Meta

JavaScript!

It's everywhere!

- the browser
- node.js
- mobile devices

Everyone works with it!

- client-side developers
- server-side developers
- embedded and systems engineers

These Videos!

Focus on JavaScript as a language

- no node.js
- no mobile devices
- no browsers (well, almost)

Plenty of resources focus on these other things, so we will

- learn starting from first principles.
- study variables, functions, types, conditionals, loops, arrays, objects
- avoid graphics or special libraries -- JUST JAVASCRIPT!

These Videos!

Along the way we'll...

- solve problems!
- play with open data!
- play with twitter data!
- build a poker simulation!

Me!

Some guy who knows JavaScript and...

- used to be a teacher
- is now a software engineer
- co-organizes a “Learn JavaScript” meetup in San Jose, CA

You!

I don’t know you, but...

- you want to learn JavaScript.
- you’re probably relatively new to coding.
- you’re excited, motivated and ready to start learning and memorizing!

Help?!?

I ran into problems! What do I do?

- Solutions are on Github!
- Use the “Issues” feature on Github.
- Tweet at me: *@semmypurewal*
- Email me: *me@semmy.me*

Materials

You'll need the following

- Chrome web browser (http://www.google.com/chrome)
- A Text Editor (http://www.sublimetext.com)
- The examples from Github

Variables

Overview

Important Concepts

- values
- expressions
- variables

At the end of this section, you will be able to

- differentiate between values, expressions, and variables.
- declare a variable using the `var` keyword.
- store a value in a variable using the assignment operator.
- reassign a new value to a variable that already stores a value.

Summary

- *values* are concrete program entities, e.g. `5` or `"hello world"`
- *expressions* create new values from existing values
- *variables* store values so they can be used later in the program
- we *declare* variables with the `var` keyword
- we *define* a variable with the assignment operator (`=`)
- the right-hand-side of an assignment can be a value or an expression
- we can re-assign new values to variables after they are created

Practice

For this section, the practice problems are all at the end of the `readme.md` file in the `01-variables` directory.

Functions

Overview

Important Concepts

- functions
- function inputs
- local variables and function scope

At the end of this section, you will be able to

- package repeated behavior into a function.
- identify the major parts of a function (the inputs, the body, & the output).
- describe function scope.

```
var addThree = function (a, b, c) {
    var sum = a + b + c;
    return sum;
}

addThree(5, 10, 15);
//=> 30
```

```
var addThree = function (a, b, c) {
    var sum = a + b + c;
    return sum;
}

addThree(5, 10, 15);
```

```
var addThree = function (a, b, c) {
    var sum = a + b + c;
    return sum;
}

addThree(5, 10, 15);
//=> a = 5; b = 10; c = 15;
```

```
var addThree = function (a, b, c) {
    var sum = a + b + c;
    return sum;
}

addThree(5, 10, 15);
//=> a = 5; b = 10; c = 15;
//=> evaluate the expression a + b + c
```

```
var addThree = function (a, b, c) {
    var sum = a + b + c;  // 30
    return sum;
}

addThree(5, 10, 15);
//=> a = 5; b = 10; c = 15;
//=> evaluate the expression a + b + c
//=> store the result in sum, a local variable
```

```
var addThree = function (a, b, c) {
    var sum = a + b + c;
    return sum;
}

addThree(5, 10, 15);
//=> a = 5; b = 10; c = 15;
//=> evaluate the expression a + b + c
//=> store the result in sum, a local variable
//=> 30
```

Summary

- *functions* are structures used to package repeated code
- functions have zero or more *inputs*, one *output*, and a *body*
- the inputs to a function are also known as *parameters* or *arguments*
- the output is specified by the `return` statement
- a function is a value and, like other values, can be stored in a variable
- *local variables* are variables defined inside a function
- local variables do not exist outside of their associated function

Practice

There are two questions at the end of the `readme.md` file in the `02-functions` directory.

For the rest, we'll edit the `practice.js` file and use the `runner.html` file to test whether our answers are correct. You'll complete the following functions.

- `add (a, b)`
- `totalCost (numItems, costPerItem)`
- `cardString (rank, suit)`
- `openTag (tagName)`
- `closeTag (tagName)`
- `toTagString(tagName, content)`

Types

Overview

Important Concepts

- types (numbers, strings and booleans)
- expressions using operators relevant to specific types
- functions that verify types and extended types

At the end of this section, you will be able to

- explain why types are an essential concept.
- use the `typeof` operator to check a value's type.
- write expressions that operate on numbers, strings, and booleans.
- create functions that verify values have certain properties.

Arithmetic Operators

Operator	Operation
`a * b`	Multiplication, `a` times `b`
`a / b`	Division, `a` divided by `b`
`a % b`	Remainder when `a` is divided by `b`
`a + b`	Addition, `a` plus `b`
`a - b`	Subtraction, `a` minus `b`

Math Functions

Operator	Operation
`Math.pow(a, b)`	calculate a power a^b
`Math.round(a)`	round argument to the integer nearest `a`
`Math.floor(a)`	smallest integer above `a`
`Math.ceil(a)`	largest integer below `a`
`Math.max(a,b,c,...)`	largest of all arguments `a,b,c,...`
`Math.min(a,b,c,...)`	smallest of all arguments `a,b,c,...`
`Math.random()`	random number between `0` and `1`

String Methods

Operator	Operation
`str.toLowerCase()`	return `str` with all upper-case letters
`str.toUpperCase()`	return `str` with all lower lower-case letters
`str.indexOf(a)`	first starting index of substring `a` or `-1` if `a` is not found
`str.slice(a, b)`	get a substring starting at index `a` ending at `b-1`
`str.charAt(index)`	return the character at `index`
`str.length`	return length of the string (**a property, not a method**)

Boolean Operators

Operator	Operation
`a < b`	returns `true` if `a` is less-than `b`, `false` otherwise
`a <= b`	returns `true` if `a` is less-than or equal to `b`, `false` otherwise
`a > b`	returns `true` if `a` is greater-than `b`, `false` otherwise
`a >= b`	returns `true` if `a` is greater-than or equal to `b`, `false` otherwise
`a === b`	returns `true` if `a` is equal to `b`, `false` otherwise
`a !== b`	returns `true` if `a` is not-equal to `b`, `false` otherwise

Logical Operators

Operator	Operation
`a || b`	logical *or*; `true` if either `a` or `b` is `true`
`a && b`	logical *and*; `true` if `a` and `b` are both `true`
`!a`	logical *not*; `true` if `a` is `false`, `false` if `a` is true

Summary

- a variable can store a value of any type
- we can determine the type of a value by using the `typeof` operator
- the value types we'll work with are strings, numbers, and booleans
- values allow for different operations depending on their type
- operations on these types are accessed inconsistently:
    - numbers use built-in operators and functions in the `Math` object
    - strings (mostly) use methods via the dot operator
    - booleans use built-in operators
- we can extend the types available to us by writing boolean functions

Practice

There are two questions at the end of the `readme.md` file in the `03-types` directory. In `practice.js`, complete the following functions:

- `isDivisibleBy3(number)`
- `celsToFahr(celsiusTemp)`
- `fahrToCels(fahrTemp)`
- `randUpTo(maxNumber)`
- `isSuit(potentialSuit)`
- `isRank(potentialRank)`
- `isCapitalized(str)`
- `getHTMLText(htmlElement)`
- `isHTMLElement(potentialHTMLElement)`

Conditionals

Overview

Important Concepts

- `if`-statements and `if`-statements with `else`-clauses
- the `if-else-if` and nested `if`-statement patterns
- the `throw` operator

At the end of this section, you will be able to

- use `if`-statements to change program behavior based on values.
- write code that verifies complex properties of values.
- write functions that only operate on certain types.

```
var age = 25;

if (age >= 13) {
    console.log(“You can have a Facebook account!”);
}
console.log(“finished!”);
```

```
var age = 25;

if (age >= 13) {
    console.log(“You can have a Facebook account!”);
}
console.log(“finished!”);
```

```
var age = 25;

if (age >= 13) {  // is 25 >= 13?
    console.log("You can have a Facebook account!");
}
console.log("finished!");
```

```
var age = 25;

if (age >= 13) {  // YES!
    console.log("You can have a Facebook account!");
}
console.log("finished!");

//=> You can have a Facebook account!
```

```
var age = 25;

if (age >= 13) {
    console.log("You can have a Facebook account!");
}
console.log("finished!");

//=> You can have a Facebook account!
//=> finished!
```

```
var age = 11;

if (age >= 13) {
    console.log(“You can have a Facebook account!”);
}
console.log(“finished!”);
```

```
var age = 25;

if (age >= 13) {  // is 11 >= 13?
    console.log("You can have a Facebook account!");
}
console.log("finished!");
```

```
var age = 25;

if (age >= 13) {  // NO!
    console.log("You can have a Facebook account!");
}
console.log("finished!");
```

```
var age = 25;

if (age >= 13) {
    console.log("You can have a Facebook account!");
}
console.log("finished!");

//=> finished!
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) {
    console.log(“You can ride Space Mountain!”);
} else {
    console.log(“Sorry! You can’t ride this year.”);
}
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) {
    console.log("You can ride Space Mountain!");
} else {
    console.log("Sorry! You can't ride this year.");
}
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) {
    console.log("You can ride Space Mountain!");
} else {
    console.log("Sorry! You can't ride this year.");
}
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) {  // 3.75
    console.log("You can ride Space Mountain!");
} else {
    console.log("Sorry! You can't ride this year.");
}
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) {  // 3.75 >= 4 ?
    console.log("You can ride Space Mountain!");
} else {
    console.log("Sorry! You can't ride this year.");
}
```

```
var height = 45; // inches
var minHeightInFeet = 4;

if (height/12 >= minHeightInFeet) { // NO!
    console.log("You can ride Space Mountain!");
} else {
    console.log("Sorry! You can't ride this year.");
}
//=> "Sorry! You can't ride this year.
```

Summary

- `if`-statements change the behavior of programs based on boolean values
- `if`-statements execute their code-block if their condition evaluates to true
- `else`-clauses execute their code-block if their condition evaluates to false
- We can use `throw` to ensure functions only operate on expected types

Practice

In `practice.js`, complete the following functions:

- `passwordStrength(password)`
- `isLeapYear(year)`
- `firstInDictionary(word1, word2, word3)`
- `randUpTo(maxNumber)`
- `getTagName(htmlElement)`
- `improveTweet(tweet)`
- `isQuestion(str)`
- `magic8Ball(question)`
- `interjectAt(interjection, index, tweet)`
- `randomInterject(tweet)`

Loops

Overview

Important Concepts

- `while`-loops
- `for`-loops
- the structure of `for`-loops

At the end of this section, you will be able to

- identify the four parts of a `for`-loop.
- write `for`-loops that iterate forward and backward over a set of numbers.
- write `for`-loops that iterate over all the characters in a string.
- calculate properties of sets of values using a `for`-loop.

A `for`-loop consists of four things

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log(“num is currently “ + num);
}
```

A `for`-loop consists of four things

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}
```

the initialization statement, executed once before the loop

A `for`-loop consists of four things

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}
```

the continuation condition, checked each time before executing the body

A `for`-loop consists of four things

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}
```

the update statement, executed each time the loop body ends

A `for`-loop consists of four things

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}
```

the loop body, executed each time the continuation condition is true

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is declared
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is set to 0 (initialization)
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 0, is num <= 5 true? (yes)
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 1
num is currently 0
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 1, is num <= 5 true? (yes)
num is currently 0
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
num is currently 1
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 2
num is currently 0
num is currently 1
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 2, is num <= 5 true? (yes)
num is currently 0
num is currently 1
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
num is currently 1
num is currently 2
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 3
num is currently 0
num is currently 1
num is currently 2
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 3, is num <= 5 true? (yes)
num is currently 0
num is currently 1
num is currently 2
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
num is currently 1
num is currently 2
num is currently 3
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 4
num is currently 0
num is currently 1
num is currently 2
num is currently 3
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 4, is num <= 5 true? (yes)
num is currently 0
num is currently 1
num is currently 2
num is currently 3
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 5
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 5, is num <= 5 true? (yes)
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> loop body executed
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
num is currently 5
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> update statement executed, num is now 6
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
num is currently 5
```

```
var num;

for (num = 0; num <= 5; num = num + 1) {
    console.log("num is currently " + num);
}

//=> num is 6, is num <= 5 true? (NO!)
num is currently 0
num is currently 1
num is currently 2
num is currently 3
num is currently 4
num is currently 5
```

Summary

- *loops* allow are programs to repeatedly execute collections of statements
- `while`-loops are similar to `if`-statements, but they execute the body until the condition becomes false
- `for`-loops are while loops with built-in bookkeeping, and we'll use them more frequently

Practice

In `practice.js` in the `05-loops` directory complete the following functions:

- `isVowel(letter)`
- `isLowerCaseLetter(letter)`
- `sumUpTo(number)`
- `sumAToB(a, b)`
- `countVowels(str)`
- `reverseString(str)`
- `isPrime(number)`
- `sumPrimesUpTo(number)`
- `sumOfFirstNPrimes(n)`
- `removeNonLetters(str)`
- `isPalindrome(str)`

Arrays

Overview

Important Concepts

- arrays and indices
- the similarities and differences between strings and arrays
- the `Array.isArray` function

At the end of this section, you will be able to

- define an array value.
- iterate over an array using a `for`-loop.
- use basic array methods like `slice`, `indexOf`, and `push`.
- determine whether a value is an array with `Array.isArray`.

Summary

- Arrays allow us to associate multiple values with a single entity
- Arrays are similar to strings and share some of the same methods
- We access array elements with the square bracket operator and an *index*
- We use the `push` method to add values to the end of an array
- `typeof` doesn't work as expected arrays, use `Array.isArray` instead

Practice

There are two questions in `readme.md`. Also complete `practice.js`:

- `containsTwice(value, arr)`
- `containsNTimes(value, arr)`
- `atLeastOneEven(arr)`
- `allStrings(arr)`
- `containsAnyTwice(values, arr)`
- `getValuesAppearingTwice(arr)`
- `range(a,b)`
- `mapToTags(htmlElements)`
- `filterToLol(tweets)`

Array Methods

Overview

Important Concepts

- `forEach`, `map`, `filter`, `some`, `every`, `reduce`
- chaining array methods
- converting between strings and arrays

At the end of this section, you will be able to

- use array methods to avoid `for`-loops where it makes sense.
- construct complex computations by chaining array methods.
- use split and join to convert between arrays and strings.

```
var square = function (number) {
    return number * number;
}

var numbers = [ 2, 4, 6, 8 ];
numbers.map(square);
```

```
var square = function (number) {
    return number * number;
}

var numbers = [ 2, 4, 6, 8 ];
numbers.map(square);
//=> [ square(2), square(4), square(6), square(8) ]
```

```
var square = function (number) {
    return number * number;
}

var numbers = [ 2, 4, 6, 8 ];
numbers.map(square);
//=> [ 4, 16, 36, 64 ]
```

```
var square = function (number) {
    return number * number;
}

var numbers = [ 2, 4, 6, 8 ];
numbers.map(square);
//=> [ 4, 16, 36, 64 ]
```

```
var square = function (number) {
    return number * number;
}

var numbers = [ 2, 4, 6, 8 ];
numbers.map(function (number) {
    return number * number;
});
//=> [ 4, 16, 36, 64 ]
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> [ 126, 199 ]
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
//=> number: 5, returns false, result: []
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
//=> number: 5, returns false, result: []
//=> number: 126, returns true, result: [ 126 ]
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
//=> number: 5, returns false, result: []
//=> number: 126, returns true, result: [ 126 ]
//=> number: 75, returns false, result: [ 126 ]
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
//=> number: 5, returns false, result: []
//=> number: 126, returns true, result: [ 126 ]
//=> number: 75, returns false, result: [ 126 ]
//=> number: 199, returns true, result: [ 126, 199 ]
```

```
var numbers = [ 50, 126, 75, 199 ];

numbers.filter(function (number) {
    return number > 100;
});
//=> result = []
//=> number: 5, returns false, result: []
//=> number: 126, returns true, result: [ 126 ]
//=> number: 75, returns false, result: [ 126 ]
//=> number: 199, returns true, result: [ 126, 199 ]
//=> [ 126, 199 ]
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> 26
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> sumSoFar = 5
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> sumSoFar = 5
//=> sumSoFar: 5, number: 6, returns 5 + 6
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> sumSoFar = 5
//=> sumSoFar: 5, number: 6, returns 5 + 6
//=> sumSoFar: 11, number: 7, returns 11 + 7
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> sumSoFar = 5
//=> sumSoFar: 5, number: 6, returns 5 + 6
//=> sumSoFar: 11, number: 7, returns 11 + 7
//=> sumSoFar: 18, number: 8, returns 18 + 8
```

```
var numbers = [ 5, 6, 7, 8 ];

numbers.reduce(function (sumSoFar, number) {
    return sumSoFar + number;
});
//=> sumSoFar = 5
//=> sumSoFar: 5, number: 6, returns 5 + 6
//=> sumSoFar: 11, number: 7, returns 11 + 7
//=> sumSoFar: 18, number: 8, returns 18 + 8
//=> 26
```

Summary

- Arrays have a set of chainable methods that simplify looping
- Strings can access these methods by being converted via `split`
- Each method accepts a function as an argument

`forEach`	applies the function to each element and returns nothing
`map`	returns an array with the function applied to each element
`filter`	applies the function to each element, and returns an array with only those elements for which it returns true
`some/every`	returns true if the function returns true on any/all of the elements and false otherwise. It may not run on the entire array
`reduce`	applies the function to each element, and allows for an “accumulator” to be passed into the next iteration

Practice

There are 10 questions in `readme.md` that relate to the `names.html` practice page.

Also complete the following functions in `practice.js`:

- `reverse(arr)`
- `flatten(arrayOfArrays)`
- `sumOfMultiplesOf3And5(maxNumber)`
- `atLeastOneVowel(str)`
- `longestAwesomeTweet(tweets)`
- `elementsToContent(htmlElements)`
- `randomArray(length, max)`
- `randomElements(values, length)`

Objects

Overview

Important Concepts

- Objects
- keys and values
- `Object.keys`

At the end of this section, you will be able to

- use an object to aggregate structured data.
- use the dot and square-bracket operators to access object values.
- use `Object.keys` to construct arrays from objects.

Summary

- *Objects* are like arrays, but the indices can be strings instead of numbers
- Each entry in an object contains a *key* and a *value*
- Unlike an array, there is no notion of ordering in an object
- We access object values with square brackets and the value's key
- We can also access an object's values with the dot-operator
- Objects don't have `map`, `filter`, `reduce`, etc. but we can use `Object.keys` to leverage them

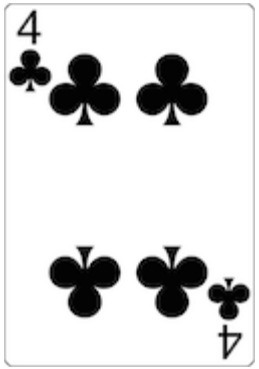
Practice

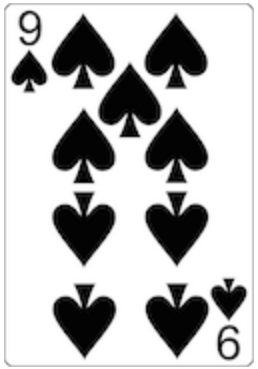
There are 10 questions in `readme.md` that relate to the `tweets.html` practice page.

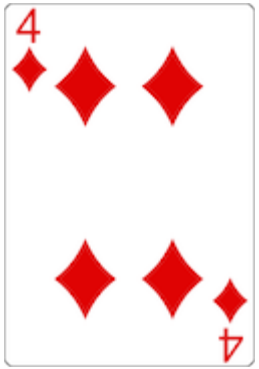
In addition, complete the following functions in the `practice.js` file.

- `isUser(potentialUser)`
- `userToDiv(user)`
- `userWithTweetsToDiv(user)`
- `frequencies(list)`

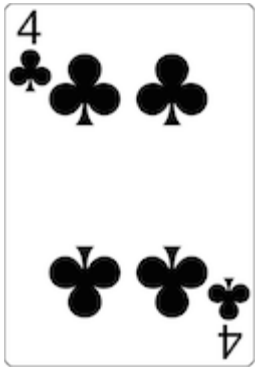
Poker Simulation

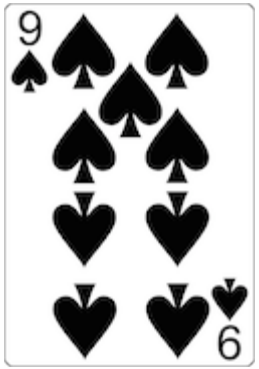


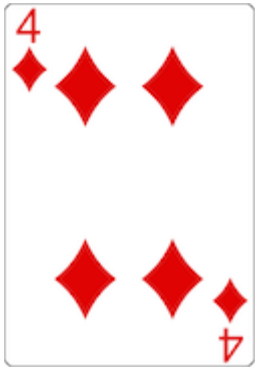




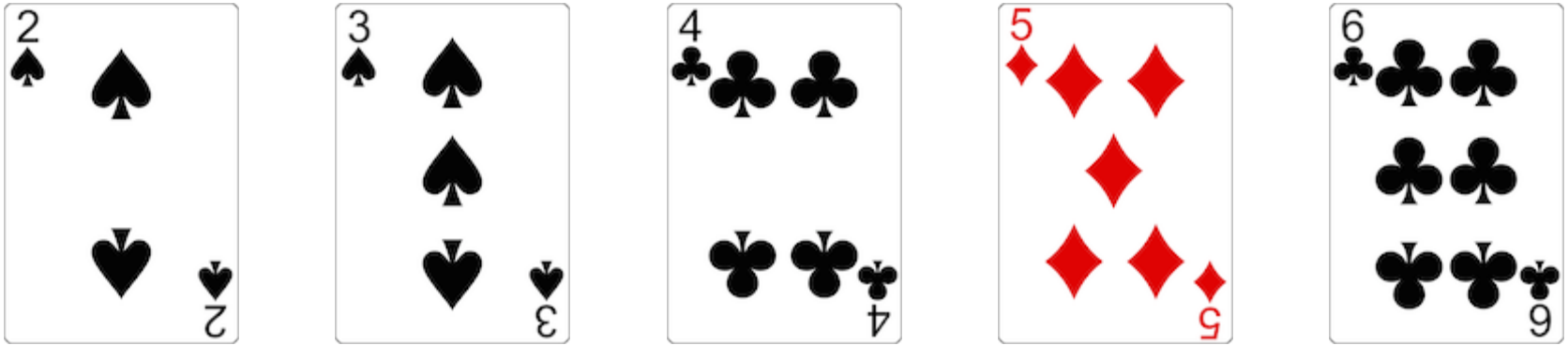
a pair



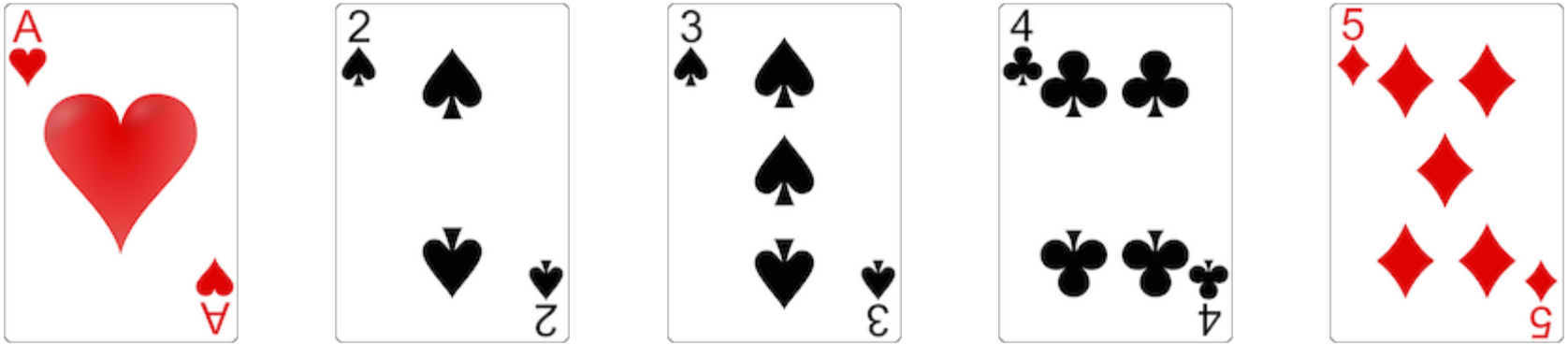




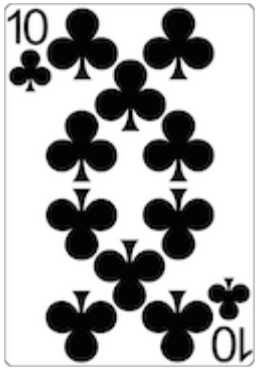
two pair



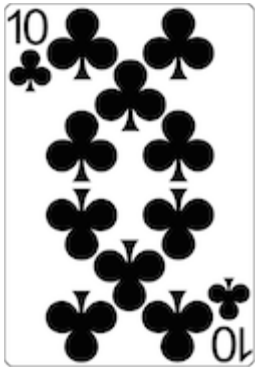
one type of straight

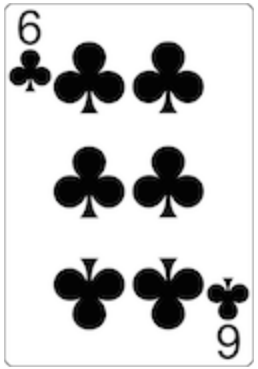


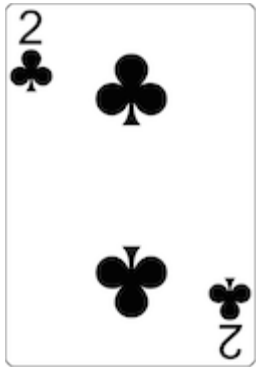
another type of straight (low ace)

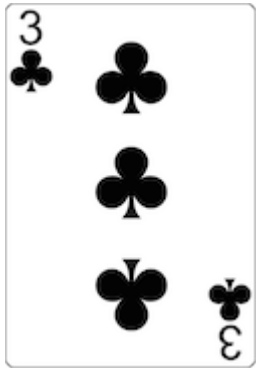


yet another type of straight (high ace)

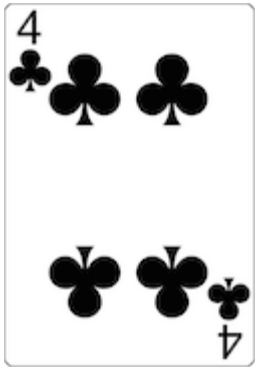
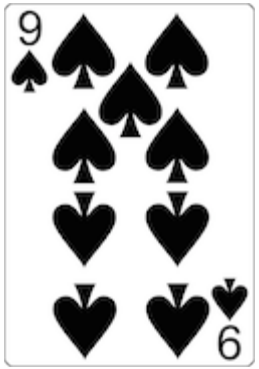
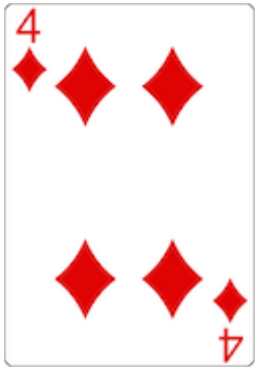




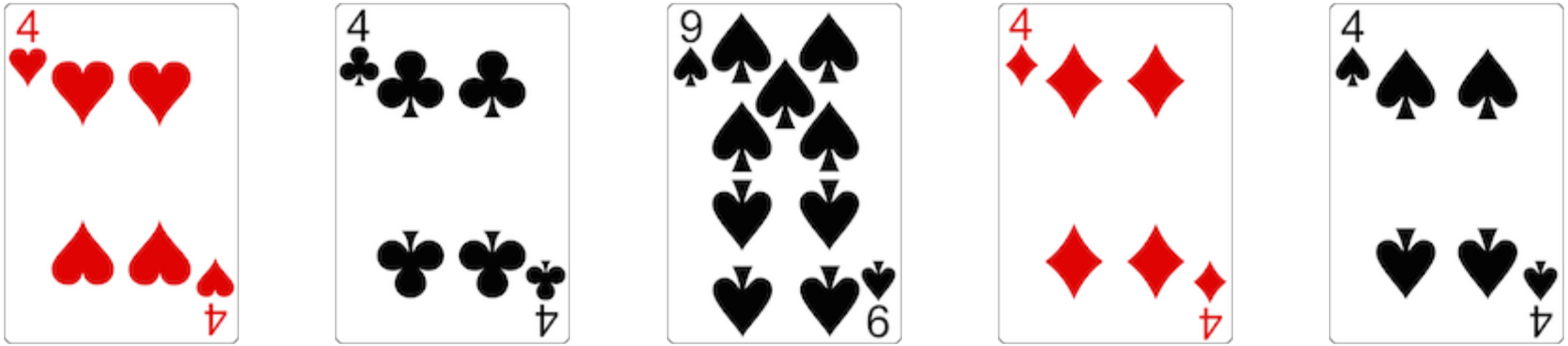




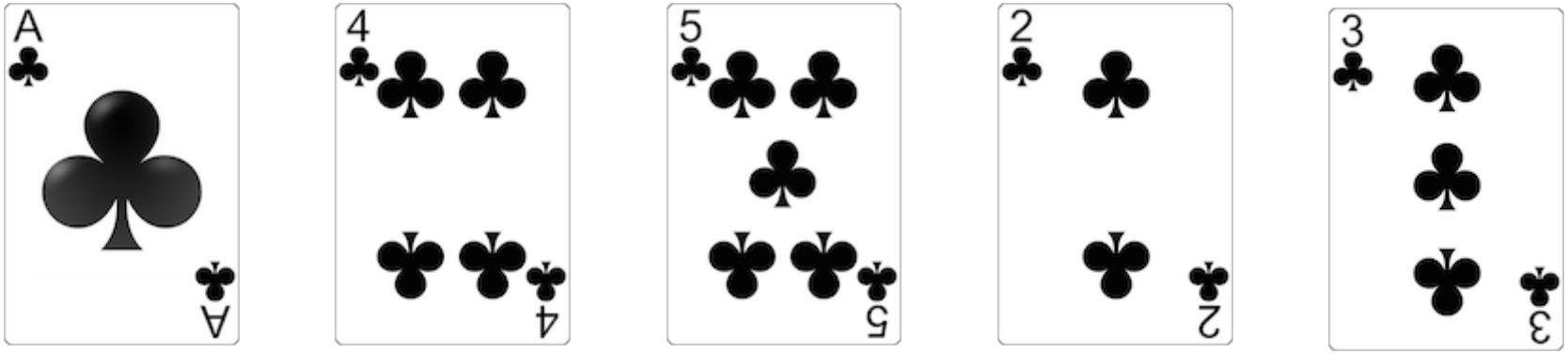
flush

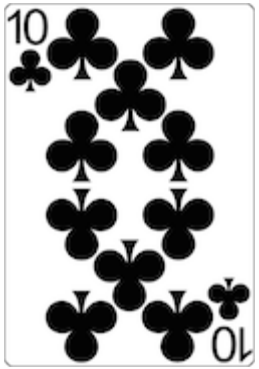

full house



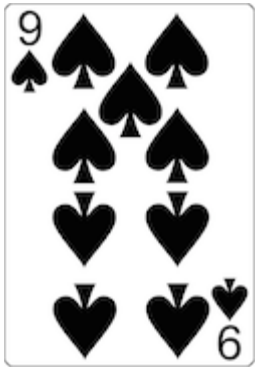
four of a kind

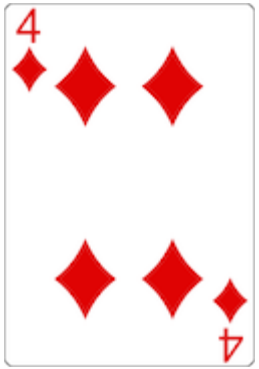


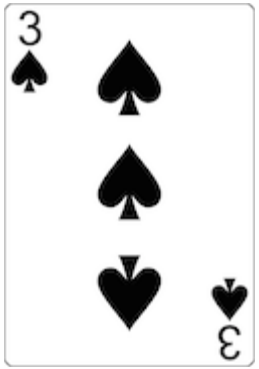
straight flush (ace low)



royal flush







high card (no other pattern)