Topic - Javascript

About Me

Anushree Rai

Javascript

Senior Software Engineer

Anushree has 2.6 years of experience in the IT industry across the Fin-Tech domain

Expertise : Java (Spring, Hibernate)
JavaScript (Angular, AngularJs)
Database (MySQL, PostgreSql)
Extras: Blockchain, Rabbitmq

Agenda

01 JS Fundamentals
02 Prototypes
03 Event
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 RegEx & Other Concepts
09 Questions

Agenda

01 **JS Fundamentals**
02 Prototypes
03 Events
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 Other Concepts
09 Questions

Overview

- What is JavaScript?
  - It is the language of Web.
  - Used along with HTML , CSS.
  - Controls behavior of DOM elements.
  - Makes dynamic / interactive web-pages on the client side.

- Who developed it?
  - Brenden Eich (for Netscape)
  - Original name in the 90s was Mocha.
  - Later was called as LiveScript.
  - Standardly called as ECMAscript.
  - Generally called as Javascript.

What are its Data Types ?

Primitive Data Types

String : represents sequence of characters e.g.”Howdy!”

Number : represents numeric values e.g. 100

Boolean : represents boolean values e.g. false or true

Undefined : represents undefined value

Null : represents null i.e. no value at all

Symbol (new in ECMAScript 2015). A data type whose instances are unique and immutable.

Non-Primitive Data Types

Object : represents instance through which we can access members

Array : represents group of similar values

RegExp : represents regular expression

Basics

How to declare variables ?

- With the keyword var. ⇒ var x = 42; (function scoped)
- By simply assigning it a value. ⇒ x = 42;
- With the keyword let. ⇒ let y = 13; (block scoped)

Evaluating variables

```
var a;
console.log("The value of a is " + a); // logs "The value of a is undefined"
console.log("The value of b is " + b); // throws ReferenceError exception

console.log("The value of x is " + x); // throws ReferenceError exception
let x;
```

Evaluating variables

- undefined can be used to whether a variable has value.

```
var input;
 if(input === undefined){
   doThis();
 } else {
   doThat();
 }
```

- undefined behaves as false when used in boolean context

```
var myArray = [];
 if (!myArray[0]) myFunction();
```

- undefined value converts to NaN when used in numeric context.

```
var a;
 a + 2;  // Evaluates to NaN
```

- null value behaves as 0 in numeric contexts and as false in boolean contexts.
  For example:

```
var n = null;
 console.log(n * 32); // Will log 0 to the console
```

Data Type Conversion

- JavaScript is a dynamically typed language.

```
var answer = 42;

answer = "Thanks for all the fish...";

x = "The answer is " + 42 // "The answer is 42"

y = 42 + " is the answer" // "42 is the answer"

"37" - 7 // 30

"37" + 7 // "377"
```

Converting strings to numbers

parseInt()

parseFloat()

parseInt will only return whole numbers.
Use *radix* parameter to specify numeric system.
+ (unary plus) operator for string to number conversion.

```
"1.1" + "1.1" = "1.11.1"
(+"1.1") + (+"1.1") = 2.2
// Note: the parentheses are added for clarity, not required.
```

Literals

You use literals to represent *values* in JavaScript.
These are *fixed values*, not variables, that you *literally* provide in your script.

- Array literals
- Boolean literals
- Floating-point literals
- Integers
- Object literals
- RegExp literals
- String literals

Literals

Array literals

```
var coffees = ["French Roast", "Colombian", "Kona"];
```

- Arrays are also Objects
- Array literals are also Array objects.

Extra commas in array literals

```
var fish = ["Lion", , "Angel"];

var myList = ['home', , 'school', ];

var myList = [ , 'home', , 'school'];

var myList = ['home', , 'school', , ];
```

Literals

Boolean literals

- The Boolean type has two literal values: true and false.
- The Boolean object is a wrapper around the primitive boolean data type.

Integers

Decimal integer literal consists of a sequence of digits without a leading 0 (zero).
Leading 0 (zero) on an integer literal, or leading 0o (or 0O) indicates it is in octal.
Leading 0x (or 0X) indicates hexadecimal.
Leading 0b (or 0B) indicates binary.

Examples::
0, -345 (decimal, base 10), 015, -0o77 (octal, base 8)
0x1123, 0x00111 and -0xF1A7 (hexadecimal, "hex" or base 16)
0b11, 0b0011 and -0b11 (binary, base 2)

Floating-point literals

A decimal integer which can be signed (preceded by "+" or "-"),
A decimal point ("."),
A fraction (another decimal number),
An exponent.

Variable Scope

- Global Variable
- Local Variable
- Block Statement Scope

```
if (true) {
  var x = 5;
}
console.log(x);  // 5
```

- This behavior changes, when using the let declaration introduced in ECMAScript 2015.

```
if (true) {
  let y = 5;
}
console.log(y);  // ReferenceError: y is not defined
```

Literals

Object Literals

- A list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}).
- Use a *numeric* or *string* literal for the name of a property
- You can nest an object inside another.
- Property names can be any string, including the empty string
- Use Quotes if invalid identifiers or numbers are property names
- Property names that are not valid identifiers cannot be accessed as a dot (.) property, but can be accessed and set with the array-like notation("[]").

Variable Hoisting

```
/**
 * Example 1
 */
console.log(x === undefined); // ??
var x = 3;
```

```
/**
 * Example 2
 */
var myvar = "my value";

(function() {
  console.log(myvar); // ??
  var myvar = "local value";
})();
```

Variable Hoisting

```
/**
 * Example 1
 */
console.log(x === undefined); // undefined
var x = 3;
```

Why didn't JS throw *ReferenceError* ?

```
/**
 * Example 2
 */
var myvar = "my value";

(function() {
  console.log(myvar); // undefined
  var myvar = "local value";
})();
```

Variable Hoisting

```
/**
 * Example 1
 */
console.log(x === undefined); // true
var x = 3;

/**
 * Example 2
 */
// will return a value of undefined
var myvar = "my value";

(function() {
  console.log(myvar); // undefined
  var myvar = "local value";
})();
```

```
/**
 * Example 1
 */
var x;
console.log(x === undefined); // true
x = 3;

/**
 * Example 2
 */
var myvar = "my value";

(function() {
  var myvar;
  console.log(myvar); // undefined
  myvar = "local value";
})();
```

Best Practice:
All var statements in a function should be placed as near to the top of the function as possible.

Global Variables, Constants

Global variables

- Global variables are in fact properties of the *global object*.
- In web pages the global object is window

Constants

- read-only, named constant
- const keyword
- Syntax for Constant Identifier same as Variable Identifier.The syntax of a constant identifier is the same as for a variable identifier

```
const PI = 3.14;
```

- Value can’t be changed or re-declared.
- Has to be initialized to a value.
- Scope rules for constants are same as those for let block scope variables.
- Cannot declare a constant with the same name as a function or variable in the same scope.

Built-in Objects

Value properties

These global properties return a simple value; they have no properties or methods.

- Infinity
- NaN
- undefined
- null literal

Function properties

These global functions—functions which are called globally rather than on an object—directly return their results to the caller.

- eval()
- isFinite()
- isNaN()
- parseFloat()
- parseInt()
- decodeURI()
- decodeURIComponent()
- encodeURI()
- encodeURIComponent()

Built-in Objects

Fundamental objects

These are the fundamental, basic objects upon which all other objects are based. This includes objects that represent general objects, functions, and errors.

Object
Function
Boolean
Symbol
Error
EvalError
InternalError
RangeError
ReferenceError
SyntaxError
TypeError
URIError

Numbers and dates

These are the base objects representing numbers, dates, and mathematical calculations.

Number
Math
Date

Built-in Objects

Text processing

These objects represent strings and support manipulating them.

String

RegExp

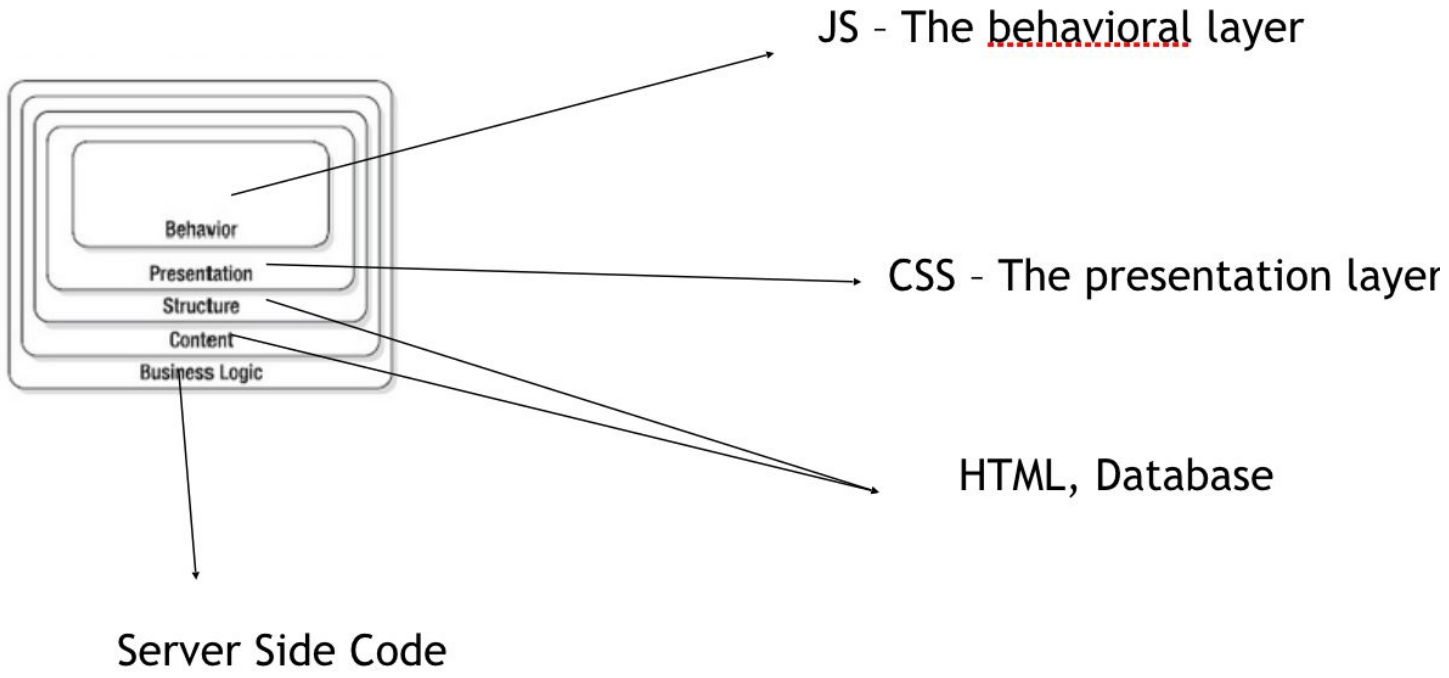
Indexed collections

These objects represent collections of data which are ordered by an index value. This includes (typed) arrays and array-like constructs.

Array

Place in Web

- ***Why do we use it?***
  - Dynamic DOM Manipulation
  - We need some scripts to be executed on client side



DOM Manipulation

- The HTML DOM model is constructed as a tree of Objects. With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

DOM Manipulation

Finding HTML Elements

Method	Description
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Adding and Deleting Elements

Method	Description
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick=function() {*code*}	Adding event handler code to an onclick event

Changing HTML Elements

Method	Description
element.innerHTML=	Change the inner HTML of an element
element.attribute=	Change the attribute of an HTML element
element.setAttribute(*attribute,value*)	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

Operators

If **x = 20, y = 5, and z = result**, we have:

Operator	Java Script Example	Result
Addition: +	z = x + y	z = 25
Subtraction: -	z = x - y	z = 15
Multiplication: *	z = x * y	z = 100
Division: /	z = x / y	z = 4
Modulus: %	z = x / y	z = 0
Increment: ++	z = ++x	z = 21
Decrement --	z = --x	z = 19

If x = 10 we have:

Operator	What is it?	Example
==	equal to	x == 5 is false
===	exactly equal to value and type	x === 10 is true x === "10" is false
!=	not equal	x != 2 is true
>	greater than	x > 20 is false
<	less than	x < 20 is true
>=	greater than or equal to	x >= 20 is false
<=	less than or equal to	x <= 20 is true

Programming Constructs

Conditional Statement -
- if
- if … else
- switch

Repetitive Statement -
- for
- while
- do...while

Standardization & JS Engine

Standardization

- Microsoft's JScript and Netscape's Javascript standardized to ECMAScript...

- The current edition of the ECMAScript standard is 11, released in June 2020

JS Engine

- A JavaScript engine is process virtual machine which interprets and executes JavaScript .

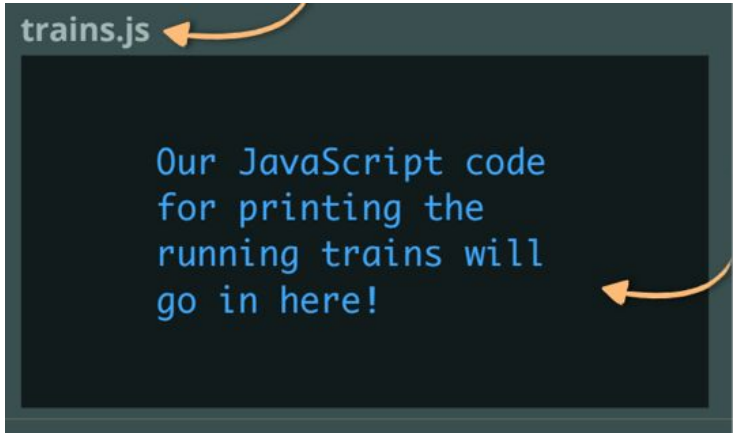
JS Engine	Browser
Spider Monkey (First JS Engine)	Netscape
V8	Chrome
Rhino,Tracemonkey, JagerMonkey, IonMonkey, OdinMonkey	Firefox
JavaScriptCore (Nitro Engine)	Safari
Carakan	Opera

JS in web pages

Including JS in a browser

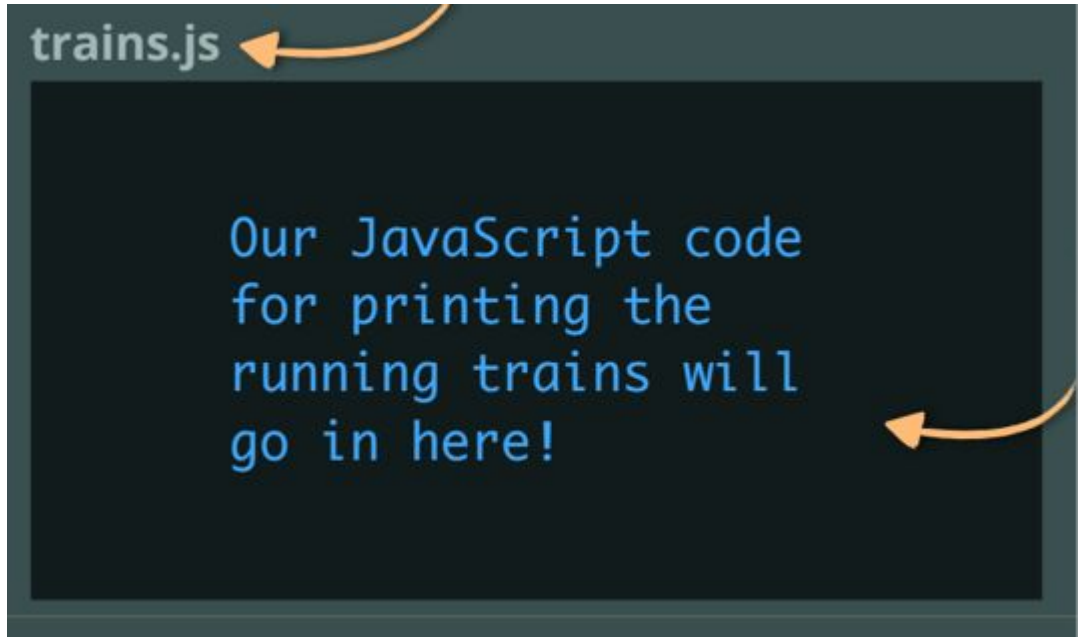
- JavaScript can be put in the <head> or in the <body> of an HTML document
- JavaScript snippets can be put in a separate .js file
- JavaScript can be put in an HTML form object, such as a button

```
<html>
<header>
<script src="trains.js"></script>
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```



JS in web pages

Building a file of JavaScript code that can be used repeatedly by our site



JS in web pages

Where do these files go?

So that our src = "trains.js" code works correctly, we'd need to place our trains.js file in the same directory as the index.html file.

```
<html>
<header>
<script src="trains.js"></script>
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

JS in web pages

Staying organized as you code


```
<html>
<header>
<script src="scripts/trains.js"></script>
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

JS in web pages

Link syntax for distant files


```
<html>
<header>
<script src="../scripts/trains.js"></script>
</header>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

Agenda

01 JS Fundamentals
02 **Events**
03 Prototypes
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 Other Concepts
09 Questions

Events

- HTML events are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Attaching Event Handlers

With single quotes:
*<element event='**some JavaScript**'>*

With double quotes:
*<element event="**some JavaScript**">*

Using Javascript itself.
document.getElementById("myCheck").click = function() {}

The Window Object

- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.
- Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
is the same as:
document.getElementById("header");
```

- Window.open()
- Window.close()

Common HTML Events

Event	Description
Onload	Document's contents has been loaded.
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Exercise

1. Create a sample page with a button on it.
- Clicking the button should show an alert saying ‘Hello World’.

2. Create a sample page with a button on it.
- Clicking the button should alert 'Thanks for visiting my site!' and than redirect it to another website.

3. Create a sample page with two text box for input and one text box for output. And One button.
- Clicking on the button should display the result of addition of the two input numbers in the output box.

Agenda

01 JS Fundamentals
02 Events
03 **Prototypes**
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 Other Concepts
09 Questions

Prototype

- Every JavaScript object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype.
- Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype.
- Objects created with new Date() inherit the Date.prototype.
- The Object.prototype is on the top of the prototype chain.
- All JavaScript objects (Date, Array, RegExp, Function, ...) inherit from the Object.prototype.
- Create an object prototype is to use an object constructor function:

```
function Person(first, last, age, eyecolor) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eyecolor;
}
```

- Use the **new** keyword to create new objects from the same prototype:

●The constructor function is the prototype for Person objects.

●It is considered good practice to name constructor function with an upper-case first letter.

●Adding new properties (or methods) to an existing object.

- Adding new properties (or methods) to all existing objects of a given type.
- To add a new property to a constructor, you must add it to the constructor function:
- Prototype properties can have prototype values (default values).
- JavaScript prototype property allows you to add new properties to an existing prototype:

- Only modify your **own** prototypes. Never modify the prototypes of standard JavaScript objects.

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 **Functions**
05 Arrays
06 Maps
07 JSON Parsing
08 Other Concepts
09 Questions

Functions

- A JavaScript function is a block of code designed to perform a particular task.

- Function Declaration Syntax -

```
functionName(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Invoking a function -
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

- JavaScript Functions are Objects
- JavaScript Functions can be assigned to a variable
- JavaScript functions can also be passed as values to other functions.

Function Hoisting

```
/* Function declaration */

foo(); // "bar"

function foo() {
  console.log("bar");
}


/* Function expression */

baz(); // ReferenceError: baz is not a function

var baz = function() {
  console.log("bar2");
};
```

Functions

Global Variables -

- global variables belong to the window object
- can be used (and changed) by all scripts in the page
- live as long as application live

Local Variables -

- can only be used inside the function where it is defined
- have short lives

A counter Dilemma -

```
var counter = 0;

function add() {
    counter += 1;
}

add();
add();
add();
```

```
function add() {
    var counter = 0;
    counter += 1;
}

add();
add();
add();
```

Functions

- Private variables can be made possible with closures.

```
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})()

add();
add();
add();
```

- variable add is assigned the return value of a self invoking function
- The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression
- How is counter accessible in the recursive return statement?
- The "wonderful" part is that add is now a function and can access the counter in the parent scope.
- This is called a JavaScript closure.

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 Functions
05 **Arrays**
06 Maps
07 JSON Parsing
08 Other Concepts
09 Questions

Arrays

- The JavaScript **Array** object is a global object that is used in the construction of arrays; which are high-level, list-like objects.
- Array indexes start at 0.
- Arrays can grow dynamically – just add new elements at the end.
- Arrays can have *holes*, elements that have no value.
- Array elements can be anything - numbers, strings, or arrays!
- Javascript does not support 2-dimensional arrays natively (as part of the language)
- BUT – each array element can be an array.
- Example:var board = [[1,2,3],
  [4,5,6],
  [7,8,9]];

```
for (i=0;i<3;i++)
  for (j=0;j<3;j++)
   board[i][j]++;
```

Access and Assignment

```
1.  var box = [];
2.
3.  box['size'] = 9;
4.  box['0'] = 'meow';
5.
6.  box['size']; //??
7.  box[0]; //??
```

Access and Assignment

```
1.  var box = [];
2.
3.  box['size'] = 9;          //Similar to a map, but not for numeric indices!
4.  box['0'] = 'meow';
5.
6.  box.size; //??
7.  box.0; //??

 //try on W3 schools
```

Native Properties

```
1.  var box = [];
2.
3.  box['size'] = 9;
4.  box['0'] = 'meow';
5.
6.  box.length; //??
```

Native Properties

```
1.  var box = [];
2.
3.  box['size'] = 9;
4.  box['0'] = 'meow';
5.  box[3] = {'babyBox':true};
6.
7.  box['length']; //??
```

Undefined???

Some Array Methods

Add to the end of an Array

```
var newLength = fruits.push("Orange");
// ["Apple", "Banana", "Orange"]
```

Remove from the end of an Array

```
var last = fruits.pop(); // remove Orange (from the end)
// ["Apple", "Banana"];
```

Remove from the front of an Array

```
var first = fruits.shift(); // remove Apple from the front
// ["Banana"];
```

Add to the front of an Array

```
var newLength = fruits.unshift("Strawberry") // add to the front
// ["Strawberry", "Banana"];
```

Find the index of an item in the Array

```
fruits.push("Mango");
// ["Strawberry", "Banana", "Mango"]

var pos = fruits.indexOf("Banana");
// 1
```

Remove an item by Index Position

```
var removedItem = fruits.splice(pos, 1); // this is how to remove an item
// ["Strawberry", "Mango"]
```

Copy an Array

```
var shallowCopy = fruits.slice(); // this is how to make a copy
// ["Strawberry", "Mango"]
```

Iterating over Array

```
function logArrayElements(element, index, array) {
  console.log('a[' + index + '] = ' + element);
}


// Notice that index 2 is skipped since there is no item at
// that position in the array.
[2, 5, , 9].forEach(logArrayElements);
// logs:
// a[0] = 2
// a[1] = 5
// a[3] = 9
```

Excercise

```
1.  var box = {};
2.
3.  box["material"] = "cardboard";
4.
5.  box["^&*"] = ‘meow’;
6.
7.  box.material; //??
8.  box.^&*; //??
```

```
1.  var box = {
2.   "material" : "cardboard";
3.   "^&*" : ‘meow’;
4.  };
5.
6.  box[‘material’]; //??
7.  var key = ‘^&*’;
8.  box[key]; //??
```

```
1.  var box = {};
2.
3.  box["material"] = "cardboard";
4.
5.  box["^&*"] = ‘meow’;
6.
7.  var key = “material”;
8.  box.material; //??
9.  box.key; //??
10. var key = “^&*”;
11. box[‘key’]; //??
```

```
1.  var foo = {a: "alpha", 2: "two"};
2.  console.log(foo.a);    // ??
3.  console.log(foo[2]);   // ??
4.  console.log(foo.2);  // ??
5.  console.log(foo[a]); // ??
6.  console.log(foo["a"]); // ??
7.  console.log(foo["2"]); // ??
```

Excercise

1. Write a JavaScript program along with UI which accept a string as input and swap the case of each character and print in the output box on clicking the button. For example if you input 'Accolite' the output should be 'aCCOLITE'.

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 Functions
05 Arrays
06 **Maps**
07 JSON Parsing
08 Other Concepts
09 Questions

Map

- The **Map** object is a simple key/value map.
- Any value (both objects and primitive values) may be used as either a key or a value.
- Syntax **:** new Map([iterable])
- Iterable is an Array or other iterable object whose elements are key-value pairs.
- null is treated as undefined.
- A Map object iterates its elements in insertion order
- a for...of loop returns an array of [key, value] for each iteration.

Map : methods

- Map.prototype.clear()
- Map.prototype.delete(key)
- Map.prototype.entries()
- Map.prototype.forEach(callbackFn[, thisArg])
- Map.prototype.get(key)
- Map.prototype.has(key)
- Map.prototype.keys()
- Map.prototype.set(key, value)
- Map.prototype.values()
- Map.prototype[@@iterator]()

Objects vs Map

- Objects have been used as Maps historically.
- But performance wise, maps perform better.
- The keys of an Object can only be Numbers, Strings or Symbols, whereas they can be any value for a Map, even an object!
- You can get the size of a Map easily while you have to manually keep track of size for an Object.

https://www.geeksforgeeks.org/map-vs-object-in-javascript/

Map Examples

Normal Usage of `Map` object

```
var myMap = new Map();

var keyString = "a string",
    keyObj = {},
    keyFunc = function () {};

// setting the values
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, "value associated with keyObj");
myMap.set(keyFunc, "value associated with keyFunc");

myMap.size; // 3

// getting the values
myMap.get(keyString);    // "value associated with 'a string'"
myMap.get(keyObj);       // "value associated with keyObj"
myMap.get(keyFunc);      // "value associated with keyFunc"

myMap.get("a string");   // "value associated with 'a string'"
                         // because keyString === 'a string'
myMap.get({});           // undefined, because keyObj !== {}
myMap.get(function() {}) // undefined, because keyFunc !== function () {}
```

Map Examples

Iterating Maps with `for..of`

```
var myMap = new Map();
myMap.set(0, "zero");
myMap.set(1, "one");


for (var [key, value] of myMap) {
  console.log(key + " = " + value);
}
// Will show 2 logs; first with "0 = zero" and second with "1 = one"

for (var key of myMap.keys()) {
  console.log(key);
}
// Will show 2 logs; first with "0" and second with "1"

for (var value of myMap.values()) {
  console.log(value);
}
// Will show 2 logs; first with "zero" and second with "one"

for (var [key, value] of myMap.entries()) {
  console.log(key + " = " + value);
}
// Will show 2 logs; first with "0 = zero" and second with "1 = one"
```

Map Examples

Iterating Maps with forEach()

```
myMap.forEach(function(value, key) {
  console.log(key + " = " + value);
}, myMap)
// Will show 2 logs; first with "0 = zero" and second with "1 = one"
```

Relation with Array objects

```
var kvArray = [["key1", "value1"], ["key2", "value2"]];

//2D key-value Array into a map
var myMap = new Map(kvArray);

myMap.get("key1"); // returns "value1"

// Use the spread operator to transform a map into a 2D key-value Array.
console.log(uneval([...myMap])); // Will show you exactly the same Array
as kvArray

// Or use the spread operator on the keys or values iterator to get
// an array of only the keys or values
console.log(uneval([...myMap.keys()])); // Will show ["key1", "key2"]
```

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 Functions
05 Arrays
06 Maps
07 **JSON Parsing**
08 Other Concepts
09 Questions

JSON

- **J**ava**S**cript **O**bject **N**otation.
- **JSON.parse()** method parses a string as JSON, optionally transforming the value produced by parsing.
- *Syntax* **:** JSON.parse(text[, reviver])

- **Using the reviver parameter**
- ```
  JSON.parse('{"p": 5}', function(k, v) {
    if (typeof v === 'number') {
      return v * 2;  // return v * 2 for numbers
    }
    return v;       // return everything else unchanged
  });

  // { p: 10 }

  JSON.parse('{"1": 1, "2": 2, "3": {"4": 4, "5": {"6": 6}}}', function(k, v) {
    console.log(k); // log the current property name, the last is "".
    return v;       // return the unchanged property value.
  });
  ```
  ```

JSON

Using `JSON.parse()`

```
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('"foo"');           // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

Using the `reviver` parameter

```
JSON.parse('{"p": 5}', function(k, v) {
  if (typeof v === 'number') {
    return v * 2;  // return v * 2 for numbers
  }
  return v;        // return everything else unchanged
});

// { p: 10 }

JSON.parse('{"1": 1, "2": 2, "3": {"4": 4, "5": {"6": 6}}}', function(k, v) {
  console.log(k); // log the current property name, the last is "".
  return v;       // return the unchanged property value.
});

// 1
// 2
// 4
// 6
// 5
// 3
// ""
```

JSON

JSON.parse() does not allow trailing commas

```
// both will throw a SyntaxError
JSON.parse('[1, 2, 3, 4, ]');
JSON.parse('{"foo" : 1, }');
```

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 **RegEx & Other Concepts**
09 Questions

RegEx

- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text replace operations.
- Used in string searches, replacements, comparisons

Methods associated with regex

- **exec**: A RegExp method that executes a search for a match in a string. It returns an array of information.
- **test**: A RegExp method that tests for a match in a string. It returns true or false.
- **match**: A String method that executes a search for a match in a string. It returns an array of information or null on a mismatch.
- **search**: A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
- **replace**: A String method that replaces the matched substring with a replacement substring.
- **split**: A String method that uses a regular expression or a fixed string to break a string into an array of substrings.

Building an Expression

Enclosed in / characters (/abc/ represents the string "abc")

The expression is an object, created one of two ways

By assignment (note no quotes!)

re = /abc/;

With the RegExp method

re = new RegExp("abc");

Use the latter with user input or where the expression will change

Example:

```
function testRegEx(string, expression)
  {
  re = new RegExp(expression)

  if (re.test(string))
    {
    document.write("<p>A match is found!</p>");
    }
  else
    {
    document.write("<p>No Match</p>");
    }
  }
```

RegEx

Special Characters

- ^ is used to match the beginning of a string: thus /^The/ matches "The fox"
- $ matches at the end: thus /fox$/ matches "The fox"
- Square brackets are used to match any one of the characters listed: thus [aeiou] matches vowels
- \ is used for escaping special characters

- \n is a new line
- \r is a carriage return
- \t is a tab
- \s is a single white space

- + matches the preceding char 1 or more times, thus /do+m/ matches "dom" and "doom"
- * matches the preceding char 0 or more times
- . matches any single character
- ? matches the preceding character 0 or 1 time

typeof Operator

- The **typeof operator** returns a string indicating the type of the unevaluated operand.

- **Syntax :** typeof *operand*
- **Parameters :** *operand* is an expression representing the object or primitive whose type is to be returned.

Type	***Result***
Undefined	"undefined"
Null	"object"
Boolean	"boolean"
Number	"number"
String	"string"
Symbol	"symbol"
Function object	"function"
Any other object	"object"

Web Standards

- Avoid global variables
- Don't Use new Object() -
  - Use {} instead of new Object()
  - Use "" instead of new String()
  - Use 0 instead of new Number()
  - Use false instead of new Boolean()
  - Use [] instead of new Array()
  - Use /(:)/ instead of new RegExp()
  - Use function (){} instead of new function()
- Beware of Automatic Type Conversions
- Use === Comparison
- Never End a Definition with a Comma
- Use semicolons wherever necessary

Agenda

01 JS Fundamentals
02 Events
03 Prototypes
04 Functions
05 Arrays
06 Maps
07 JSON Parsing
08 Other Concepts
09 **Questions**

Any Question?

Thank You