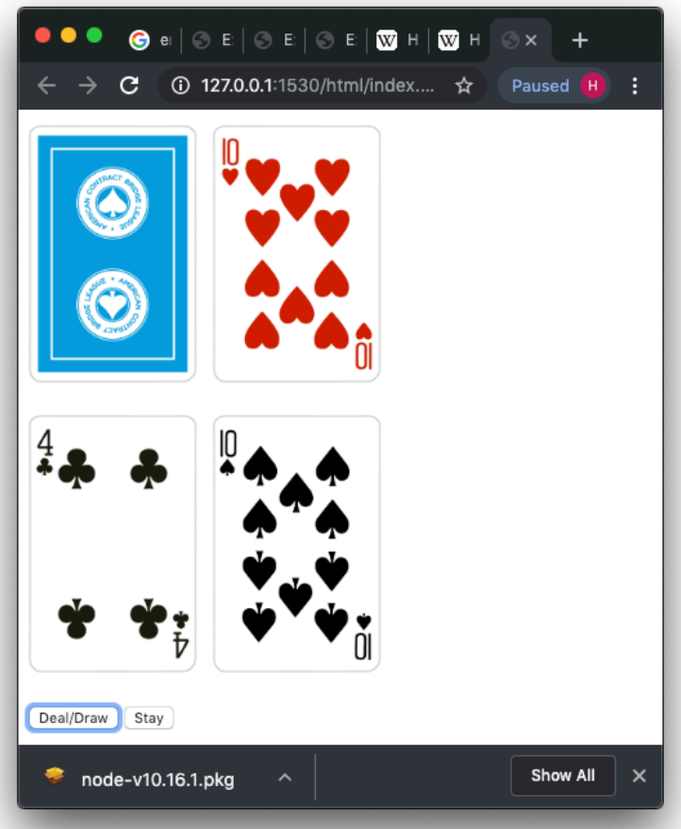
Rev. 08Aug2019

Exercise 48

BUILDING A BLACKJACK GAME

In this exercise, we'll look at building the game of Blackjack (*aka* 21). As with the last exercise, you won't be writing any code, but rather following along with the code I wrote.

The project will look like this:



As with tic-tac-toe, there's very little styling to the page.

A SIMPLIFIED SET OF RULES

Blackjack, as it's played in a casino, is more complex than what I want to model, so I've left off things like doubling down, splitting, insurance, and surrender.

INITIAL LIST OF STEPS

Of course, we start by listing the various things the game needs to account for:

- create a deck of cards
- shuffle the cards
- deal to player
- deal to dealer
- allow player to draw or stay
- play another hand
- display the cards dealt to player
- display the cards dealt to the dealer
- display the winner
- evaluate the winner

AND THE IMPLEMENTATION

Were we working for a company, we would commit this code so that we (and others) could easily use it in the future.

```
// create an unshuffled deck of cards
let suits = ['C','D','H','S']
let pips = [2,3,4,5,6,7,8,9,10,11,12,13,14]
let unshuffledDeck = () => {
  let arr = []
  suits.forEach( suit => {
    pips.forEach( pip => {
      arr.push(pip + suit)
    })
  })
  return arr
}
```

This should look familiar.

```
// create four shuffled decks
let shuffledDeck1 = shuffle(unshuffledDeck())
let shuffledDeck2 = shuffle(unshuffledDeck())
let shuffledDeck3 = shuffle(unshuffledDeck())
let shuffledDeck4 = shuffle(unshuffledDeck())
```

It occurred to me *while I was coding* that I should use multiple decks. Was that a failure of planning on my part? Yes.

```
// combine the four decks
let bigDeck =
shuffledDeck1.concat(shuffledDeck2).concat(shuffledDeck3).concat(shuffledDeck4)
```

Just one way of combining them. There are others that might be used.

```
// shuffle the bigDeck
let deck = shuffle(bigDeck)
```

Shuffle them all together.

```
// write a "dealCard" function
let dealCard = deck => {
  let dealtCard = deck.shift()
  return dealtCard
}
```

Also familiar from the last exercise.

```
// create object to store cards
let currentHand = {
  player: [],
  dealer: []
}
```

We did something similar in the last exercise, but this time, we need to store an array of cards for each player.

```
// after initialDeal, does either one have a Blackjack?
let isBlackjack = handValue => {
  return handValue === 21
}
```

If either (or both) player has an initial value of 21, the game stops, so we need to check for that.

```

```

```
// create a function to evaluate the value of a hand
let valueHand = hand => {
  let handValue = 0
  hand.forEach(card => {
    let regex = /[0-9]*/
    let cardValue = parseInt(regex.exec(card))
    if (cardValue > 10 && cardValue < 14) { cardValue = 10 }
    if (cardValue === 14) { cardValue = 11 }
    handValue += cardValue
  })
  return handValue
}
```

Just one way of implementing this necessary step.

```
// create a variable, "inHand", as a boolean defaulting to false
let inHand = false;
```

We'll need to see whether we're already in a hand for subsequent functions.

```
// create an initialDeal function that will give each player 2 cards, but one of the
dealer's cards should be displayed face-down
let initialDeal = () => {
```

I would display all the code for this, but it would take up multiple pages. And that should have been a *big* clue to me that something is wrong with my code. Functions, we've noted before, should handle one discrete task. This one does far too much. "But", my less-experienced self might argue, "it *needs* to do a lot, so the length is justified." A conundrum? Not really. How would you handle this problem?

Try to come up with an answer before proceeding to the next page.

The solution is to break that one, large function into multiple, smaller functions. *Then* the original function can call those smaller functions to perform the needed tasks.

I won't go into the rest of the code, as I hope you're getting the point of these later exercises: not so much to teach you new syntax, but to encourage you to *form a plan* for your code.