COMP 9 Lab 3: Blackjack revisited

Out: Thursday, February 10th, 1:15 PM

Due: Thursday, February 17th, 12:00 PM

1 Overview

In the previous assignment, you wrote a Blackjack game that had some significant limitations. The most significant of these was that we assumed an infinite deck; you could potentially draw 21 aces (which you might choose to score as 1) in a row, or you could draw five fours. The reason for this simplification was that you didn't yet have a way to easily model a real deck of cards. Now you do.

2 Some new topics in code

2.1 Appending to an array

As discussed in class, an array can be created with the `[]` syntax; an empty array is simply `[]`, while a non-empty array can be created (much like a literal string) such as `[1,2,3]` or `[[1,'hearts'],['jack', 'spades']]`

What we did not discuss in class is that you can *append* to an existing array. The simplest way to do this is with the `<<` method:

```
scores = [94, 23, 76]
scores << 89
# now, scores == [94, 23, 76, 89]
```

Similarly, one could add any other kind of Ruby *object* to an array, including another array:

```
my_poker_hand = [['jack', 'spades'], [10, 'hearts']]
my_poker_hand << [2, 'clubs']
# now, my_poker_hand == [['jack', 'spades'], [10, 'hearts'], [2, 'clubs']]
```

Or, if one preferred:

```
my_poker_hand = []
my_poker_hand << {:name => 'Two', :suit => 'Clubs', :value => 2}
my_poker_hand << {:name => 'Jack', :suit => 'Spades', :value => 10}
# now, my_poker_hand ==
    [{:name => 'Two', :suit => 'Clubs', :value => 2},
    {:name => 'Jack', :suit => 'Spades', :value => 10}]
```

2.2 Big, huge hint

Think about how you could populate an initially empty array *using a loop*. Consider the following example:

```
animals = []
input = ''
while input != 'bye'
    input = gets.chomp
    animals << input
end
```

Now, what if we want to *combine* two kinds of thing in a *combinatorial expansion* (also called a *cartesian product*), say, times of day and days of the week?

```
times = [11, 12, 1]
days = ['Monday', 'Tuesday', 'Wednesday']
daytimes = [] # initially empty array
days.each do |d|
    times.each do |t|
        # insert into daytimes a new hash based on d and t
        daytimes << {:day => d, :time => t}
    end
end
```

Now daytimes is the following array of hashes:

```
[{:day=>"Monday", :time=>11}, {:day=>"Monday", :time=>12},
{:day=>"Monday", :time=>1}, {:day=>"Tuesday", :time=>11},
{:day=>"Tuesday", :time=>12}, {:day=>"Tuesday", :time=>1},
{:day=>"Wednesday", :time=>11}, {:day=>"Wednesday", :time=>12},
{:day=>"Wednesday", :time=>1}]
```

2.3 Removing an element from an array

There are two ways to remove an element from an array and *keep* that element around in a variable.

First, there is the `pop` method. It always returns the *last* element in the array:

```
a = [1,2,3]
b = a.pop
# now b contains 3
# and a contains [1,2]
c = a.pop
# now c contains 2
# and a contains [1]

students = [{:name => 'Bob', :gpa => 3.76}, {:name => 'Molly', :gpa => 3.84}]
somebody = students.pop
# now students is just [{:name => 'Bob', :gpa => 3.76}]
# and somebody is the hash {:name => 'Molly', :gpa => 3.84}
```

Secondly, there is the `delete_at` method. It takes an *argument* which is the *index* of the element to remove from the array. Remember that array indexing starts at zero:

```
a = [1,2,3]
b = a.delete_at(1)
# now b contains 2
# and a contains [1,3]
```

2.4 Reordering an array

There are also multiple ways to randomly change the order of an array. First, there is the `sort_by` method, which can take a `block` that calls `rand`:

```
a = [1,2,3,4,5]
a = a.sort_by{rand}
# note that sort_by returns a new array, so we have to assign it back to a
# now a might look like [2, 3, 5, 4, 1] or [3, 1, 4, 5, 2]
```

There is also the `shuffle` method, which does the same thing:

```
a = [1,2,3,4,5]
a = a.shuffle
# note that shuffle returns a new array, so we have to assign it back to a
# now a might look like [2, 3, 5, 4, 1] or [3, 1, 4, 5, 2]
```

Finally, there is `shuffle!` Methods whose name ends in an exclamation point are *destructive* in ruby; they *change* the receiver:

```
a = [1,2,3,4,5]
a.shuffle! # note, we do not have to assign it to a variable
# now a might look like [2, 3, 5, 4, 1] or [3, 1, 4, 5, 2]
```

3 First, some housekeeping

Use `mkdir` to create a new directory within your `comp9` directory, called `lab3`.

Within Kate, you can create the file `blackjack2.rb`

4 Design sketch

Please write out an *outline* of your program. You may do it on paper; either use *pseudocode* (just English describing your program in detail) or diagrams of any sort. We ask you to check with a member of course staff (Noah, Sarah, or Joel) to approve your design before you begin coding. We will also ask you to submit this *design* along with your final lab submission. If you'd like us to photocopy the design, let us know; you may also just submit it with two names on the top, in hardcopy. Explain on paper:

1. where and how you will ask for input
2. where and how you will provide output
3. what loops and conditionals you will need
4. how you will represent your cards
5. how you will populate and shuffle the deck
6. how you will ensure that you randomly draw a card
7. how you will be sure you are upholding the rules of the Blackjack game.
8. your answer to this question: there are two fundamentally different (but equally valid) ways to represent a shuffled deck that you draw cards from. Can you think about what they might be? Be sure to discuss this in groups.

I encourage you to work on this first step with someone sitting near you. Don't share code, but discuss how you might approach the program.

You are going to write an improved blackjack game. The main simplification in the last assignment was that we assumed an infinite deck, and simply drew random numbers. This time, we're going to have a representation of cards that have both a *suit* and a *value*, and a representation of a deck that allows you to draw cards. If you draw an ace of spades from the deck, it's gone from the deck. This means that if you've drawn four aces, there is no chance of drawing a fifth.

Here's a refresher as to the rules of Blackjack:

1. The player is dealt two cards.
2. The player's object is to get as high a score as possible, without exceeding 21 ("busting").
3. Numbered cards count as their natural value
4. Jack, Queen, and King ("face cards") count as 10
5. Aces are valued at 1 or 11 according to the player's preference.
6. If the hand value exceeds 21 points, it busts.

As with the previous assignment, the game should present the player with a two-card hand initially, and then a loop that deals the player additional cards as long as they keep *hitting*. In fact, the loop structure of this program should be very similar to the last one. Remember, though, that you must populate a deck of cards before dealing the first hand.

As a reminder, sketch out your approach to this problem with a neighbor, and please ask for help if you're stuck. Once you feel your approach is reasonable, ask a member of the course staff (Noah, Sarah, or Joel) to look at your design. Once they approve it, you may begin coding. Your code must be your own; you should not collaborate with your neighbor beyond this point.

5 Code

Just to recap, write a Ruby program, `blackjack2.rb`, that implements the blackjack game described above.

6 Extra Credit

For extra credit, write an even more sophisticated program, `blackjack3.rb`, that also remembers the entire hand the player has been dealt (not just the

total score) and, when they complete the game (whether they win or bust or just stick with a score less than 21), tell them what their entire hand looks like.

For example:

```
You won! Your hand consists of:
2 of Spades
Ace of Diamonds
8 of Hearts
```

Make sure that if you do the extra credit, it is a separate program called `blackjack3.rb`; you must also submit `blackjack2.rb`

7 Handing in your solution

When you are satisfied with your program, submit it to be graded with the following command:

```
provide comp9 lab3 blackjack2.rb
```

Or, if you did the extra credit:

```
provide comp9 lab3 blackjack2.rb blackjack3.rb
```

You must also hand in your design sketch; please submit it (by hand) with the names of all involved students at the top. If you would like a photocopy for your reference, let us know.