U06HHF230 : n/p
U5QCSK76C : Thanks guys! I'll check them out
U61HA86AG : to me, data-driven means that the most fundamental abstraction in your code is just clojure's data structures, rather than functions, or macros, or protocols, etc
your program is controlled and organised in terms of data, with `map`, `assoc` and so forth being your tools of choice

<http://www.lispcast.com/data-functions-macros-why>
<http://blog.cognitect.com/blog/2016/6/28/the-new-normal-data-leverage>

U61HA86AG : you could say that datascript and datomic are data-driven databases, compared to something like MariaDB
U61HA86AG : there's also this interview with Rich - <https://gist.github.com/rduplain/c474a80d173e6ae78980b91bc92f43d1> (ctrl-f for "information")
U1LCB75M2 : lwhorton: 2) map and multimethod dispatch on some key. any code can consume nested maps and mutate it conforming to the spec. I would avoid the 1) OO trap
U0W0JDY4C : what's the issue with using records and protocols in #1?
U1LCB75M2 : unnecessary bundling data + behaviour.
U0CGFT70T : Anything more idomatic?
U050MP39D : aside from "I hope this isn't a real system" the clojure looks good
U1ALMRBLL : <@U0CGFT70T> it is clear and you will find different opinions on this as it's largely a question of style and idioms.personally, I would either pass in the valid users:
```
(defn valid-user? [users username password] ...```

or, close over them and return a function:
```
(defn valid-user-fn [users]
  (fn [username password]

```

U1ALMRBLL : this way you avoid the global `def` and it becomes more easily testable
U1ALMRBLL : stylistically, I don't like the threading macro as you're not really transforming a piece of data, threading it, as it's often used. I'd prefer `some`:```
(defn valid-user-fn [users]
  (fn [username password]
    (some (fn [{u ::user/name p ::user/password}]
            (and (= u username)
                 (= p password)))
          users)))
```

U1KE7MFDY : ```(defn valid-users
  [username password]
  (->> users
       (map (juxt :user/name :user/password))
       (some #(= [username password] %))))
```

U1ALMRBLL : Now you can make your function:```
(let [valid-user? (valid-user-fn master-users-list)]
  ;; now use valid-user? as you wish...
  (if (valid-user? "bob" "abc123")
    ...```
and if you want to use a different set of username/passwords, you're not tied to any particular one. just make the new `valid-user?` function by calling `valid-user-fn` with your list

U0CGFT70T : <@U1KE7MFDY> <@U1ALMRBLL> thanks this is what I was looking for...  both great suggestions. <@U050MP39D> lol, right not a real system...lol... just creating an om-next tutorial, so just for edification purposes! :slightly_smiling_face:
```

U0W0JDY4C : hmm. i always seem to have a hard time pinning down when to use a protocl
U1LCB75M2 : generally, it's useful when 1) you're interop-ing w/ java (so you can use `extend-protocol` and do type-based dispatch) 2) you create a protocol + record to manage state lifecycle
U1LCB75M2 : otherwise, if you're just manipulating data (not state), simple data structures + the ad-hoc dispatch available w/ multimethods works nice and is more flexible/open
U1LCB75M2 : in other words... type-based (in Clojure case, actual Java types) dispatch = nominal typing, ad-hoc dispatch = more like structural typing
U0W0JDY4C : much to ponder. thanks for letting me take your time, btw
U1LCB75M2 : :+1: