# Testing Small

(Note: This reading assumes familiarity with the Memory Puzzle project.)

Make a sample card to play around with. Open up pry and make a new card (something like card = Card.new). In your code, play around with how you assign variables.

For example:

def hide

face\_up = false

end

Reload your code. In pry, call card.hide and see what happens. Does it change the way you expected?

def hide

self.face\_up = false

end

Reload your code. In pry, call card.hide and see what happens. Does it change the way you expected?

def hide

@face\_up = false

end

Reload your code. In pry, call card.hide and see what happens. Does it change the way you expected?

What's the difference between the different ways of writing the method? You might have to google for the subtle differences.

The overall goal here is to **test your code out with simple examples**.

If you can test one card on its own, it's a lot easier than trying to test the card through other code. In this case, you shouldn't test card.hide by calling a different method that calls card.hide inside.

For example, it would be hard to test card.hide by calling the method below. There's too much stuff to wade through:

class Board

#...

def update\_cards

puts "updating the cards"

self.repopulate

# a bunch of other methods

@cards.each do |card|

@grid << card

card.flip

# a bunch of other methods

card.hide #the actual line we want to test

end

end

end

board = Board.new

board.update\_cards

board.render

There's way too much going on for your brain to easily tell when card.hide **actually** gets called. And it's hard to tell if anything else is changing face\_up on the card. Instead, try to test one card on its own.

Benefits: You saved at least five minutes by testing small instead of hacking through a huge block of buggy code. You have to test your code many times a day (almost every time you make a change). Imagine that savings multiplied over and over. You're saving **hours** of your day by testing small.