Using Clojure to Solve NPR's Sunday Puzzle

Chicago Clojure Meetup

December 18, 2018

Mitchell Szczepanczyk

http://www.szcz.org

@msszczep

github.com/msszczep

Using Clojure to Solve NP | Sunday Puzzle : NPR

National Public Radio, Inc. [US] | https://www.npr.org/series/4473090/sunday-puzzle

Apps M TC Pol CMA B CF Clj Food Reborg

npr WBEZ91.5 CHICAGO'S npr NEWS STATION

SIGN IN SHOP DONATE

WBEZ Chicago
On Air Now

NEWSCAST LIVE RADIO SHOWS

NEWS ARTS & LIFE MUSIC SHOWS & PODCASTS SEARCH

Sunday Puzzle

THE WEEKLY QUIZ FROM NPR PUZZLEMASTER WILL SHORTZ

SUBSCRIBE TO DAILY NEWS EMAIL

sunday puzzle

https://www.npr.org/search

Puzzle #1

From December 2, 2018:

Think of a common 7-letter word. Drop its second letter, and you'll get a 6-letter word that does not rhyme with the first. Alternatively, you can drop the third letter from the 7-letter word to get a 6-letter word that doesn't rhyme with either of the first two. Further, you can drop both the second and third letters from the 7-letter word to get a 5-letter word that doesn't rhyme with any of the others. What words are these?

Puzzle #1

```
(def words
  (->> "resources/ospd3.txt"
       slurp
       clojure.string/split-lines))

(count words) => 79339
```

Puzzle #1

```
(defn filter-word-count [n]
  (set (filter (comp (partial = n) count) words)))

(def fives (filter-word-count 5))
(def sixes (filter-word-count 6))
(def sevens (filter-word-count 7))

(map count [fives sixes sevens]) => (8551 15055 22821)
```

Puzzle #1

```
(defn remove-letters [seven-letter-word letters-to-remove]
  "letters-to-remove is assumed to be a set of numbers"
  (->> seven-letter-word
       (map-indexed (fn [idx itm] [(inc idx) itm]))
       (remove (comp letters-to-remove first))
       (map second)
       (apply str)))

(remove-letters "abcdefg" #{3}) => "abdefg"
```

Puzzle #1

```
(def solution
  (for [seven sevens
        :when (and (sixes (transform-word seven #{2}))
                   (sixes (transform-word seven #{3}))
                   (fives (transform-word seven #{2 3})))]
    [seven
     (sixes (transform-word seven #{2}))
     (sixes (transform-word seven #{3}))
     (fives (transform-word seven #{2 3}))]))
```

Puzzle #1

```
(["pelages" "plages" "peages" "pages"]
 ["beraked" "braked" "beaked" "baked"]
 ["shnooks" "snooks" "shooks" "sooks"]
 ["thrills" "trills" "thills" "tills"]
 ["shmears" "smears" "shears" "sears"]
 ["phrases" "prases" "phases" "pases"]
 ["creases" "ceases" "crases" "cases"]
 ["splices" "slices" "spices" "sices"]
 ["through" "trough" "though" "tough"]
 ["wairing" "wiring" "waring" "wring"])
```

Puzzle #2

From October 28, 2018:

Think of a famous Broadway musical in two words. Change one letter in it to the preceding letter of the alphabet — so B would become A, C would become B, etc. Remove the space so you have a solid word. The result will name something that all of us are part of. What is it?

Puzzle #2

```
(def musicals
  (letfn [(has-one-space? [musical]
            (-> musical
                (clojure.string/split #" ")
                count
                (= 2)))]
    (->> "resources/musicals.txt"
         slurp
         clojure.string/split-lines
         (filter has-one-space?)
         set)))

(count musicals) => 380
```

Puzzle #2

```
(def words
  (->> "resources/ni2.txt"
       slurp
       clojure.string/split-lines
       (map clojure.string/lower-case)
       set))

(count words) => 233614

(def goback
  (zipmap
    (map char "bcdefghijklmnopqrstuvwxyz")
    (map char "abcdefghijklmnopqrstuvwxyz")))
```

Puzzle #2

```
(defn change-one-let [musical n]
  (apply str
    (for [i (range (count musical))]
      (if (= i n)
        (goback (nth musical n))
        (nth musical i)))))

(change-one-let "cats" 3) => "catr"
```

Puzzle #2

```
(defn transform-musical [musical]
  (let [m-temp (-> musical
                   clojure.string/lower-case
                   (clojure.string/replace #" " "")
                   (clojure.string/replace #"[^a-z]" ""))]
    (-> m-temp
        count
        range
        ((partial map (partial change-one-let m-temp))))))

(transform-musical "Cats") => ("bats" "cts" "cass" "catr")
```

Puzzle #2

```
(def answer
  (->> musicals
       (map (juxt identity transform-musical))
       (remove (comp empty? (partial filter words) last))
       (map (juxt first (comp (partial filter words) last)))))

answer => (["Mamma Mia! " ("mammalia")] ["Miss 1917" ("liss")])
```

Puzzle #3

From May 8, 2018:

Name something in 11 letters that's a common household item. You can rearrange the first six letters to form a synonym of a word spelled by the middle three letters. What is the item, and what are the words?

Puzzle #3

```
(let [words (->> "resources/ospd3.txt"
                 slurp
                 clojure.string/split-lines)

      three-letter-words
        (set (filter (comp (partial = 3) count) words))

      six-letter-frequencies
        (set (map frequencies
                  (filter (comp (partial = 6) count) words)))]
```

Puzzle #3

PRINCETON UNIVERSITY

WordNet

A Lexical Database for English

Search...

- What is WordNet
- People
- News
- Use Wordnet Online
- Download
- Citing WordNet
- License and Commercial Use
- Related Projects
- Documentation

What is WordNet?

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the creators of WordNet and do not necessarily reflect the views of any funding agency or Princeton University.

When writing a paper or producing a software application, tool, or interface based on WordNet, it is necessary to properly cite the source. Citation figures are critical to WordNet funding.

About WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also

Note

Due to funding and staffing issues, we are no longer able to accept comment and suggestions.

We get numerous questions regarding topics that are addressed on our FAQ page. If you have a problem or question regarding something you downloaded from the "Related projects" page, you

Puzzle #3

```
00080242 04 n 01 insider_trading 0 001 @ 00079398 n 0000 | buying or selling corporate stock by a corporate officer or other insider on
 the basis of information that has not been made public and is supposed to remain confidential
00080474 04 n 01 naked_option 0 001 @ 13241600 n 0000 | a put or call option for which the seller or buyer has no underlying security p
osition
00080619 04 n 01 covered_option 0 001 @ 13241600 n 0000 | a put or call option backed by the shares underlying the option
00080743 04 n 02 call_option 0 call 4 004 @ 13241600 n 0000 #p 00349213 n 0000 + 00874002 v 0201 ! 00080968 n 0101 | the option to buy
a given stock (or stock index or commodity future) at a given price before a given date
00080968 04 n 02 put_option 0 put 0 003 @ 13241600 n 0000 #p 00349213 n 0000 ! 00080743 n 0101 | the option to sell a given stock (or s
tock index or commodity future) at a given price before a given date
00081174 04 n 01 straddle 1 001 @ 13241600 n 0000 | the option to buy or sell a given stock (or stock index or commodity future) at a g
iven price before a given date; consists of an equal number of put and call options
00081395 04 n 02 incentive_option 0 incentive_stock_option 0 001 @ 13241600 n 0000 | an option granted to corporate executives if the c
ompany achieves certain financial goals
00081572 04 n 02 buying 0 purchasing 0 006 @ 00079018 n 0000 + 02207206 v 0202 + 02207206 v 0101 ~ 00081836 n 0000 ~ 00082223 n 0000 ~
00082347 n 0000 | the act of buying; "buying and selling fill their days"; "shrewd purchasing requires considerable knowledge"
00081836 04 n 01 shopping 0 004 @ 00081572 n 0000 + 02326355 v 0101 + 02325968 v 0101 ~ 00082081 n 0000 | searching for or buying goods
 or services; "went shopping for a reliable plumber"; "does her shopping at the mall rather than down town"
00082081 04 n 01 marketing 1 002 @ 00081836 n 0000 + 02298471 v 0101 | shopping at a market; "does the weekly marketing at the supermar
ket"
00082223 04 n 02 mail-order_buying 0 catalog_buying 0 001 @ 00081572 n 0000 | buying goods to be shipped through the mail
00082347 04 n 02 viatication 0 viaticus 0 003 @ 00081572 n 0000 + 02824194 a 0201 + 02824194 a 0101 | purchasing insurance policies for
 cash from terminally ill policy holders
00082525 04 n 01 acceptance 3 003 @ 00077419 n 0000 + 01985557 a 0102 + 02236124 v 0101 | the act of taking something that is offered;
"her acceptance of the gift encouraged him"; "he anticipated their acceptance of his offer"
00082754 04 n 02 succession 1 taking_over 0 001 @ 00077419 n 0000 | acquisition of property by descent or by will
00082870 04 n 02 assumption 0 laying_claim 0 004 @ 00077419 n 0000 + 02381726 v 0101 + 02301825 v 0104 + 02274482 v 0101 | the act of t
aking possession of or power over something; "his assumption of office coincided with the trouble in Cuba"; "the Nazi assumption of pow
er in 1934"; "he acquired all the company's assets for ten million dollars and the assumption of the company's debts"
00083260 04 n 01 assumption 1 003 @ 00030358 n 0000 + 00632236 v 0101 ~ 00083448 n 0000 | the act of assuming or taking for granted; "y
our assumption that I would agree was unwarranted"
00083448 04 n 01 position 4 002 @ 00083260 n 0000 + 00716758 v 0102 | the act of positing; an assumption taken as a postulate or axiom

00083585 04 n 02 inheritance 0 heritage 0 002 @ 00077419 n 0000 + 02315525 v 0101 | hereditary succession to a title or an office or pr
operty
00083729 04 n 03 procurement 0 procurance 0 procural 0 004 @ 00077419 n 0000 + 02238770 v 0301 + 02238770 v 0201 + 02238770 v 0101 | th
e act of getting possession of something; "he was responsible for the procurement of materials and supplies"
```

Puzzle #3

```
(->>
  (slurp "resources/wordnet_data_cleaned.noun")
                                                  ;; Wordnet list of nouns
  clojure.string/split-lines                      ;; split lines
  (map #(clojure.string/split % #" "))
  (map (juxt #(nth % 4) second))                  ;; get lexicographer contents
  (filter (comp (partial = "06") last))           ;; get human-made objects
  (filter (comp (partial = 11) count first))      ;; filter out 11-letter words
  (remove (comp (partial re-find #"\_") first))   ;; remove entries with underscores
  (map first)                                     ;; isolate words
  (map (juxt identity #(subs % 0 6) #(subs % 4 7))) ;; get substrings
  (filter (comp three-letter-words (partial last))) ;; filter valid three-letter words
  (filter (comp six-letter-frequencies
                frequencies
                (partial second)))                ;; filter anagrams of 6-letter words
  ))
```

Puzzle #3

```
["planetarium" "planet" "eta"]
["restoration" "restor" "ora"]
["regimentals" "regime" "men"]
["restoration" "restor" "ora"]
["salinometer" "salino" "nom"]
["sclerometer" "sclero" "rom"]
["seismograph" "seismo" "mog"]
["speedometer" "speedo" "dom"]
["spherometer" "sphero" "rom"]
["stroboscope" "strobo" "bos"]
["submersible" "submer" "ers"]
["submersible" "submer" "ers"]
["suppressant" "suppre" "res"]
["thermograph" "thermo" "mog"]
["thermograph" "thermo" "mog"]
["thermometer" "thermo" "mom"])
```

Puzzle #4

From August 16, 2016:

Take the word EASILY. You can rearrange its letters to spell SAY and LEI. These two words rhyme even though they have no letters in common.

What is the longest familiar word you can find that can be anagrammed into two shorter words that rhyme but have no letters in common? The two shorter words must have only one syllable.

Puzzle #4

The CMU Pronouncing Dictionary

query | phonemes | about | | Speech at CMU | Speech Tools

- **Look up the pronunciation for a word or phrase in CMUdict (version 0.7b)**

C M U Dictionary [Look It Up]

☐ Show Lexical Stress

- **Download the current CMU dictionary from SourceForge**

http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict
Find an error? Please contact the maintainers! We will check it out. (See at bottom for contact information.)

Note: If you are looking for a dictionary for use with a speech recognizer, this dictionary is not the one that you are looking for.
For that purpose, see http://svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/sphinxdict or use this tool. You can also download the dict from https://github.com/Alexir/CMUdict/blob/master/cmudict-0.7b.

- **About the CMU dictionary**

The Carnegie Mellon University **Pronouncing Dictionary** is an open-source machine-readable pronunciation dictionary for North American English that contains over **134,000**

Puzzle #4

```
ABRIL  AH0 B R IH1 L
ABROAD  AH0 B R AO1 D
ABROGATE  AE1 B R AH0 G EY2 T
ABROGATED  AE1 B R AH0 G EY2 T IH0 D
ABROGATING  AE1 B R AH0 G EY2 T IH0 NG
ABROGATION  AE2 B R AH0 G EY1 SH AH0 N
ABROL  AH0 B R OW1 L
ABRON  AH0 B R AA1 N
ABRUPT  AH0 B R AH1 P T
ABRUPTLY  AH0 B R AH1 P T L IY0
ABRUPTNESS  AH0 B R AH1 P T N AH0 S
ABRUTYN  EY1 B R UW0 T IH0 N
ABRUZZESE  AA0 B R UW0 T S EY1 Z IY0
ABRUZZO  AA0 B R UW1 Z OW0
ABS  EY1 B IY1 EH1 S
ABS(1)  AE1 B Z
ABSALOM  AE1 B S AH0 L AH0 M
ABSARAKA  AE0 B S AA1 R AH0 K AH0
ABSCAM  AE1 B S K AE0 M
ABSCESS  AE1 B S EH2 S
ABSCOND  AE0 B S K AA1 N D
ABSCONDED  AE0 B S K AA1 N D AH0 D
ABSCONDING  AE0 B S K AA1 N D IH0 NG
ABSCONDS  AE0 B S K AA1 N D Z
```

Puzzle #4

```
(require '[clojure.math.combinatorics :as combo])

(def vowels #{"AA" "AH" "EH" "IH" "OW" "UH" "AE" "AO"
              "AY" "IY" "ER" "EY" "AW" "OY" "UW"})
```

Puzzle #4

```
(defn wipe-and-split [s]
  (clojure.string/split (clojure.string/replace s #"\d" "") #" "))


(defn find-monosyllables [syllables]
  (->> (wipe-and-split syllables)
       (filter vowels)
       count
       (= 1)))

(defn get-nucleus-and-coda [syllables]
  (->> (wipe-and-split syllables)
       (drop-while #(not (contains? vowels %)))
       (apply str)))
```

Puzzle #4

```
(defn clean-up-word [word]
  (clojure.string/replace word #"[^A-Z]" ""))


 (def one-syllable-words
   (->> "resources/cmudict-0.7b.txt"
        slurp
        clojure.string/split-lines
        (map #(clojure.string/replace-first % #" " ""))
        (map #(clojure.string/split % #" " 2))
        (filter #(find-monosyllables (second %)))
        (map (juxt (comp clean-up-word first) (comp identity second)))))

 (take 5 (shuffle one-syllable-words))
 => (["GRINS" "G R IH1 N Z"] ["GAZ" "G AE1 Z"] ["FRILL" "F R IH1 L"] ["DOTS" "D AA1
T S"] ["NORTHS" "N AO1 R TH S"])
```

Puzzle #4

```
(def rhymed-words-map
  (group-by (comp get-nucleus-and-coda second) one-syllable-words))

(def sets-of-words
  (map (comp (partial map first) val) rhymed-words-map))

(take 2 (shuffle sets-of-words))
 => (("BLOG" "BOG" "COG" "DOG" "FOG" "HAUG" "LOG" "ZAUGG") ("PAINTS" "SAINTS" "SAINT
S" "SAINTS" "TAINTS"))
```

Puzzle #4

```
(def get-word-pairs
  (->> (map #(combo/combinations % 2) sets-of-words)
       (remove nil?)
       flatten
       (partition 2)))

(defn check-if-words-have-shared-letters [[w1 w2]]
  (empty? (clojure.set/intersection (set (map char w1))
                                    (set (map char w2)))))

(def final-pairs
  (filter check-if-words-have-shared-letters get-word-pairs))

(count final-pairs) => 15081
```

Puzzle #4

```
(def frequencies-of-final-words
  (->> final-pairs
       (map (comp count (partial apply str)))
       frequencies
       (sort-by key)
       reverse))

frequencies-of-final-words => ([11 40] [10 188] [9 771] [8 2008] [7 3786] [6 4401]
[5 2563] [4 1000] [3 283] [2 41])

(defn find-pairs-by-combined-word-length [pairs n]
  (->> pairs
       (filter (comp (partial = n)
                     count
                     (partial apply str)))))
```

Puzzle #4

```
(def all-the-words
  (->> "resources/ni2.txt"
       slurp
       clojure.string/split-lines
       set))

 ; (count all-the-words) => 234936

(defn compare-pair-to-word [pair word]
  (let [p-result (-> (apply str pair)
                     (clojure.string/replace #"\d" "")
                     (clojure.string/lower-case))]
    (= (frequencies word) (frequencies p-result))))

 (compare-pair-to-word '("lei" "say") "easily") => true
```

Puzzle #4

```
(def final-answer
 (remove (comp empty? second)
         (for [p (find-pairs-by-combined-word-length final-pairs 11)]
           [p (filter #(compare-pair-to-word p %) all-the-words)])))

final-answer => ([("SIEW" "THROUGH") ("housewright")])
```

```
(def final-answer-10
 (remove (comp empty? second)
         (for [p (find-pairs-by-combined-word-length final-pairs 10)]
           [p (filter #(compare-pair-to-word p %) all-the-words)])))

final-answer-10 => ([("CROW" "NEAULT") ("counterlaw")] [("GROW" "NEAULT") ("outwr
angle")])
```

Puzzle #5

Take the four four-letter words LIMB, AREA, CORK and KNEE. Write them one under the other, and the four columns will spell four new words LACK, IRON, MERE, and BAKE.

L	I	M	B
A	R	E	A
C	O	R	K
K	N	E	E

This is called a double word square. I'd like you to find a double word square with 6-letter words. Specifically, your square must include the words PONIES, ACCEPT, SEARED and CAVIAR. These four words must be among the 12 common, uncapitalized six-letter words in the square. Can you do it?

Puzzle #5

```
(use 'clojure.core.logic)
(require '[clojure.core.logic.fd :as fd])


(def letters-list [\a \b \c \d \e \f \g \h \i \j \k \l \m
                   \n \o \p \q \r \s \t \u \v \w \x \y \z])

(def word-arrays
  (->> "resources/ospd3.txt"
       slurp
       clojure.string/split-lines
       (filter (comp (partial = 6) count))
       (map (comp seq char-array))
       set))

(def ponies '((\p \o \n \i \e \s)))
(def accept '((\a \c \c \e \p \t)))
(def seared '((\s \e \a \r \e \d)))
(def caviar '((\c \a \v \i \a \r)))
```

Puzzle #5

```
;; Take #1: Really brute force

(run* [a1 b1 c1 d1 e1 f1
       a2 b2 c2 d2 e2 f2
       a3 b3 c3 d3 e3 f3
       a4 b4 c4 d4 e4 f4
       a5 b5 c5 d5 e5 f5
       a6 b6 c6 d6 e6 f6]
      (membero (seq [a1 b1 c1 d1 e1 f1]) word-arrays)
      (membero (seq [a2 b2 c2 d2 e2 f2]) word-arrays)
      (membero (seq [a3 b3 c3 d3 e3 f3]) word-arrays)
      (membero (seq [a4 b4 c4 d4 e4 f4]) word-arrays)
      (membero (seq [a5 b5 c5 d5 e5 f5]) word-arrays)
      (membero (seq [a6 b6 c6 d6 e6 f6]) word-arrays)
      (membero (seq [a1 a2 a3 a4 a5 a6]) word-arrays)
      (membero (seq [b1 b2 b3 b4 b5 b6]) word-arrays)
      (membero (seq [c1 c2 c3 c4 c5 c6]) word-arrays)
      (membero (seq [d1 d2 d3 d4 d5 d6]) word-arrays)
      (membero (seq [e1 e2 e3 e4 e5 e6]) word-arrays)
      (membero (seq [f1 f2 f3 f4 f5 f6]) word-arrays)
      (conde
        (membero (seq [a1 b1 c1 d1 e1 f1]) ponies)
        (membero (seq [a2 b2 c2 d2 e2 f2]) ponies)
        (membero (seq [a3 b3 c3 d3 e3 f3]) ponies)
        (membero (seq [a4 b4 c4 d4 e4 f4]) ponies)
        (membero (seq [a5 b5 c5 d5 e5 f5]) ponies)
```

Puzzle #5

```
;; Take 2: Try in a grid

;; ACCEPT
;; **A*O*
;; **V*N*
;; **I*I*
;; SEARED
;; **R*S*

(run* [a1 b1 c1 d1 e1 f1
       a2 b2 c2 d2 e2 f2
       a3 b3 c3 d3 e3 f3
       a4 b4 c4 d4 e4 f4
       a5 b5 c5 d5 e5 f5
       a6 b6 c6 d6 e6 f6]
      (membero (seq [a1 b1 c1 d1 e1 f1]) accept)
      (membero (seq [a2 b2 c2 d2 e2 f2]) word-arrays)
      (membero (seq [a3 b3 c3 d3 e3 f3]) word-arrays)
      (membero (seq [a4 b4 c4 d4 e4 f4]) word-arrays)
      (membero (seq [a5 b5 c5 d5 e5 f5]) seared)
      (membero (seq [a6 b6 c6 d6 e6 f6]) word-arrays)
      (membero (seq [a1 a2 a3 a4 a5 a6]) word-arrays)
      (membero (seq [b1 b2 b3 b4 b5 b6]) word-arrays)
      (membero (seq [c1 c2 c3 c4 c5 c6]) caviar)
      (membero (seq [d1 d2 d3 d4 d5 d6]) word-arrays)
      (membero (seq [e1 e2 e3 e4 e5 e6]) ponies)
      (membero (seq [f1 f2 f3 f4 f5 f6]) word-arrays))
```

Puzzle #5

```clojure
;; Take 3: Try in a filtered grid

(defn make-filtered-word-arrays
  [letter1 position1 letter2 position2]
  (->> word-arrays
       (filter (comp (partial = letter1) #(nth % position1)))
       (filter (comp (partial = letter2) #(nth % position2)))
       (map (comp seq char-array))
       ))

(def word-arrays-as (make-filtered-word-arrays \a 0 \s 4))
(def word-arrays-ce (make-filtered-word-arrays \c 0 \e 4))
(def word-arrays-er (make-filtered-word-arrays \e 0 \r 4))
(def word-arrays-td (make-filtered-word-arrays \t 0 \d 4))
(def word-arrays-ao (make-filtered-word-arrays \a 2 \o 4))
(def word-arrays-vn (make-filtered-word-arrays \v 2 \n 4))
(def word-arrays-ii (make-filtered-word-arrays \i 2 \i 4))
(def word-arrays-rs (make-filtered-word-arrays \r 2 \s 4))


(* 46 359 51 16 57 49 53 65) => 129658980054240
```

Puzzle #5

```
(run* [a1 b1 c1 d1 e1 f1
       a2 b2 c2 d2 e2 f2
       a3 b3 c3 d3 e3 f3
       a4 b4 c4 d4 e4 f4
       a5 b5 c5 d5 e5 f5
       a6 b6 c6 d6 e6 f6]
      (membero (seq [a1 b1 c1 d1 e1 f1]) accept)
      (membero (seq [a2 b2 c2 d2 e2 f2]) word-arrays-ao)
      (membero (seq [a3 b3 c3 d3 e3 f3]) word-arrays-vn)
      (membero (seq [a4 b4 c4 d4 e4 f4]) word-arrays-ii)
      (membero (seq [a5 b5 c5 d5 e5 f5]) seared)
      (membero (seq [a6 b6 c6 d6 e6 f6]) word-arrays-rs)
      (membero (seq [a1 a2 a3 a4 a5 a6]) word-arrays-as)
      (membero (seq [b1 b2 b3 b4 b5 b6]) word-arrays-ce)
      (membero (seq [c1 c2 c3 c4 c5 c6]) caviar)
      (membero (seq [d1 d2 d3 d4 d5 d6]) word-arrays-er)
      (membero (seq [e1 e2 e3 e4 e5 e6]) ponies)
      (membero (seq [f1 f2 f3 f4 f5 f6]) word-arrays-td))
```

Puzzle #5

```
\s \a) (\a \h \o \r \s \e) (\a \i \r \e \s \t) (\a \l \d \o \s \e) (\a \l \m \o \s \t) (\a \m \b \u \s \h) (\a \m \i \d \s \t) (\a \n \e \n \s \t) (\a \n \k \u \s \h)
(\a \o \r \i \s \t) (\a \p \p \o \s \e) (\a \p \t \e \s \t) (\a \r \g \o \s \y) (\a \r \i \o \s \e) (\a \r \i \o \s \i) (\a \r \i \o \s \o) (\a \r \k \o \s \e) (\a \r
\o \u \s \e) (\a \r \r \e \s \t) (\a \r \t \i \s \t) (\a \s \p \i \s \h) (\a \s \s \e \s \s) (\a \s \s \i \s \t) (\a \t \t \e \s \t) (\a \u \g \u \s \t) (\a \u \r \i \
s \t) (\a \u \t \i \s \m) (\a \v \e \r \s \e) (\a \v \u \l \s \e) (\a \w \l \e \s \s))
user=> (def word-arrays-ce (make-filtered-word-arrays \c 0 \e 4))
#'user/word-arrays-ce
user=> (def word-arrays-er (make-filtered-word-arrays \e 0 \r 4))
#'user/word-arrays-er
user=> (def word-arrays-td (make-filtered-word-arrays \t 0 \d 4))
#'user/word-arrays-td
user=> (def word-arrays-ao (make-filtered-word-arrays \a 2 \o 4))
#'user/word-arrays-ao
user=> (def word-arrays-vn (make-filtered-word-arrays \v 2 \n 4))
#'user/word-arrays-vn
user=> (def word-arrays-ii (make-filtered-word-arrays \i 2 \i 4))
#'user/word-arrays-ii
user=> (def word-arrays-rs (make-filtered-word-arrays \r 2 \s 4))
#'user/word-arrays-rs
user=> (run* [a1 b1 c1 d1 e1 f1
  #_=>        a2 b2 c2 d2 e2 f2
  #_=>        a3 b3 c3 d3 e3 f3
  #_=>        a4 b4 c4 d4 e4 f4
  #_=>        a5 b5 c5 d5 e5 f5
  #_=>        a6 b6 c6 d6 e6 f6]
  #_=>       (membero (seq [a1 b1 c1 d1 e1 f1]) accept)
  #_=>       (membero (seq [a2 b2 c2 d2 e2 f2]) word-arrays-ao)
  #_=>       (membero (seq [a3 b3 c3 d3 e3 f3]) word-arrays-vn)
  #_=>       (membero (seq [a4 b4 c4 d4 e4 f4]) word-arrays-ii)
  #_=>       (membero (seq [a5 b5 c5 d5 e5 f5]) seared)
  #_=>       (membero (seq [a6 b6 c6 d6 e6 f6]) word-arrays-rs)
  #_=>       (membero (seq [a1 a2 a3 a4 a5 a6]) word-arrays-as)
  #_=>       (membero (seq [b1 b2 b3 b4 b5 b6]) word-arrays-ce)
  #_=>       (membero (seq [c1 c2 c3 c4 c5 c6]) caviar)
  #_=>       (membero (seq [d1 d2 d3 d4 d5 d6]) word-arrays-er)
  #_=>       (membero (seq [e1 e2 e3 e4 e5 e6]) ponies)
  #_=>       (membero (seq [f1 f2 f3 f4 f5 f6]) word-arrays-td))
([\a \c \c \e \p \t \c \l \a \m \o \r \r \a \v \i \n \e \o \r \i \g \i \n \s \e \a \r \e \d \s \t \r \e \s \s])
user=>
```

Puzzle #5

A	C	C	E	P	T
C	L	A	M	O	R
R	A	V	I	N	E
O	R	I	G	I	N
S	E	A	R	E	D
S	T	R	E	S	S

github.com/msszczep/ npr_sunday_puzzle_solutions

msszczep / **npr_sunday_puzzle_solutions**

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Using code to solve the NPR Sunday Puzzle

Edit

Manage topics

56 commits | **1** branch | **0** releases | **1** contributor

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

msszczep Solutions for December 2018 puzzles. Latest commit 3923ab8 7 days ago

resources	Assorted puzzle solution attempts.	14 days ago
src	Solutions for December 2018 puzzles.	7 days ago
README.md	Two more non-NPR challenges.	3 years ago
project.clj	Updates for October 7, 2018	2 months ago

README.md