U5TBHUM8B : Oh that would be ideal! How would I do that with a specific file that's in the zip file? (One of multiple that are zipped in the archive)
U4VDXB2TU : I'm live translating this from some groovy code I wrote a while back (we'll see if I get lynched here), but you can iterate through the entries within a zip file and then call the ZipEntry `(.getInputStream zip-file zip-entry)` and then I believe you should be able to use slurp to get the contents of the input stream
U4VDXB2TU : to find the right entry within the zip you can filter on zipEntry.name
U4VDXB2TU : cigrainger: give me a few - will fire up a repl and see if I can cook up some example code. I'm not senior with clojure but I have spent a lot of time with file manipulations and zip files :)
U5TBHUM8B : Awesome! Thanks. I'll play around with your suggestion re: ZipEntry as well. That was the general direction I was going but I couldn't find much and I'm not great with clojure yet either.
U5YHX0TQV : <@U4VDXB2TU> Have a look at <https://stackoverflow.com/questions/5419125/reading-a-zip-file-using-java-api-from-clojure#5419767>
U28947274 : Hey folks,I have the following snippet:
```
(let [coll [1 2 3]]
   (map #(println :test %) coll)
   (map #(println :test2 %) coll))
```

This results in:
:test2 1
:test2 2
:test2 3
(nil nil nil)

And not my expectation:
:test 1
:test 2
:test 3
:test2 1
:test2 2
:test2 3
(nil nil nil)

Can anyone explain this behavior?

U5YHX0TQV : <@U28947274> Yes, don't use map to execute side-effecting functions like println since map basically returns a lazy seq
U5YHX0TQV : you can wrap your map's in a (doall ...) call, but it would be more idiomatic to use doseq for this case
U4VDXB2TU : ```(ns zip-files.core
  (:import (java.util.zip ZipFile)))

(defn get-entry-data [zip-file-path entry-name]
  (let [zip-file (ZipFile. (<http://clojure.java.io/file|clojure.java.io/file> zip-file-path))
        entries (enumeration-seq (.entries zip-file))
        matching   (filter #(= entry-name (.getName %)) entries)]
    (if (not-empty matching)
      (slurp (.getInputStream zip-file (first matching)))
      (println "no entry" entry-name "found!"))))

```

U4VDXB2TU : ```(get-entry-data "test.zip" "test.txt")
=&gt; "Hello World!\n\n"
```

U28947274 : Great, thanks :slightly_smiling_face:
U4VDXB2TU : where test.txt was one of many files within test.zip and the contents of test.txt were "Hello World!\n\n"
U4VDXB2TU : returns a string in this case or nil if no entry was found
U4VDXB2TU : <@U5YHX0TQV> thanks for the so link - looks more or less like what I came up with

U5TBHUM8B : Awesome! Thank you!
U5YHX0TQV : And be careful with slurp'ing in data, you'll have all in memory
U38J3881W : Hey! I was wondering if repeated subvecs would ever allow the excluded data to be garbage collected if there was no way to access it any more? Or would I need to use `(vec (rest x))` if I wanted to _actually_ drop the first item in a vector?
U38J3881W : I would use a persistent queue but I want in place updates too, performance of `(vec (rest x))` being O(n) is a non-issue for me too, looking for the "idiomatic" solution more than anything :slightly_smiling_face:
U06HTKDMF : damn didn't realize `for` re-evals the inner range expression
U06HTKDMF : ```boot.user=> (for [x (do (println "x") [1 2]) y (do (println "y") [1 2])] [x y])x
y
y
([1 1] [1 2] [2 1] [2 2])```

U060FKQPN : it's by design
U060FKQPN : how would `(for [x [[1 2] [3 4]] y x] y)` work otherwise
U06HTKDMF : i was p confused for a bit because i was using `(q/random ...)` (from quil) inside the inner one xD
U06HTKDMF : yea makes sense :open_mouth:
U5XMV6DQT : <@U28947274> or you can make something like that```
(let [coll [1 2 3]
      res (concat (map #(vector :test %) coll)
                  (map #(vector :test2 %) coll))]
            (doseq [x res]
              (apply println x)))
```

U5XMV6DQT : I prefer not to mix side-effects with pure stuff
U5N8R3NF4 : Someone Please suggest real-time-messaging clojure library. I might also need Screen Sharing and video chat.
U051SS2EU : jgeraert: `doseq` works great for this, but there's also `run!` which like this usage of map takes a function and a collection as args, but is run for side effects eagerly.
U051SS2EU : <@U28947274> meant to tag you above
```