Clojure Macro Sigils Reference https://purelyfunctional.tv/clojure-macros

`

Name: backquote AKA backtick
Purpose: evaluate code at runtime
Example:

```
(defmacro when* [test & body]
  `(if ~test
       (do ~@body)
       nil))
```

evaluate if at runtime

~

Name: unquote
Restrictions: must be inside of a backquote
Purpose: evaluate code at compile time
Example:

```
(defmacro when* [test & body]
  `(if ~test
       (do ~@body)
       nil))
```

evaluate if at runtime

evaluate test at compile time

since test is a symbol, that means it evaluates to the expression used when calling the macro

if we don't eval it at compile time, the symbol itself will be put there

~@

Name: unquote splice
Restrictions: must be inside of a backquote
Purpose: evaluate code at compile time and splice result into containing list
Example:

```
(defmacro when* [test & body]
  `(if ~test
       (do ~@body)
       nil))
```

evaluate if at runtime

evaluate test at compile time

evaluate body at compile time and splice into the resulting seq

since body is a symbol, it evaluates to the expression used when calling the macro

body is a list of rest arguments - we want to splice it into the do

~'

Name: unquote quote
Restrictions: must be inside of a backquote
Purpose: insert symbol without namespace expansion
Example:

```
(defmacro ana-if [test then else]
  `(if-let [~'it ~test]
     ~then ~else))
```

evaluate if-let at runtime

insert the symbol it directly

un-sigilized symbols are expanded to include the full namespace, which we don't want here

#

Name: auto-gensym
Restrictions: must be inside of a backquote; suffix (at the end of a symbol)
Purpose: create a unique variable name
Example:

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

x# expands to a new unique symbol each time the macro is called

useful for making local variables

but inside of a single macro call, they all expand to the same symbol