Macros: Summary

Macros: Summary

- Macros take code as arguments and return new code.

- The compiler calls a macro function when it sees a macro invocation, and completely replaces that macro invocation with the code returned by the macro function.

- That process is repeated until there is no macro invocation anymore.

Macros: Summary

- Use macros for syntactic abstractions, to hide implementation details that cannot be hidden otherwise.

- Note that macros cannot be used as first-class values at runtime, unlike functions, at least not in a straightforward way.

Macros: Summary

- Macros support destructuring to get at the different elements of a macro invocation.

- Backquote supports constructing new code from those destructured elements.

- Backquote constructs lists without evaluating most positions. It makes writing macros nicer, but is independent from writing macros.

How to Write Macros

- Decide if the macro is necessary.
- Write down the syntax of the macro, using one or more examples.
- Figure out the expansion.
- Use defmacro to implement the expansion.
- Test the macro.

First Macro Pitfalls

- Macros can expand into macro invocations. Make sure that macro expansion terminates!

- Make sure that phases are always separated. Macros are evaluated at macro-expansion time and generally cannot see runtime values.

- Macro-expansion time is typically compile time.