U0CKDHF4L : I'm trying to think of a case where you describe a nested structure of collections by a flat spec
U0GC1C09L : does clojure have a representation of infinity that can work with mathematical operators? something like `(<= 1 2 Infinity)` ?
U0CKDHF4L : ```Double/POSITIVE_INFINITY```
U0CKDHF4L : or ```Number.POSITIVE_INFINITY``` in CLJS
U0GC1C09L : !! thanks :slightly_smiling_face:
U1NGX4Z6F : hey guys
U1NGX4Z6F : what's you're preferred method of checking for nils before assignation ?
U060FKQPN : <@U0GC1C09L> see also <https://dev.clojure.org/jira/browse/CLJ-1074>
U1NGX4Z6F : i just realized I am repeating `(:username respone)` when I `(if-not (nil? (:username response) (:username response) fallback-value))`
U1NGX4Z6F : isn't there a syntactic sugar around this ?
U0CKDHF4L : ```(or maybe-nil-thing default-value)```
U060FKQPN : assuming you also want to exclude `false`
U0CKDHF4L : (assuming that)
U1NGX4Z6F : ok
U1NGX4Z6F : (-> good sounds)
U287C9JRE : <@U1NGX4Z6F> I end up writing functions like `(?apply pred f val &amp; args)` for these purposes. If the predicate fails, then pass the value along
U0CKDHF4L : but ```(if (nil? maybe-nil-thing) default-value maybe-nil-thing)```
U0CKDHF4L : is clear
U060FKQPN : if you specifically want to exclude nil and not false, there's `if-some`
U060FKQPN : `(if-some [u (:username foo)] u default)`
U1NGX4Z6F : there-s also (some->
U0CKDHF4L : if-let
U1NGX4Z6F : which short circuits when nil
U643TBNP4 : <@U1NGX4Z6F> I usually use some? when checking for nil. <https://clojuredocs.org/clojure.core/some_q>
U1NGX4Z6F : I think points for brevity go to `(or value fall-back)`
U1NGX4Z6F : but yeah, some? is nice