Introduction

Now we're going to use the virtual deck of cards that we created in the last video to recreate a web version of one of my favourite childhood game. I used to love watching Play Your Cards Right. Some of you may not have seen or heard of this show, so I've provided a clip in the resources section below. The basic idea, though, was that contestants had to guess whether a randomly chosen card would be higher or lower than the previous card.

Coding Our Program

To get started, you'll need a file called *play_your_cards_right.rb*:

```
require 'sinatra'
enable :sessions

get '/' do
 session[:deck] = []
  suits = %w[ Hearts Diamonds Clubs Spades ]
  values = %w[ Ace 2 3 4 5 6 7 8 9 10 Jack Queen King ]
  suits.each do |suit|
    values.each do |value|
      session[:deck] << "#{value} of #{suit}"
    end
  end
  session[:deck].shuffle!
  session[:guesses] = -1
  redirect to('/play/cards')
end

get '/play/:guess' do
  card = session[:deck].pop

  value = case card[0]
    when "J" then 11
    when "Q" then 12
    when "K" then 13
    else card.to_i
  end

  if (params[:guess] == 'higher' and value < session[:value]) or (params[:guess] == 'lower' and va
    "Game Over! The card was the #{ card }. You managed to make #{session[:guesses]} correct gues
  else
    session[:guesses]  += 1
    session[:value] = value

    "The card is the #{ card }. Do you think the next card will be <a href='/play/higher'>Higher</
  end
end
```

This app only uses two route handlers, but they're quite complicated, so we'll take a look at each one in turn.

The first route handler uses the root (/) URL. This route basically sets up the game by creating a deck of cards like we did earlier, but this time we're going to store the deck in the `session` hash. For that reason we add

```
enable :sessions
```

at the start of the file.

Going back to the route handler, we first of all set up a session hash with a name of `:deck`, and we assign this a empty array. Next iterate over the list of suits and values and then push a string representing each card into the deck in the session.

Once we've created the deck of cards and stored it in the session, we then use the `shuffle` method with a bang on the end to shuffle all the cards.

Next, we create another session variable with the key of `guesses` to store the number of guesses that the player has made. Now, this is set to `-1` initially, which seems a bit odd at first, but I'll explain more about why we do this later on.

Sinatra's Helper Methods

Then we use Sinatra's helper methods, `redirect` and `to`, to redirect to the URL `/play/cards`. We don't actually have a route handler for this route, we just have a generic route handler that uses a named parameter called play and then get. So it doesn't say `/play/cards`, but because it's a generic handler using the name parameter, this route that we just redirected to will be covered by the route handler.

First of all, we create a variable called `card`, and use the pop method to take a card from the session that's stored in the deck variable - this will store a string representing a card inside the variable called `card`.

Pop Method

Now, we need to get the value of this card, since the game is about guessing if a card is higher or lower. To do this, we use this `case` statement. This will eventually store the value of the card in the variable called `value`. To find out the value of the card, we look at the first letter of the string that's stored in the card variable:

```
card[0]
```

This is similar to how we access each item in an array where the counting starts at 0. So this represents the first character in a string. What the `case` statement does is look and see if the first character is a "J", then the card must be a jack, so we give it a value of `11`, and so on.

For all the other cards the first character in the string will actually be its value. We know that the `to_i` method will convert a string into an integer equal to the first numerical value in that string. So in all these cases, for all the other cards, their actual value is equal to the first integer in the string. So we can just use this method to grab their value. And as an added bonus, this will also work for aces, because if a string doesn't contain any numbers at all, then this method will convert into 0, and even though we'd normally expect aces to have a value of 1, this doesn't actually matter because we're only testing if the cards are higher or lower. As long as the ace has a lower value than all the other cards, then the game will still work.

So, now we have a variable for the card, which is a string representation of the card, it might be the king of spades or four of diamonds, and we also have a variable that stores the value of the card.

If Statement

We then enter a big `if` statement. It checks to see if the player has made an incorrect guess or a correct guess. It starts by checking what the value of the `params[:guess]` is, which usually, if the player has made a guess, will be set to higher or lower, but remember, this is the first time we've hit this route, and it's right at the start of the game.

Initially, the value of the `params` hash will be nothing. This will mean that this won't get evaluated at all - the `number` of guesses will be simple bumped by one. If you remember earlier, we set the number guesses to `-1` - this is because when we go through the route handler the first time, the player hasn't actually made a guess.The game has just started, and you can see here that this will bump it up to 0.

Session Variable

Then we output the block of HTML that will display what the card is. It has links to say, do you want to choose higher or lower? The links actually link back to the same route handler that starts with `play`, but the named parameter changes. This will be stored in the `params[:guess]` hash, and it will either be `higher` or `lower`. So, when the player clicks on one of these links, they'll be taken back to the start of this same route handler, and the whole sequence will repeat again.

A new card will be picked from the session deck, and stored in the variable card. We'll then grab the value of this card, and this time the `if` condition becomes important, because `params[:guess]` will either be higher or lower, depending on which link the player clicked on.

Params

The first condition is saying the player picks higher, that is stored in the `params[:guess]` variable, but, the value of the current card is actually lower than the value of the previous card, which is stored in the session - that's incorrect. Another condition is also incorrect: the player guessed lower, but the value of the current card is actually bigger than the previous card. So if either of those conditions are met, the player has guessed incorrectly. In that case, the message will be displayed and the game will end.

Using Interpolation

We'll use the `session` variable for the number of guesses to display how many guesses were made using interpolation. We'll also put a link for the player to play again, and this will send them back to the root URL.

If the player guesses correctly, then we hit the `else` part of the `if` statement. This actually plays through like it does the first time through. The first time these conditions aren't met either purely because `params[:guess]` does not exist. This increases the number of correct guesses and then sets the value stored in the session to the current value, because this card is about to become the previous card, as the next card gets picked. Then we output the message again, giving the player the option to choose higher or lower.

This process will repeat over and over again, using the same route handler, until the player makes an incorrect guess, and they get the Game Over message. Once the game is played again, we go back to the route URL.