U04V70XH6 : Change `:q/b` to `(s/coll-of even?)`
U04V70XH6 : (you can specify a minimum length of 1 to require at least one element)
U0NCTKEV8 : that'll be the next issue, which will get him a weird failing spec
U0NCTKEV8 : right now the error from feeding a non-number to odd? is bubbling out and killing the checking
U04V70XH6 : Ah, I see where you're going now. I was jumping ahead.
U04V70XH6 : <@U0CKDHF4L> "total" means "defined for all inputs"
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