U051HUZLD : ah, it's not wrapped in macro.
U050SC7SV : yep, I prefer eval personally for that stuff. a macro def will stay here in all its uselessness after you used it to generate your spec
U050SC7SV : depends if you need to do that a lot or not
U051HUZLD : I wanted a macro initially, because I have too many `s/cat`s where I basically reuse spec names as dispatch keys
U051HUZLD : figured I'd try to just re-use spec names instead of coming up with throw-away names time and time again.
U051HUZLD : <@U2PGHFU5U> thanks, I just forgot about ~@ splicing. it's all good now
U051HUZLD : while we are on macro subject: how do grown ups validate macro's input (e.g. with spec)?
U09LZR36F : I use spec lately. asserts are great too though.
U064X3EF3 : <@U051HUZLD> s/conform
U064X3EF3 : will give you the destructured version. if it's ::s/invalid, then s/explain.
U051HUZLD : is there a way to get fn's arity? like```
(arity filter)
;; => [1 2]
```
actually I am not even sure why I am asking this :dafuq:

U064X3EF3 : no
U06B8J0AJ : <@U051HUZLD> In cljs, you can do `(.-length (.-constructor (.-prototype render)))`
U06B8J0AJ : Where `render` would be the function, for example
U04V4KLKC : If function was defined with defn - `(:arglists (meta #'filter))`
U06B8J0AJ : Or why not `(-&gt; render .-prototype .-constructor .-length)`. We're not barbarians after all.
U3JURM9B6 : anyone else find code written via reduce to be "obfuscated" ?
U06B8J0AJ : <@U3JURM9B6> Example?
U3JURM9B6 : for some reason, when writing code involving reduce, it's always:1. how can I write this imperatively ? 2. then I reformulate it as reduce

U3JURM9B6 : <@U06B8J0AJ>: no concrete example, just that most of the time, I want to do:```
(let [state (atom ...)]
  (doseq ...))
```

U3JURM9B6 : then I end up "inverting" the doseq / modification to the @state atom in order to get my reduce code
U06B8J0AJ : <@U3JURM9B6> There's nothing inherently wrong with that, in my opinion. For me, it's a quite common pattern that I pull out the transducers once I start noticing boilerplate and repeating patterns in the code.
U3JURM9B6 : <@U06B8J0AJ>: have you used Haskell? the haskell solution to this would be the 'state' monad, then an external sequence_
U06B8J0AJ : <@U3JURM9B6> Unfortunately I haven't used it beyond the tutorial on the home page (which is quite nice)
U06B8J0AJ : `map` does seem to be easier to visualize than `reduce` though. I think `map` corresponds more to everyday patterns of life. You can imagine walking along a row of potted plants and watering each for example, getting a row of watered plants.
U06B8J0AJ : But what would be the `reduce` version of that? Repot them in one large pot, one plant at a time?
U06B8J0AJ : But also, the order in which they were repotted would somehow matter. I don't know, it's not _as_ straightforward.
U66SFLTPT : <@U3JURM9B6> in Haskell how would you solve the problem?
U66SFLTPT : `foldr` over a sequence?