Why did we extend clojure.contrib.monads?

- We wanted a modular, monadic interpreter example
  - Effect requirements: errors, mutable state, continuations, logging and an environment
- This requires new monads
  - error-m
  - env-m [for completeness and testing]
- Also new monad transformers
  - cont-t
  - env-t
  - writer-t

Interpreter modularity requirements

- Our interpreter languages are built out of plug-and-play language fragments
- The interpreter monad is refined independently of the language fragments
- Changing the monad stack should affect the fragments and their assembly as little as possible
- Unfortunately, direct top-level functions as monad operations are not modular
- Their implementation must change when the structure of the monad stack changes

Implementation possibilities

- Different functions for each monad stack variation?
  - Combinatorial explosion of operation variants
  - Operation variants need to be matched to monads (difficult and error-prone)
- Better alternative: operation lifting
  - *Compute* the changes in monad operations when the transformer stack changes
  - Haskell implements lifting with typeclasses
  - Problem: this is a dynamically typed setting

Solution

- Add some indirection – monadic operations are now retrieved via the monad structure
- with-monad, domonad, etc. make these structure fields available as bound identifiers
- Monad transformers implement "uniform operation lifting" behind-the-scenes
  - This depends on some carefully-chosen auxiliary functions
- Modified library at: monad-tutorial/clojure/newmonads.clj
- This library is not polished (yet) – it just met our requirements for this tutorial example

New monad operations

- Error monad - m-fail
- State monad – m-get, m-put
- Environment monad – m-capture-env, m-local-env
- Continuation monad – m-call-cc
- Writer monad – m-write, m-listen, m-censor
- All optional, like m-zero and m-plus

- What happens if you have more than one kind of the same monad in the stack?
  (e.g. state-t (state-m))