# DRY (Don't Repeat Yourself)

In The Pragmatic Programmer, Hunt and Thomas define the DRY principle as follows:

*Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.*

In other words: don't repeat yourself!

The most common way to avoid repetition is to break duplicated code into methods.

def process\_consumer\_address\_form(fields)

unless fields[:zip] =~ /[1-9][0-9]{4}/

raise InvalidZipCodeError

end

# more form processing

end

def process\_business\_address\_form(fields)

unless fields[:zip] =~ /[1-9][0-9]{4}/

raise InvalidZipCodeError

end

# more form processing

end

See how we've repeated the bit that checks that the zip code is valid? This is bad; if there were a bug with our zip code validation, we'd have to fix the code **in two places**. This is a pain, and we'll probably forget to fix the duplicated code everywhere, so it's also a source of bugs. For this reason, we want to find a way to eliminate the duplicated code:

def process\_consumer\_address\_form(fields)

raise InvalidZipCodeError unless valid\_zip?(fields[:zip])

# more form processing

end

def process\_business\_address\_form(fields)

raise InvalidZipCodeError unless valid\_zip?(fields[:zip])

# more form processing

end

def valid\_zip?(zip)

zip =~ /[1-9][0-9]{4}/

end

Breaking the duplicated code into a method has the side-effect of making the consumer/business methods shorter and more focused on the details of checking the form, rather than the details of zip code validation.

A good rule of thumb with DRY is that if you find yourself copying and pasting code into other places, you should most likely refactor to avoid duplication.