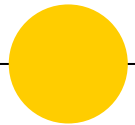
CS4220 Node.js & Angular.js

Cydney Auman
Albert Cervantes
CSULA



Intermediate Javascript

What is Javascript?

JavaScript is a cross-platform, lightweight, interpreted, prototype-based object-oriented language with first-class functions. It is a multi-paradigm programming language.

In true object-oriented programming - first a Class is created to serve as a “blueprint” and then objects are created based on this blueprint. Because Javascript is prototype-based object-oriented - a blueprint is never really created. In fact, objects inherit directly from other objects.

Javascript shares concepts from functional programming in that it supports first class functions. However, a key component to functional programming is data immutability. In JavaScript, numbers, strings and booleans are immutable. However, objects and arrays are mutable.

What is Javascript?

Number Methods

parseInt(*string*) function parses a string argument and returns an integer.

```
parseInt('5')      // 5
parseInt('5.5')    // 5
parseInt('hello') // NaN
```

parseFloat(*string*) method parses a string argument and returns a floating point number.

```
parseFloat('5')      // 5
parseFloat('5.5')    // 5.5
parseFloat('hello') // NaN
```

String Methods

toUpperCase() returns the calling string value converted to upper case

```
'hello'.toUpperCase()    // HELLO
```

toLowerCase() returns the calling string value converted to lower case

```
'HEY'.toLowerCase() // hey
```

trim() removes whitespace from both ends of a string

```
'   hello world '.trim()     // hello world
```

String Methods

split(*separator***)** splits a string into an array of strings by separating the string into substrings based on the separator.

```
const sentence = 'hello world'
sentence.split('')
// [ 'h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'
]

sentence.split(' ')
// [ 'hello', 'world' ]
```

String Methods

substring(*start, end***)** returns a subset of a string between one index an optional end index.

substr(*start, length***)** returns the characters in a string beginning at the specified location through the optional length.

```
const sentence = 'hello world'
sentence.substring(0, 5)
sentence.substr(0, 5)
// hello
```

Object Methods

Object.assign(*obj, obj*) is used to copy the values of all properties from one or more objects to a target object. It will return the target object.

```
const pet = {
    type: 'dog',
    breed: 'border collie',
    colors: ['black', 'white']
}
Object.assign(pet, { name: 'Fido' })
console.log(pet)
// { type: 'dog', breed: 'border collie', colors: [ 'black', 'white' ], name: 'Fido' }
```

Object Methods

Object.keys(*obj*) returns an array of a given object's properties.

```
const pet = {
    type: 'dog',
    breed: 'border collie',
    colors: ['black', 'white']
}

const keys = Object.keys(pet)
// [ 'type', 'breed', 'colors' ]
```

Array Methods

join() joins all elements of an array and returns a string.

```
const alpha = ['a', 'b', 'c']
alpha.join(',') // a,b,c
alpha.join(' | ') // a | b | c
```

Array Methods

pop() method removes the **last** element from an array and returns that element. This method changes the length of the array.

shift() method removes the **first** element from an array and returns that element. This method changes the length of the array.

unshift(*value***)** method adds one or more elements to the beginning of an array and returns the new length of the array.

```
const alpha = ['a', 'b', 'c']

alpha.pop()

console.log(alpha) // ['a', 'b']

alpha.shift()

console.log(alpha) // ['b']

alpha.unshift('a')

console.log(alpha) //['a', 'b']
```

Array Methods

forEach(*function***)** executes a provided function once for each array element.

map(*function***)** creates a new array with the results of calling a provided function on every element in this array.

```
const alpha = ['a', 'b', 'c']
alpha.forEach((letter, index) => {
  console.log(letter, index)
})

const mappedAlpha = alpha.map((letter, index) => {
    return { [letter]: index }
})
console.log(mappedAlpha) // [ { a: 0 }, { b: 1 }, { c: 2 } ]
```

Array Methods

slice(*begin, end***)** returns a shallow copy of a portion of an array into a new array object. The original array will not be modified.

```
const list = [ 'a', 'b', 'c', 'd']
const sliced = list.slice(1, 3)

console.log(list) // [ 'a', 'b', 'c', 'd']
console.log(sliced) // [ 'b', 'c' ]
```

Array Methods

splice(*start, removeCount, items***)** changes the contents of an array by removing existing elements and/or adding new elements (optionally).

```
const list = [ 'a', 'b', 'c', 'd']
const spliced = list.splice(1, 3)

console.log(list) // [ 'a' ]
console.log(spliced) // [ 'b', 'c', 'd' ]

list.splice(1, 0, 'b', 'c', 'd') // [ 'a', 'b', 'c', 'd']
```

Destructuring

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to extract data from arrays or objects into distinct variables.

```
const arr = [ 1, 2, 3, 4 ]
const [ a, b ] = arr
console.log(a) // 1
console.log(b) // 2
```

```
const fido = {
    type: 'dog',
    breed: 'border collie',
    colors: ['black', 'white']
}
const { type, breed, colors } = fido
console.log(type) // dog
```

Default Values

A variable can be assigned a default, in the case that the value pulled from the array or object is undefined.

```
const arr = [ 1, 2 ]
const [ a, b, c = 0 ] = arr
console.log(a) // 1
console.log(b) // 2
console.log(c) // 0
```

```
const fido = {
    breed: 'border collie',
    colors: ['black', 'white']
}
const { type = 'cat' , breed, colors } = fido
console.log(type) // cat
```

Template Literals

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.

Template literals are enclosed by the back-tick (` `) instead of double or single quotes. Template literals can contain placeholders. These are indicated by the dollar sign and curly braces (${expression}).

```
const food = 'sandwiches'
console.log(`i like ${food}`)
// i like sandwiches
```

ES6 Classes

Nearly all objects in JavaScript are instances of Object - a typical object inherits properties and methods from Object.prototype.

JavaScript classes introduced in ES6 are syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is not introducing a new object-oriented inheritance model to JavaScript.

ES6 Classes

To declare a class, you use the **class** keyword.

```
class Polygon {

}
```

The **constructor** method is a special method for creating and initializing an object created with a class.

```
class Polygon {
    constructor(height, width) {
         this.height = height
         this.width = width
    }
}
```

ES6 Classes

```
class Polygon {
     constructor(height, width) {
          this.height = height
          this.width = width
     }

     calcArea() {
          return this.height * this.width
     }
}

const square = new Polygon(10, 10)
console.log(square.calcArea()) // 100
```

ES6 Classes

The **get** syntax binds an object property to a function that will be called when that property is looked up.

```
class Polygon {
     constructor(height, width) {
          this.height = height
          this.width = width

     }

     get area() {
          return this.calcArea()
     }

     calcArea() {
          return this.height * this.width
     }
}
const square = new Polygon(10, 10)
console.log(square.area) // 100
```

References and Reading

Mozilla Developer Network (Methods and Properties on Numbers, Strings, Array and Objects)

-- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

Eloquent Javascript

-- http://eloquentjavascript.net/

-- Chapters 4, 5, 6

Destructuring

-- https://leanpub.com/understandinges6/read#leanpub-auto-destructuring-for-easier-data-access

Template Literals

-- https://leanpub.com/understandinges6/read#leanpub-auto-template-literals

Classes

-- https://leanpub.com/understandinges6/read#leanpub-auto-class-declarations