U051SS2EU : fn* is implemented in java code
U051SS2EU : right, remembering that it's an fn that can't destructure is probably enough
U65U08BB4 : could you please give an example? of the difference?
U051SS2EU : ```+user=> ((fn [[a]] a) [1])1
+user=> ((fn* [[a]] a) [1])
CompilerException java.lang.IllegalArgumentException: fn params must be Symbols,
compiling:(NO_SOURCE_PATH:2:2)```

U051SS2EU : [a] as a parameter is a destructure that says "bind the first element of this sequencable input to the name a"
U051SS2EU : fn* doesn't understand that syntax
U65U08BB4 : so the difference is only about the destructuring of the parameters? fn supports it, while fn* doesn't?
U051SS2EU : that's the main one, I forget if it's the only one
U65U08BB4 : hmm~ in lazy-seq macro:
U65U08BB4 : boot.user=> (source lazy-seq)(defmacro lazy-seq
  "Takes a body of expressions that returns an ISeq or nil, and yields
  a Seqable object that will invoke the body only the first time seq
  is called, and will cache the result and return it on all subsequent
  seq calls. See also - realized?"
  {:added "1.0"}
  [& body]
  (list 'new 'clojure.lang.LazySeq (list* '^{:once true} fn* [] body)))

U65U08BB4 : ```boot.user=> (source lazy-seq)(defmacro lazy-seq
  "Takes a body of expressions that returns an ISeq or nil, and yields
  a Seqable object that will invoke the body only the first time seq
  is called, and will cache the result and return it on all subsequent
  seq calls. See also - realized?"
  {:added "1.0"}
  [& body]
  (list 'new 'clojure.lang.LazySeq (list* '^{:once true} fn* [] body)))```

U65U08BB4 : why is fn* preferred here~? is it some performance concern, as fn* is the basic one?
U051SS2EU : right, destructuring is defined in terms of lazy-seq
U051SS2EU : so lazy-seq can't use fn, which destructures
U051SS2EU : source of fn shows you how it builds the form for fn* - it does many things, some of which are lazy seq generating
U65U08BB4 : I will try to read the source of fn~ I probably digged too deep of the dark magics, as I just started learning Clojure~
U051SS2EU : another difference, discovered by reading the source of fn, is that fn* can't do preconditions - or at least not reasonably ```+user=> ((fn [a] {:pre [(number? a)]} a) :a)AssertionError Assert failed: (number? a)
user/eval36/fn--37 (NO_SOURCE_FILE:7)
+user=> ((fn* [a] {:pre [(number? a)]} a) :a)
:a
```

U65U08BB4 : thanks for the explanation, I probably need some more time to digest it~
U051SS2EU : yeah- all you need to remember is fn is fancier, does some things fn* can't, but code that exists before fn is defined (code that fn uses for example...) has to use fn*
U65U08BB4 : got it~ thx~!
U06B8J0AJ : So I'm parsing XML. That's enough for wanting to shoot myself in the face, but on top of it I need to check that it conforms to a certain shape, because the source(s) can be unreliable.
U06B8J0AJ : I'm using `clojure.xml/parse` for this.
U06B8J0AJ : Now, spec seems like the right tool for conforming. However, I've also heard tales of spec being able to destructure as well.
U06B8J0AJ : Where can I read about how the destructuring works, specifically? If I can avoid hand-disassembling the stain upon humanity that is the output of `parse`, I'd be delighted.
U050487DQ : <@U0BKWMG5B> invite sent
U06B8J0AJ : I just discovered zippers
```