Intro to JS - Part 1 - Day 3: Sept 12

Creating arrays

```
let myArray = [1, 2, 3];
```

OR

```
const myArray2 = []; myArray2[0] = "a"; myArray2[1] = "b";
```

OR

```
const arrayName = new Array();
```

OR

```
let myArray3 = new Array(2); //to specify array size
myArray3[0] = 1; myArray3[1] = "ring to rule them all";
```

Properties

Length

Returns the number of elements in an array.

```
let myArray4 = [1, 2, 3];
console.log(myArray4.length); // Output: 3
```

Constructor

Returns the constructor function that was used to create the array.

```
const myArray5 = [1, 2, 3]; console.log(myArray5.constructor);
```

Prototype

To add custom properties and methods to the Array prototype.

```
Array.prototype.multiplyByTwo = function() {
  const newArray = [];
  for (let i = 0; i < this.length; i++) {
    newArray[i] = this[i] * 2; }
  return newArray;}
const myArray5 = [1, 2, 3];
console.log('multiplyByTwo result: ' + myArray5.multiplyByTwo());
// Expected result: [2, 4, 6]
```

The `prototype` property of an array should be used with care. Adding custom methods and properties to the `Array.prototype` object can affect the behavior of all arrays in your codebase!

Built-in Methods

push()

Adds one or more elements to the end of the array and returns the new length of the array.

pop()

Removes the last element from the array and returns it

shift()

Removes the first element from the array and returns it

unshift()

Adds one or more elements to the beginning of the array & returns the new array length

concat()

Joins two or more arrays and returns a new array

slice()

Returns a new array containing a portion of the original array
Takes two arguments: the starting index & the ending index (exclusive). If the ending index is not specified, returnr all elements from the starting index to the end of the array.

splice()

usually two arguments: the starting index and the number of elements to remove.

Can add elements to the array at the specified starting index by passing additional arguments after the second argument:

Example

```
const myArray10 = ['index0', 'index1', 'index2', 'index3'];
myArray10.splice(2, 0, "a", "b", "c");
console.log(myArray10);
```

indexOf()

Returns the first index at which a specified element can be found, or -1 if it is not present

lastIndexOf()

Returns the last index at which a specified element can be found, or -1 if it is not present

forEach()

Calls a function for each element in the array.

```
let myArray13 = [1, 2, 3, 4, 5];
myArray13.forEach(function(element) { console.log(element); });
```

map()

Calls a function for each element in the array and returns a new array with the results

```
let myArray13 = [1, 2, 3, 4, 5];
const newArray1 = myArray13.map(function(element) {return element * 2;});
// Output: [2, 4, 6, 8, 10]
```

filter()

Calls a function for each element in the array and returns a new array with the elements that pass the test.

```
let myArray13 = [1, 2, 3, 4, 5];
const newArray2 = myArray13.filter(function(element) {
  return element > 2;});
console.log('newArray2: ' + newArray2); // Output: [3, 4, 5]
```

reduce()

Calls a function for each element in the array and returns a single value that is the result of the function. The reduce method takes two arguments:

1) a function that adds the current element to the running total
2) an initial value of 0 for the running total.

```
let myArray13 = [1, 2, 3, 4, 5];
const sum = myArray13.reduce(function(total, element) { return total + element; }, 0);
console.log('sum: ' + sum); // Output: 15
```

Creating Objects / object literals

```
let person1 = { name: 'John', age: 30, city: 'New York' };
```

Accessing Properties & Methods

Both dot notation and bracket notation are valid ways to access object properties, and they can be used interchangeably.

Use bracket notation is when the property name contains special characters or spaces, as these cannot be used with dot notation.

Dot notation

```
console.log('Name: ' + person1.name);
```

Bracket notation

```
console.log('Age' + person1['age']);
```

'this'

this refers to the object that is currently executing a given piece of code. When used inside an object method, **this** refers to the object that the method is called on.

```
let person5 = {
  name: 'Frodo Baggins',
  age: 50,
  home: 'The Shire',
  printName: function() {
    console.log(this.name);
  }
}

person5.printName();
```

Built-in Object Methods

Object.assign()

The **Object.assign()** method is used to copy the values of all enumerable properties (can be accessed using a loop) from one or more source objects to a target object.

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };
Object.assign(target, source);
console.log(target); // { a: 1, b: 4, c: 5 }
```

hasOwnProperty()

To determine whether an object has a property with a specified name. This method returns a boolean value (True or False).

Object.keys()

Returns an array of all the enumerable property names of an object.

```
const obj = { name: "John", age: 30, country: "USA" };
const objectKeys = Object.keys(obj);
console.log(objectKeys); // Output: ["name", "age", "country"]
```

Object.values()

Returns an array of all the enumerable property values of an object.

```
const obj = { name: "John", age: 30, country: "USA" };
const objectValues = Object.values(obj);
console.log(objectValues); // Output: ["John", 30, "USA"]
```

Object.entries()

Returns an array of all the enumerable properties of an object in the form of an array, where each sub-array contains two elements

- the property key
- the property value

```
const obj = { name: "John", age: 30, country: "USA" };
const objectEntries = Object.entries(obj);
console.log(objectEntries);
// Output: [["name", "John"], ["age", 30],["country", "USA"]]
```

Browser Object Model

not standardized, the objects & methods available vary between browsers

Window Object

The browser window that contains the current web page

- window.innerWidth
- window.innerHeight
- window.document
- window.localStorage

Location Object

The URL of the current web page and methods for working with it

- location.reload()
- location.assign(url)
- location.replace(url) - no back button to go to the original page

History Object

Access to the browser's session history

- history.back()
- history.forward()
- history.go(number) - Moves the user forward/backward the number of pages. For example, **history.go(-2)** would move the user back two pages.

Document Object Model

Represents the web page as a hierarchical tree-like structure, where each element in the tree corresponds to a part of the web page

Node Types

1. Element nodes - ex. `<div>`, `<p>`, and `<img>`.
2. Attribute nodes - ex. **src**, **href**, and **class**.
3. Text nodes - ex. text within a `<p>` element.
4. Comment nodes - comments (not displayed on the web page).

Selecting Elements

- getElementById()
- getElementsByClassName()
- getElementsByTagName()
- querySelector()
- querySelectorAll()

Intro to Event Handling

add event listeners to elements using the **addEventListener()** method

```
myElement.addEventListener('click', function() {
      // some code to run when the element is clicked
});
```

Manipulating Styles

Element.style.property

```
myElement.style.backgroundColor = 'red';
myElement.style.fontSize = '1.25rem';
```

Manipulating CSS Classes

- classList.add()
- classList.remove()
- classList.toggle()

```
const header = document.getElementById('header');
const paragraphs = document.getElementsByClassName('paragraph');
const asideElements = document.getElementsByTagName('aside');
const firstMatchingElement = document.querySelector('#footer');
const allMatchingElements = document.querySelectorAll('.p2');

// getElementById event listener
header.addEventListener('click', () => {
   header.style.backgroundColor = "lightseagreen";
});

// getElementsByClassName event listener
for (i = 0; i < paragraphs.length; i++) {
  paragraphs[i].addEventListener('click', (e) => {
    e.target.classList.toggle('purple');
  });
}

// getElementsByTagName event listener
for (i = 0; i < asideElements.length; i++) {
  asideElements[i].addEventListener('click', (e) => {
    e.target.classList.add('coral')
  });
}

// querySelector event listener
firstMatchingElement.addEventListener('click', () => {
  firstMatchingElement.style.backgroundColor = "lightseagreen";
});
```

```
// querySelectorAll event listener
for (i = 0; i < allMatchingElements.length; i++) {
  allMatchingElements[i].addEventListener('click', (e) => {
   e.target.style.textTransform = 'uppercase';
  });
}
```