Clojure Macroexpand Reference

purelyfunctional.tv/clojure-macros

In these examples, we define a macro `square` **like this:**

```
(defmacro square [x]
  `(let [x# ~x]
     (when (number? x#)
       (* x# x#))))
```

Expand only the toplevel macro call one time:

```
(macroexpand-1 '(square 10))

   => (clojure.core/let [x_12 10]
        (clojure.core/when (clojure.core/number? x_12)
          (clojure.core/* x_12 x_12)))
```

square gets expanded but let and when remain

they are both macros

notice vars are now namespace-qualified by backquote

Expand the toplevel macro call until the toplevel is no longer a macro call:

```
(macroexpand '(square 10))

   => (let* [x_13 10]
        (clojure.core/when (clojure.core/number? x_13)
          (clojure.core/* x_13 x_13)))
```

square gets expanded as well as let because they are both toplevel

when remains because it is nested

let* is a special form, so it's has no namespace

Expand all of the macro calls in the expression until no macro calls exist:

```
(clojure.walk/macroexpand-all '(square 10))

   => (let* [x_14 10]
        (if (clojure.core/number? x_14)
          (do
            (clojure.core/* x_14 x_14))))
```

special forms

square, let, and when are expanded

there are no more macros

Keybindings

Emacs (CIDER)		vim (fireplace)	Cursive
`cider-macroexpand-1`	`C-c C-m`	`c1m{motion}`	With a REPL open, select
`cider-macroexpand-all`	`C-c M-m`	`cm{motion}`	*Tools -> REPL -> View expansion*