PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

Lesson 02: Objects, Arrays, and Functions

JavaScript 310B

Intro to JavaScript in the Browser

CLASS OBJECTIVES

- Get acquainted with rich data types in JavaScript, Objects and Arrays
- Learn various ways of how to access, set, and change properties
- ES6 syntax helpers
- Primitive vs Reference Values
- Intro to Regular Expressions

OBJECTS

Objects in JavaScript are basically just key /value pairs. The key must be a string, value may be anything (even objects in objects).

```
const myObject = {};
// preferred way to create empty object

const person = {

  firstName: 'John',

  'last name': 'Doe',  // quote if spaces

  age: 47

}; // initialize with values
```

OBJECTS

Accessing values with 'dot notation' or 'bracket notation'

```
person.firstName; // 'John' – dot notation

person['firstName']; // 'John' – bracket notation


const lastNameKey = 'last name';

person[lastNameKey]; // 'Doe' bracket notation can
be used when the key name may not be known
```

OBJECTS

Adding or updating values

```
person.firstName = 'Mark'; // updates to 'Mark'

person['firstName'] = 'Juan'; // updates to 'Juan'

person.hasKids = true; // Adds new property
hasKids, sets to true
```

OBJECT Destructuring

A shorter way to extract values from objects

```
const person = {firstName: 'John', age: 47};
let {firstName, age} = person;


// same as:
let firstName = person.firstName;
let age = person.age;
```

Key Name Same as Variable

Shorter way to create an object where

key = variable name,

value = variable value

```
//new way
const firstName= 'Merlin';
const jobTitle = 'Magician';
const person = {
  firstName,
  jobTitle,
};
```

EXERCISE: You as an Object (10 min)

Make an object that represents you. It should include:

- `firstName`
- `lastName`
- `'favorite food'`
- `bestFriend` (another person object that has the same 3 properties as above).

Then `console.log` the `bestFriend` name and your favorite food

ARRAYS

Arrays are a unique type of Object.

Each item in an array can be any value, including objects and other arrays.

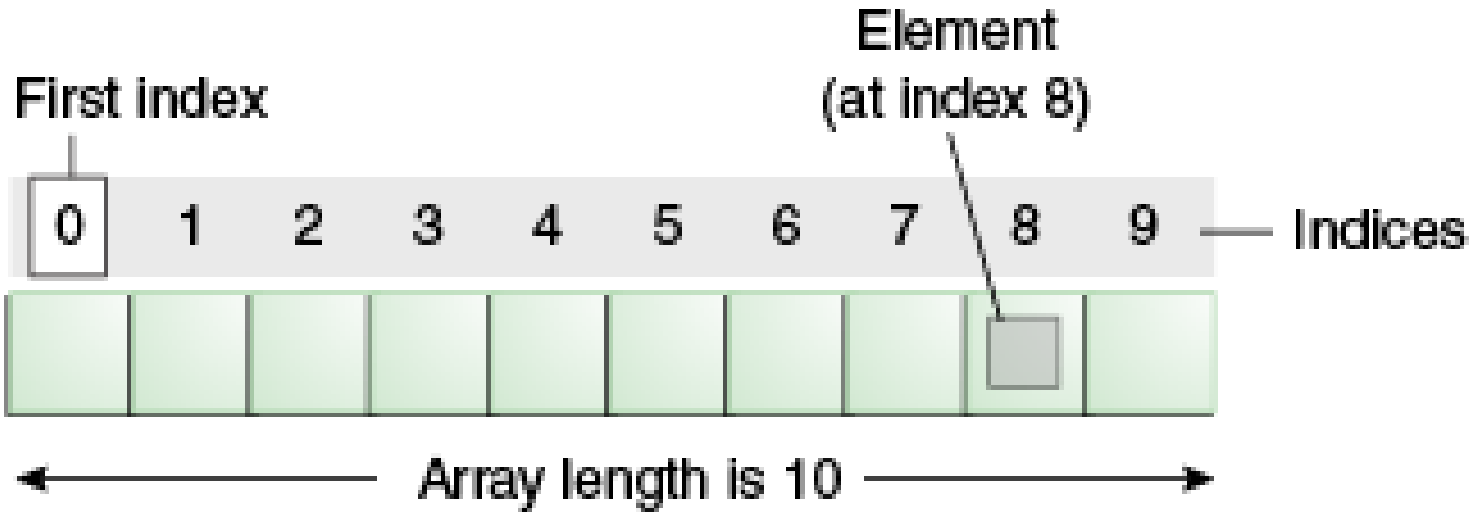
```
const myArray = new Array();

const myArray = []; // preferred, same as above

const xMen = [
  'Rogue',
  'Wolverine',
  'Jubilee'
];
```

ARRAYS

An array is a sort of simple list without `key: value` pairs. However, every item in an array corresponds to an `index`, beginning with index `0`.



ARRAYS

Access values with [ ]

```
xMen[0]; // 'Rogue'

xMen[xMen.length - 1]; // last value 'Jubilee'
```

Array methods can change (or 'mutate') the array called on, or they can return a new array with changes made.

```
xMen.pop(); // returns & removes 'Jubilee', xMen
array now is: ['Rogue', 'Wolverine']

xMen.push('Cyclops'); // xMen array now is:
['Rogue', 'Wolverine', 'Cyclops']
```

ARRAYS

Examples of Accessor methods

```
xMen.concat(['Colossus', 'Jean Gray']); // returns
['Rogue', 'Wolverine', 'Cyclops', 'Colossus',
'Jean Gray'].  xMen is still ['Colossus', 'Jean
Gray']

xMen = xMen.concat(['Colossus', 'Jean Gray']); //
now xMen is ['Rogue', 'Wolverine', 'Cyclops',
'Colossus', 'Jean Gray']

xMen.slice(1, 3); // returns ['Wolverine',
'Cyclops'], xMen is not changed from previous
```

ARRAYS

You can change values directly by specifying the index of an item in a bracket notation.

```
xMen[4] = 'Phoenix';

// now xMen is ['Rogue', 'Wolverine', 'Cyclops',
'Colossus', 'Phoenix']


xMen[8] = Angel';

// now xMen is ['Rogue', 'Wolverine', 'Cyclops',
'Colossus', 'Phoenix', 'Angel']

// BUT xMen.length is now 9

// Don't recommend!
```

ARRAY Destructuring

Similar to object destructuring, there is a simplified syntax for quickly extracting values from an array.

```
const xMen = ['Cyclops', 'Colossus', 'Phoenix'];


const [cyclops, colossus, phoenix] = xMen;
// same as:
const cyclops = xMen[0];
const colossus = xMen[1];
const phoenix = xMen[2];
```

Multi-Dimensional Arrays

Since arrays themselves can be items in an array, this allows for a useful code pattern called multi-dimensional arrays that can programmatically represent 2, 3, why, a *virtually unlimited* number of dimensions!

```
> const twoDimensionalArray =
  [[row1Col1, row1Col2, row1Col3],
   [row2Col1, row2Col2, row2Col3],
   [row3Col1, row3Col2, row3Col3]]
```

EXERCISE: Tic Tac Toe (10 min)

	O	
	X	O
X		X

You can use '-' for the empty squares.

After the array is created, make an update so that 'O' claims the top right square.

Log the grid to the console.

HINT: *log each row separately*

PRIMITIVE vs REFERENCE values

Primitive values . . .

- Primitive values are strings, numbers, booleans, and a few special values (null, undefined)
- Primitive values are copied when a variable takes on the value of another variable

```
let nickName = 'Matt';

const myName = nickName; // myName is 'Matt'

nickName = 'Sleepy';

// myName is still 'Matt', nickName is 'Sleepy'
```

PRIMITIVE vs REFERENCE values

Reference values . . .

- Objects are reference values (since arrays are objects, they are also reference values).
- When a variable takes on the value of another variable that is a reference value, it is not copied. They both point to the same thing

```
const scooter = {age: 12, name: 'Scooter'};

const myDog = scooter;

scooter.favTreat = 'jerky';

// both equal {age: 12, name: 'Scooter', favTreat:
'jerky'}
```

PRIMITIVE vs REFERENCE values

Reference values . . .

- However, if you set one variable to a new object, the variables will no longer be equal

```
let numbers = [1, 2];

const myArray = numbers;

numbers.push(3); // both equal [1,2,3]


numbers = numbers.concat(99); // creates a new array

// numbers is [1,2,3,99]

// myArray is [1,2,3]
```

ARRAY: Spread Operators

Spread operators allow you to quickly copy an array (or object) with simplified syntax

```
const states = ['WA', 'OR'];

const stateCopy = [...states];

stateCopy[0] = 'CA';

states // ['WA', 'OR']

stateCopy // [CA', 'OR']


// Can also combine:
const manyStates = ['NM', 'AZ', ...states];

manyStates // ['NM', 'AZ', 'WA', 'OR']
```

OBJECT: Spread Operators

Spread operators allow you to quickly copy an array (or object) with simplified syntax

```
const russell = { touchdowns: 31, yards: 4110 };

const russellCopy = { ...russell };

russellCopy.yards = 4300;

russell // { touchdowns: 31, yards: 4110 }

russellCopy // { touchdowns: 31, yards: 4300 }
```

OBJECT: Spread Operators

```
// Can add new keys & have duplicate keys
// If keys are repeated, last value is kept

const russellWeek17 = {
  ...russell,
  touchdowns: 33,
  completionPercent: 66
};

russellWeek17
// {
//   yards: 4110
//   touchdowns: 33,
//   completionPercent: 66
// }
```

Regular Expressions (REGEX)

- Regular expressions are super useful to test for a pattern or extract information from a string.
- Great resource for testing regular expressions: https://rubular.com/.
- Create a regular expression by starting and ending with front slash '/' characters.

```
var testExp = new RegExp('^\d+$');
var testExp2 = /^\d+$/; // preferred
// This particular expression would match a string
// that included only digits
```

REGEX Capturing Groups

- Remember trying to pull out specific information from an address? We can pull this info using capturing groups.
- Create a capturing group by surrounding any part of the regex in parenthesis.
- Let's extract the info from this: 'Seattle, WA 98101'

```
var addressParts =
address.match(/^([^,]+),\s*(\w+)\s*(\d+)$/);

// ["Seattle, WA 98101", "Seattle", "WA", "98101]
```

EXERCISE: Validate Email (10 min)

You are given an email as string `myEmail`, make sure it is in correct email format. Should be 1 or more characters, then @ sign, then 1 or more characters, then dot, then one or more characters - no whitespace:

foo@bar.baz

Hints:

Use rubular to check a few

Use regexp test method: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test