COMP 2015

Introduction to JavaScript and jQuery

Contents

Classes and Objects

Data Types

JavaScript has six primitive data types (String, Number, Boolean, Null, Symbol, and Undefined), and one complex data type: Object.

Throughout the course we have used objects from several of JavaScript's Web APIs, e.g. document, event, and window

Structure of Objects

Objects have properties (values which define the state of an object) and methods (behaviours or actions that can an object perform)

An example of an object property we have used in the course is the .length property of Arrays

```
var allImages = document.images;

// allImages is now a reference to an Array object, so we can use the
// .length property of the Array class

alert(allImages.length); // returns the number of images in the array
```

Examples of object methods we have used are window.alert() and console.log(), both of which perform a type of action

Defining Custom Objects

JavaScript provides many objects for us to use, however we can also define our own custom objects, properties and methods. There are several ways to create these custom objects, but the simplest and most common is the *Object Literal*

Object Literal

Object Literals are created by assigning comma separated name/value pairs inside of curly braces to a variable. The name/value pairs can be properties or methods.

```
var dog = {
    name: 'Charlie',
    age: 12,
    speak: function() { alert('Woof!'); }
}

alert(dog.name);   // alerts 'Charlie'
dog.speak();        // alerts 'Woof!'
```

Object Constructor

We can also use the *Object Constructor* to create an object

```
var dog = new Object();
```

then assign properties and methods as needed

```
dog.name = 'Charlie';
dog.age = 12;
dog.sleep = function() {
    alert('zzzzzzzz');
}
```

Constructor Pattern

In the Constructor Pattern, we assign name/value pairs inside of a method (i.e. function), which is then called to create our object. Note that this is the first means we have seen for creating objects that is **reusable**.

We can also add properties and methods to a specific object after it has been created.

```
function Dog(dogName, dogAge, dogBreed) {
    this.name = dogName;
    this.age = dogAge;
    this.breed = dogBreed;
    this.speak = function() {
        alert(this.dog + ' says woof!'); }
}

var dog1 = new Dog('Charlie', 12, 'ridgeback');
var dog2 = new Dog('Finn', 8, 'mutt');

alert(dog1.name);   // alerts 'Charlie'
dog2.speak();       // alerts 'Finn says woof!'

dog2.weight = '100lbs';    // adding a property to dog2
```

Prototype Pattern

The Prototype Pattern uses a constructor function similar to the Constructor pattern, but allows us to modify the implementation of every object created from the function, by using the *prototype* keyword.

```
function Dog(dogName) {
    this.name = dogName
}

Dog.prototype.numberOfLegs = 4;

var dog1 = new Dog('Charlie');
alert(dog1.name + ' has ' + dog1.numberOfLegs + ' legs');
```

Inheritance

The Prototype Pattern also allows us to define inheritance, where one object inherits properties and/or methods from another class.

Inheritance allows us to create an ‘is a’ relationship between objects, e.g. a dog ‘is a’ four legged animal.

```
// All animals have an age
function Animal(age) {
    this.age = age;
}

// All animals can bite
Animal.prototype.bite = function(target) {
    return ‘ bit ‘ + target + ‘!’;
}

// All dogs have a name
function Dog(name) {
    this.name = name;
}

// Create inheritance
Dog.prototype = new Animal();  // now all dogs also have an age, and can bite

var dog1 = new Dog(‘Charlie’);

alert(dog1.name + dog1.bite("Jason"));  // returns ‘Charlie bit Jason!’
```

Classes

A Class is like a blueprint for an object. In the same way that an actual blueprint can be used to create many houses/buildings, a class can be used to create many objects.

Two things are needed to create a class; the keyword *class*, and a function named *constructor* that defines the properties that are set when an object is created.

```
class Dog {
    constructor(dogName, dogAge, dogBreed) {
        this.name = dogName;
        this.age = dogAge;
        this.breed = dogBreed;
    }

    speak() {
        alert(this.name + ' says woof!');
    }
}

var dog = new Dog('Charlie');
dog.speak();   // alerts 'Charlie says woof!'
```

Accessing an Object's Properties

Objects have two types of properties: *own* and *inherited*. *Own* properties are the properties that are defined on the object, and *inherited* properties are the properties that are inherited from the object's Prototype or parent class (e.g. the 'age' property of our Animal class would be an inherited property of the Dog class).

We can enumerate all own and inherited properties using a for loop and the *in* operator. The *in* operator returns true if a property has been defined; false otherwise.

```
var dog1 = {
    breed: 'ridgeback',
    age: 12
}

for (var value in dog1) {
    console.log(dog1[value]);
}

/*
prints
   ridgeback
   12
to the console
*/
```

```
var dog1 = {
    breed: 'ridgeback',
    age: 12
}

for (var name in dog1) {
    console.log(name);
}

/*
prints
   breed
   age
to the console
*/
```

hasOwnProperty()

The hasOwnProperty method returns true only if a property is an own property (i.e. not inherited).

```
function Animal(age) {
    this.age = age;
}

function Dog(name) {
    this.name = name;
}

// Create inheritance
Dog.prototype = new Animal();  // now dogs have an age

var dog1 = new Dog('Charlie');

console.log(dog1.hasOwnProperty('name'));  // returns true

console.log(dog1.hasOwnProperty('age'));   // returns false,
                                           // age is inherited from Animal
```

Final Exam

Due 11:59pm the final day of class.

Download it from our course section of COMP2015 on D2L http://learn.bcit.ca (COMP2015 > Content > Final)