U0CHY4VNW : <@U11BV7MTK> Thanks I'll take a look at that
U11BV7MTK : if in CIDER, apropos, grimoire, browse ns and eldoc are quite nice
U11BV7MTK : i've been using grimoire quite a bit
U11BV7MTK : That's awesome. It's just so handy. Much appreciated
U61KCTX8S : can anyone explain me what a java.lang.IllegalArgumentException: array element type mismatchmeans?

U0NCTKEV8 : it means the array type you are calling a java method with doesn't match
U61KCTX8S : so fi it expects an array of ints and i am using an array of strings
U61KCTX8S : thanks
U0CMVHBL2 : Yes, there is. You can verify yourself whether this is so using identical? on the pieces you hope are being shared, as shown below
U0CMVHBL2 : user=> (def mymap {:mykey [{:A 1, :B 2} {:A 2, :B 2} {:A 3, :B 2}]})#'user/mymap
user=> (def m2 (assoc mymap :mykey (map #(assoc % :A (inc (:A %))) (:mykey mymap))))
#'user/m2
user=> (identical? (-> mymap :mykey (nth 0) :B) (-> m2 :mykey (nth 0) :B))
true

U0DATSMH6 : Hmm, my gut tells me that's not a good example because: ```user=> (identical? 2 2)
true
```

U0DATSMH6 : This would be a better test:```
user=&gt; (identical? {:test 1} {:test 1})
false
user=&gt; (def mymap {:mykey [{:A 1, :B {:test 1}} {:A 2, :B {:test 1}} {:A 3, :B {:test 1}}]})
#'user/mymap
user=&gt; (def m2 (assoc mymap :mykey (map #(assoc % :A (inc (:A %))) (:mykey mymap))))
#'user/m2
user=&gt; (identical? (-&gt; mymap :mykey (nth 0) :B) (-&gt; m2 :mykey (nth 0) :B))
true
```

U0CMVHBL2 : The better test works, too, as I expected it would. Thanks.
U0DATSMH6 : Also, for clarity, there isn't any structural sharing going on _within_ the vectors themselves - only at the same key "paths" between `mymap` and `m2`:```
user=> (identical? (-> mymap :mykey first :B) (-> mymap :mykey last :B))
false
user=> (identical? (-> m2 :mykey first :B) (-> m2 :mykey last :B))
false
user=> (-> mymap :mykey first :B)
{:test 1}
user=> (-> mymap :mykey last :B)
{:test 1}
```

U0DATSMH6 : Yeah - this is a cool approach to discover the structural sharing. I hadn't thought of using `identical?` for this.
U3JURM9B6 : keep, for, map -- they want pure functions and return a lazy list
U3JURM9B6 : I want something which is okay to pass an unpure function to ... and returns a strict list
U3JURM9B6 : in good clojure style, do people do (map impure-function ...) or do we do something else when we have to run an impure function and also get its return value ?
U051KLSJF : <@U3JURM9B6> usually you'll use `doall`
U051KLSJF : if you care about the return value
U1ALMRBLL : <@U3JURM9B6> it is fine, as far as I know, to pass a function with side-effects to `keep`, `filter`, etc -- the gotcha is that you should not *expect* that your code will necessarily execute. So, your side effect may happen, and if you don't mind, great -- but, your side effect might *not* happen, and that's why it says to avoid impure functions (or in some cases, may get called *more than once*).
```