U04V70XH6 : `even?` and `odd?` are only defined for numeric inputs.
U04V70XH6 : Hence <@U0NCTKEV8>'s suggestion to use `(s/and number? odd?)`
U0CKDHF4L : ok a simpler version works: ```(s/explain (s/cat :this (s/* (s/coll-of odd?)) :that (s/coll-of even?)) '((7 3 1) (9 7 3) [2 4 6]))```
U0CKDHF4L : using ```(s/def :q/b (s/coll-of (s/and number? even?)))``` does not work
U0CKDHF4L : ```(s/explain (s/cat :this (s/* (s/coll-of odd?)) :that (s/keys :req [:q/b])) '((7 3 1) (9 7 3) {:q/b [2 4 6]}))IllegalArgumentException Argument must be an integer: [:q/b [2 4 6]] clojure.core/even? (core.clj:1383)```

U0NCTKEV8 : you need to do it for odd? too
U0CKDHF4L : however, ```(s/explain (s/cat :that (s/keys :req [:q/b])) '({:q/b [2 4 6]}))Success!```

U0CKDHF4L : oh
U0NCTKEV8 : if I recall odd? is just (not (even? ...))
U0CKDHF4L : ah yes that works ok
U0CKDHF4L : ```(s/explain (s/cat :this (s/* (s/coll-of (s/and number? odd?))) :that (s/keys :req [:q/b])) '((7 3 1) (9 7 3) {:q/b [2 4 6]}))Success!```

U0CKDHF4L : please explain why!?
U0NCTKEV8 : because odd? and even? as predicates aren't total, so they will throw exceptions when not passed numbers instead of returning false
U0NCTKEV8 : s/and tries each predicate it order
U0CKDHF4L : yes but why should they be passed non-numbers ?
U0NCTKEV8 : because in order for s/* to stop matching it has to fail a match
U0NCTKEV8 : otherwise it would match everything
U0CKDHF4L : oh yes I see! ```(odd? {})IllegalArgumentException Argument must be an integer: {} clojure.core/even? (core.clj:1383)```

U0CKDHF4L : i had misunderstood how s/* worked
U0CKDHF4L : thanks so much!
U1ALMRBLL : serg: I saw you deleted this. I noticed if you removed `into []` and just used the seq instead of a vector, it works. Did you solve it?
U0W0JDY4C : this is perhaps a little convoluted.. but now i'm curious just for curiosity's sake. if I had a function that maps over some collection, invokes a provided fn, and returns the value of that invoked fn as a different shape, how does this work with compose?```
(defn do-map [f coll]
  (map (fn [[foo val baz]] [foo (f val) baz]) coll)) ;; &lt;- notice the [a b c]
;; wont work unless xform-2 and xform-1 are "aware" of the [a b c] shape
(do-map (comp xform-2 xform-1) coll)
```

U0W0JDY4C : is there a generic way to compose transformations and, after each xform, re-shape the data?
U0W0JDY4C : the `do-map` couldn't really know whether `f` was composed or not, it just sees a function. so it would be hard to do something like `(-&gt; coll f0 reshape f1 reshape ...)`
U051SS2EU : <@U0W0JDY4C> sounds like you want `(fn [f] (fn [[a b c]] [a (f b) c]))`
U051SS2EU : then you can wrap your functions in it?
U051SS2EU : and -&gt; isn't a composer of functions, it's a rewriter of forms, if you use `comp` it's easier to get the right behavior via wrapping functions without jumping through syntactic hoops
```