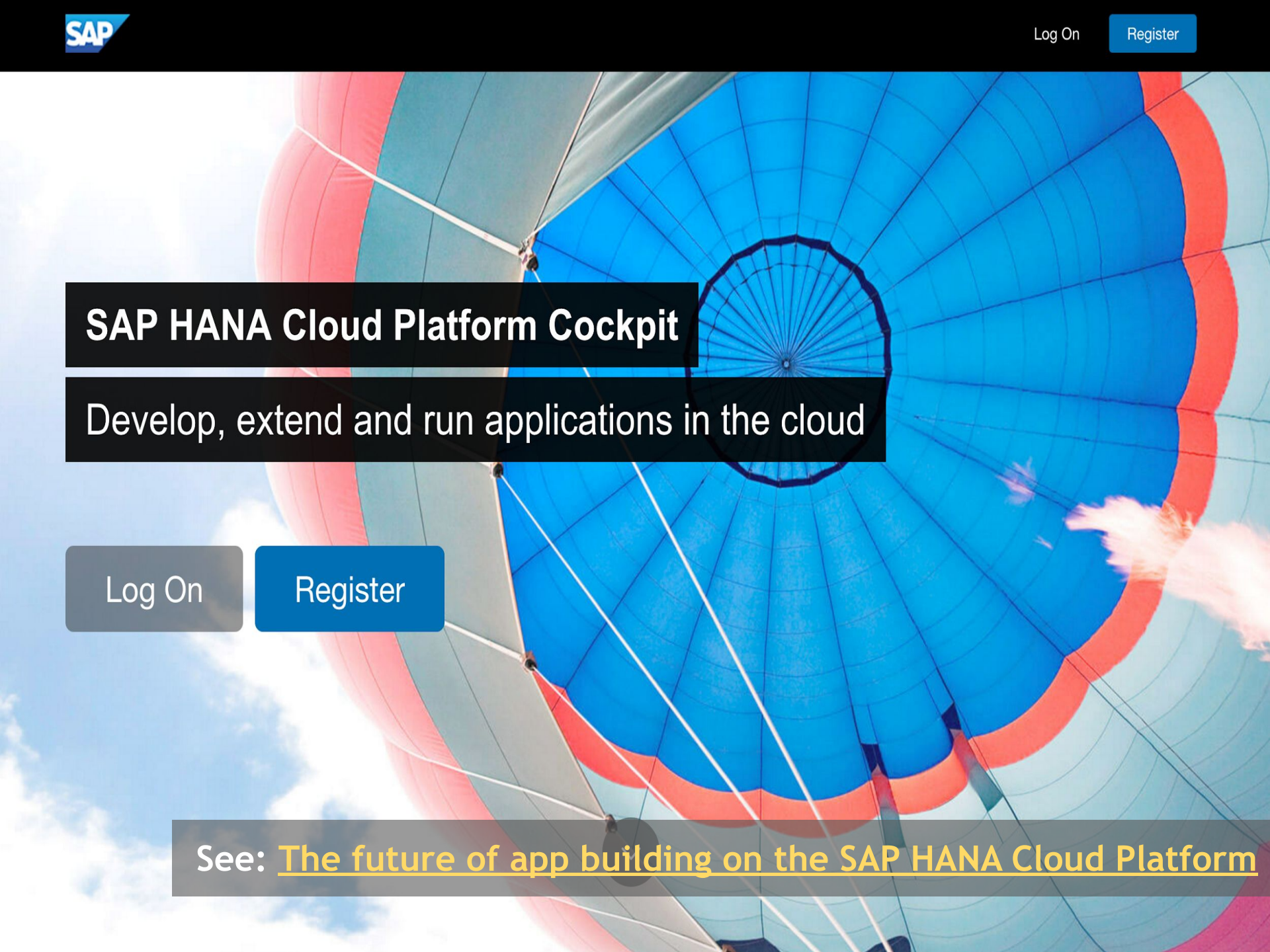
Texts

Shipping Instructions | English

# JavaScript with SAP's UI5 library

map
filter
reduce

# Clojure


SAP
Log On
Register
SAP HANA Cloud Platform Cockpit
Develop, extend and run applications in the cloud
Log On
Register
See: The future of app building on the SAP HANA Cloud Platform

# Toolchain Community

# Native diction

```
17       (map second)                ;; take each found nam
16       (sort)))                    ;; sort them all
15
14 ;; Calculate the value of a character (A=1, B=2 etc)
13 (defn char-val
12   [char]
11   (- (int char)
10      64))
 9
 8 ;; Work out the score for a name: The sum of char values
 7 ;; multiplied by its position in the (1-indexed) list
 6 (defn name-score
 5   [name pos]
 4   (* (inc pos)
 3      (reduce + (map char-val name))))
 2
 1 ;; Solution is the sum of all the name scores
62 (reduce + (map name-score names (range)))
 1
 2
 3 ;; Problem 52
 4
 5 (defn digits
 6   "Return the set of digits in a number."
 7   [n]
 8   (set (str n)))
 9
10 (defn same-digits?
11   "Says whether the digits in each of the numbers
12   in the given seq are the same."
13   [nums]
14   (apply = (map digits nums)))
15
16 (defn multiples
17   "Produces subsequent multiples of a given
NORMAL  <core.clj   clojure   41% :  62:  1   ! trailing[134]
```

```
dj@gargantubrain:~/Google Drive/projects/clojure/scratchpad
$ lein repl
nREPL server started on port 56952 on host 127.0.0.1 - nrepl://12
7.0.0.1:56952
REPL-y 0.3.5, nREPL 0.2.6
Clojure 1.6.0
Java HotSpot(TM) 64-Bit Server VM 1.8.0-b132
    Docs: (doc function-name-here)
          (find-doc "part-of-name-here")
  Source: (source function-name-here)
 Javadoc: (javadoc java-object-or-class-here)
    Exit: Control+D or (exit) or (quit)
 Results: Stored in vars *1, *2, *3, an exception in *e

scratchpad.core=>
```

leiningen, tmux
vim with vim-fireplace etc

```
0 1 1 1:vim* 2:bash#-
```

# Project Euler
# 4Clojure
# Advent of Code

# Project Euler #2 - Even Fibonacci Numbers

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

# loop/recur - still thinking mechanically & procedurally

```
(loop [a 0
       b 1
       n 0]
  (let [c (+ a b)]
    (if (< n 10)
      (do
          (print c)
          (recur b c (inc n))))))

;=> 123581321345589
```

Don't do this!

# Gratuitous recursion - and losing sight of the problem

```
(defn nth-fib
  [n]
  (if (or (= 0 n)
          (= 1 n))
    n
    (+ (nth-fib (- n 1))
       (nth-fib (- n 2)))))

(nth-fib 10)
;=> 55

(map nth-fib (range 10))
;=> (0 1 1 2 3 5 8 13 21 34)
```

# Calmness - simple function & building sequences

```
(defn next-fib-pair
  [[a b]]
  [b (+ a b)])

(next-fib-pair [0 1])
;=> [1 1]

(take 10 (iterate next-fib-pair [0 1]))
;=> ([0 1] [1 1] [1 2] [2 3] [3 5] [5 8] [8 13] [13
21] [21 34] [34 55])

(map first (take 10 (iterate next-fib-pair [0 1])))
;=> (0 1 1 2 3 5 8 13 21 34)
```

# Rearranging - for reuse & understanding

```
(map first (take 10 (iterate next-fib-pair [0 1])))
;=> (0 1 1 2 3 5 8 13 21 34)

(take 10 (map first (iterate next-fib-pair [0 1])))
;=> (0 1 1 2 3 5 8 13 21 34)

(->> (iterate next-fib-pair [0 1])
     (map first)
     (take 10))
;=> (0 1 1 2 3 5 8 13 21 34)
```

# Taking stock - give the sequence a name

```
(def fibs (map first (iterate next-fib-pair [0 1])))

(take 10 fibs)
;=> (0 1 1 2 3 5 8 13 21 34)
```

# Process chain - adding to the pipeline

```
(->> fibs
     (take-while #(< % 4000000)))
;=> (0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
1597 2584 4181 6765 10946 17711 ... 3524578)

(->> fibs
     (take-while #(< % 4000000))
     (filter even?))
;=> (0 2 8 34 144 610 2584 10946 46368 196418 832040
3524578)
```

# Simplicity - a final reduce gets us to the solution

```
(defn next-fib-pair
  [[a b]]
  [b (+ a b)])

(def fibs (map first (iterate next-fib-pair [0 1])))

(->> fibs
     (take-while #(< % 4000000))
     (filter even?)
     (reduce +))
;=> 4613732
```

everything is a list

# Project Euler #22 - Names Scores

Using names.txt (right click and 'Save Link/Target As...'), a 46K text file containing over five-thousand first names, begin by sorting it into alphabetical order. Then working out the alphabetical value for each name, multiply this value by its alphabetical position in the list to obtain a name score.

For example, when the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 938th name in the list. So, COLIN would obtain a score of 938 × 53 = 49714.

What is the total of all the name scores in the file?

# Marshall the data - let the dog see the rabbit

```
"MARY","PATRICIA","LINDA","BARBARA", ... "ALONSO"

(def names
  (->> (slurp "resources/names.txt")
       (re-seq #"\"(\w+)\"")
       (map second)
       (sort)))

(take 3 names)
;=> ("AARON" "ABBEY" "ABBIE")
```

# Helper - calculate the value of a char (A=1, B=2 etc)

```
(seq "ABC")
;=> (\A \B \C)

(int \A)
;=> 65

(defn char-val
  [char]
  (- (int char)
     64))

(map char-val "ABC")
;=> (1 2 3)
```

# Helper - sum of char values times position in list

```
(defn name-score
  [name pos]
  (* (inc pos)
     (reduce + (map char-val name))))

(map char-val "ABE")
;=> (1 2 5)

(name-score "ABE" 1)
;=> 16
```

# Solution - the sum of all the name scores

```
(reduce
  +
  (map name-score
       names
       (range)))
;=> 871198282
```

everything is a list

# 4Clojure #21 - Nth Element

Write a function which returns the Nth element from a sequence.

```
(= (__ '(4 5 6 7) 2) 6)

(= (__ [:a :b :c] 0) :a)

(= (__ [1 2 3 4] 1) 2)

(= (__ '([1 2] [3 4] [5 6]) 2) [5 6])
```

Special restrictions: nth

# Comparing Solutions - recursive head / tail approach

```
(defn my-nth
  [s n]
  (if (= 0 n)
    (first s)
    (my-nth (rest s) (dec n))))

(= (my-nth '(4 5 6 7) 2) 6)
;=> true
```

# Comparing Solutions - minimal list-oriented approach

```
(= (#(last (take (inc %2) %1)) '(4 5 6 7) 2) 6)
;=> true
```

everything is a list

# Triangular Numbers via recursion

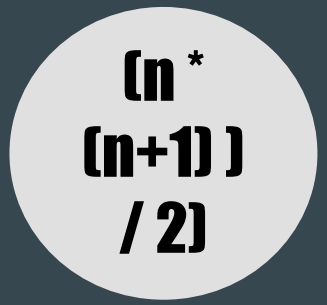
From http://www.braveclojure.com/functional-programming/

```
(defn tri*
  "Generates lazy sequence of triangular numbers"
  ([] (tri* 0 1))
  ([sum n]
     (let [new-sum (+ sum n)]
       (cons new-sum (lazy-seq (tri* new-sum (inc
n)))))))

(def tri (tri*))
(take 10 tri)
;=> (1 3 6 10 15 21 28 36 45 55)
```

# Triangular Numbers via reductions

```
(take 10 (reductions + (range)))
;=> (1 3 6 10 15 21 28 36 45 55)
```



everything is a list

# Advent of Code Day 10 - Elves Look, Elves Say

Today, the Elves are playing a game called look-and-say. They take turns making sequences by reading aloud the previous sequence and using that reading as the next sequence. For example, 211 is read as "one two, two ones", which becomes 1221 (1 2, 2 1s). Look-and-say sequences are generated iteratively, using the previous value as input for the next step. For each step, take the previous value, and replace each run of digits (like 111) with the number of digits (3) followed by the digit itself (1).

For example:
1 becomes 11 (1 copy of digit 1).
11 becomes 21 (2 copies of digit 1).
21 becomes 1211 (one 2 followed by one 1).
1211 becomes 111221 (one 1, one 2, and two 1s).
111221 becomes 312211 (three 1s, two 2s, and one 1).

Starting with the digits in your puzzle input ("31133221"), apply this process 40 times. What is the length of the result?

# Say what you see - first attempt

Influenced by the initial input being a string

```
(defn say
  [saying]
  (let [digits (partition-by identity saying)]
    (->> digits
         (map (fn [d] (vector (count d) (first d))))
         (apply concat)
         (apply str))))

(say "3113322113")
;=> "132123222113"
```

Not ideal!

# Clean input makes for clean processing

```
(defn char-to-digit [c] (- (int c) 48))

(char-to-digit \5)
;=> 5


(defn string-to-digits [s] (map char-to-digit s))

(string-to-digits "3113322113")
;=> (3 1 1 3 3 2 2 1 1 3)
```

# With cleaner input you can work more calmly

```
(def myinput (string-to-digits "3113322113"))

(partition-by identity myinput)
;=> ((3) (1 1) (3 3) (2 2) (1 1) (3))

(->> (partition-by identity myinput)
     (map #(vector (count %) (first %))))
;=> ([1 3] [2 1] [2 3] [2 2] [2 1] [1 3])

(->> (partition-by identity myinput)
     (map #(vector (count %) (first %)))
     flatten)
;=> (1 3 2 1 2 3 2 2 2 1 1 3)
```

# Now we can iterate and get to the solution

```
(defn say
  [digits]
  (->> (partition-by identity digits)
       (map #(vector (count %) (first %)))
       (flatten)))

(count (last (take 41 (iterate say myinput))))
;=> 329356
```

everything is a list

Sequences
Immutability
Pure functions

Simplicity
Solid-state
Calmness