

CS 22A

JavaScript for Programmers

© 2019, Dr. Baba Kofi Weusijana (Bah-bah Co-fee
Way-ou-see-jah-nah)
All Rights Reserved



Today (Day10)

- ES5 Objects: Literals & Properties
- Assignment 2: A Basic Calculator
 - The eval function
 - Jasmine for Behavior Driven Design

Making & Using JS Objects

- You have seen some of this before (such as in Java) but **pay attention**, there are subtle and important JavaScript differences!

Defining ES5 Objects

- To begin using JavaScript objects, you need to find out what they are and how they can be useful to you in your scripts.
- First take a look at what JavaScript objects are.
- Objects are part of the programming paradigm known as **object-oriented programming**, sometimes shortened to **OOP**.
- JavaScript is not a typical object-oriented language (because it uses prototype inheritance)
 - This enables the programmer to take advantage of some (but not all) of the good things that come with object-orientation.
- For Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

What is an Object?

- An Object is a collection of *characteristics* (properties) & some of those are *actions or functions* (methods).

- Object Syntax

- The syntax for using an object is:

`object.property`

`object.method()`

- A **string** is an object in JavaScript and has several *properties* and *methods*:
`var someString = new String("This is a string of characters!");`

`var x = someString.length; // "length" is a property`

`var y = someString.toUpperCase(); // toUpperCase() is a method`

What is an Object?

- In JavaScript almost everything (except undefined and null symbols) are objects.
 - o Booleans can be objects or primitive data treated as objects
 - o Numbers can be objects or primitive data treated as objects
 - o Strings are also objects or primitive data treated as objects
 - o Dates are **always** objects
 - o Math and Regular Expressions are **always** objects
 - o Arrays are **always** objects
 - o **Functions are always objects**

Built-In Objects

- JavaScript has several objects built into the language, like **String**, **Math**, **Date**, and **Array**.
- The Math object is a collection of methods and properties for performing mathematical operations like min(), max(), sin(), cos(), etc.
- The Date object is a collection of methods for working with dates and time.
- The Array object allows programmers to create collections of data, and so on.

The **new** Operator/Keyword

- Objects cannot simply be typed into your JavaScript programs. They must first be created.
- We can use the "new" operator/keyword to create a new instance of an object.
- **The new operator creates a usable copy of an existing object and assigns the name you want to it.**
- The generic syntax is:

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

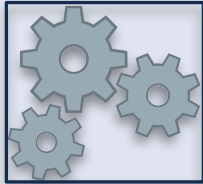
- Since objects are made up of functions (methods) and characteristics (properties), the newly created myObject in this case has all the same methods and properties of the original Object.

Why Objects Are Useful

- Objects are useful because they give you another way to conceptually **organize** things within a script.
- Rather than having a bunch of similar *variables* that are out there on their own, you can group them together under an object.

JavaScript

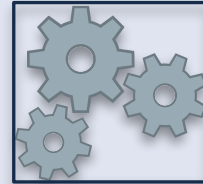
Object



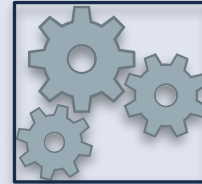
Object



Object



Object



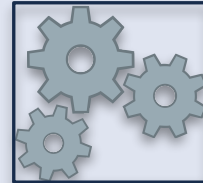
Object



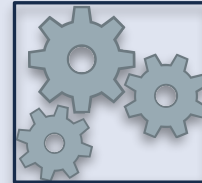
Object



Object



Object



Restaurant

Manager



Cook



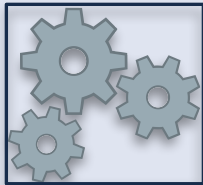
Prep



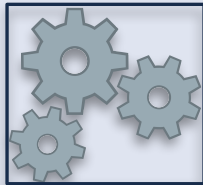
Dishwasher



Server



Busser



Front Desk



Bartender



Creating Objects

- In JavaScript, an object can be created in a number of ways.
- A primary way to create an object is by using two curly braces, like so:

```
var myObject = { };
```

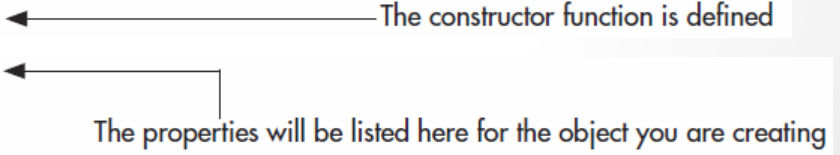
 - This is called an object initializer or object literal.
- Objects can also be created with the new keyword, with a constructor function like so:

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

Objects: Constructor Function

- A **constructor function** allows you to build an object using the same basic syntax as a *regular* function. The only difference is the code you place inside of the function and how you access its contents.
- For example, to create a **Car** object, you would create a *constructor function* named **Car()** and then add your properties within the function.
- The following example shows an outline of **the Car()** function:

```
var Car = function () {  
    //properties go here  
};
```



The diagram consists of two arrows pointing from text boxes to the code. The first arrow points from the text 'The constructor function is defined' to the opening curly brace of the function definition. The second arrow points from the text 'The properties will be listed here for the object you are creating' to the comment '//properties go here'.

← The constructor function is defined

← The properties will be listed here for the object you are creating

Constructor Function

- To complete the preceding function, you need to add your properties to the function.
- Recall that we wanted to create an object named **Car** with the properties of **seats**, **engine**, and **soundSystem**.
- The following code shows how this is done:

```
var Car = function (seats, engine, soundSystem) {  
    this.seats      = seats;  
    this.engine     = engine;  
    this.soundSystem = soundSystem;  
};
```

← The function takes in three parameters

The parameter values are assigned to the properties of the object

Constructor Function

- You must use the constructor function with the **new** keyword to create what is called an **instance** of the object in order to use it, because a constructor function creates *only the structure* of an object, *not* a usable instance of an object.
- The use of the new keyword to create an instance of your Car object is shown in the following code:

```
var myCar = new Car("leather", "V-6", "Radio with  
6 CD Player");
```

Constructor Function

```
var Car = function (seats, engine, soundSystem) {  
    this.seats      = seats;  
    this.engine     = engine;  
    this.soundSystem = soundSystem;  
};
```

```
var myCar = new Car("leather", "V-6", "Radio with 6 CD  
Player");
```

- You can now access the **myCar** instance of the **Car** object.
- If you want to know what type of engine the **myCar** has, you can access it with the **dot operator**:

```
var engineType = myCar.engine;
```

This assigns the value of the engine property of the **myCar** instance of the **Car** object to the variable **engineType**. Since you sent *V-6* as the engine parameter to the *constructor*

- *function*, the **engineType** variable is assigned a value of *V-6*.

Constructor Function

- Putting It All Together To help you visualize this process, it's time to put all these parts together so that you can see how it works. The following code combines all the code of the previous examples to make things easier to see:

```
var Car = function (seats, engine, soundSystem) {  
    this.seats = seats;  
    this.engine = engine;  
    this.soundSystem = soundSystem;  
};
```

The constructor function

```
var myCar = new Car("leather", "V-6", "Radio with  
CD Player");  
var engineType = myCar.engine;
```

An instance of the object is created, sending parameters to be used as property values

A property of the new instance of the object is assigned to an independent variable

Constructor Function

- In order to see how an instance of an object works, you need to add another instance of the Car object to your code. The following code uses two instances of the Car object, one named **myCar** and a new one named **wifesCar**:

```
var Car = function (seats, engine, soundSystem) {  
    this.seats = seats;  
    this.engine = engine;  
    this.soundSystem = soundSystem;  
};  
  
var myCar = new Car("leather", "V-6", "Radio with 6 CD Player");  
var wifesCar = new Car("cloth", "V-8", "Radio with 1 CD Player  
and MP3");  
  
var engineMine = myCar.engine;  
var engineWifes = wifesCar.engine;
```

Constructor Function

- By assigning the property values of the different instances of the Car object to variables, you could now write out the features you would like to have in a **custom car** that combines features from each type of Car.
- For example, take a look at the following code, which writes out the features you might like in a **custom car**:

```
var Car = function (seats,engine,soundSystem) {  
  this.seats = seats;  
  this.engine = engine;  
  this.soundSystem = soundSystem;  
};
```

```
var myCar = new Car("leather","V-6","Radio with 6 CD Player");  
var wifesCar = new Car("cloth", "V-8", "Radio with 1 CD Player and MP3");  
var customEngine = myCar.engine;  
var customSeats = myCar.seats;  
var customSoundSystem = wifesCar.soundSystem;
```

```
$("#content").append("I want a car with " + customSeats + " seats.<br />");  
$("#content").append("It also needs a " + customEngine + " engine.<br />");  
$("#content").append("Oh, and I would like a " + customSoundSystem + " also.");
```

Constructor Function

```
var Car = function (seats, engine, soundSystem) {  
  this.seats = seats;  
  this.engine = engine;  
  this.soundSystem = soundSystem;  
};  
  
var myCar = new Car("leather", "V-6", "Radio with 6 CD Player");  
var wifesCar = new Car("cloth", "V-8", "Radio with 1 CD Player and MP3");  
  
var customCar = new Car(myCar.seats, myCar.engine, wifesCar.soundSystem);  
  
$("#content").append("I want a car with " + customCar.seats + " seats.<br />");  
$("#content").append("It also needs a " + customCar.engine + " engine.<br />");  
$("#content").append("Oh, and I would like a " + customCar.soundSystem + "  
also.");
```

Object Initializers (Literals)

- An **object initializer** (or **object literal**) is a little bit shorter than a *constructor function*.
- The following is the **syntax** of an *object initializer*:
 - **var objectName = {property:value} ;**
- **All** JavaScript objects are key:value hash maps (just some don't have any properties in them). In computer science, a **hash table**, **hash map**, **associative array**, **map**, **symbol table**, or **dictionary** is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears **just once** in the collection.
 - For reference: https://en.wikipedia.org/wiki/Associative_array

Object Initializers

Example

- You can create a **myCar** object by using the *initializer* method.
- You want the object name to be myCar, and you will have *three* sets of *properties* and *values*.
- The following code shows how to create the object by using the object initializer method:

```
var myCar = {seats:"leather", engine:"V-6",  
soundSystem:"Radio with 6 CD Player"};
```

- Since there is no need to create an instance of the object, you can use its properties just as you did before, and assign them to variables or write them to the page.
- For instance, the property of **myCar.seats** would be *leather*.

Object Initializers

Example

- If you want the wifesCar object back as well, you can use another initializer, as shown in the following example code:

```
var myCar = {seats:"leather", engine:"V-6",  
soundSystem:"Radio with 6 CD Player"};  
var wifesCar = {seats:"cloth", engine:"V-8",  
soundSystem:"Radio with 1 CD Player and  
MP3"};
```

- But when you make literal objects, and thus are not using **new** and a constructor, **you don't get to take advantage of object inheritance.**

Assignment 2:

A Basic Calculator

- Our first group assignment
- Once assigned to a group, contact your team members via Canvas Inbox
- Share with group **what you are good at & what are your challenges**
 - Helps set roles appropriately & encourages empathy
 - If a group member doesn't get in touch with all of above info **by 9AM the morning of Day13, contact your professor**
- **Plan to meet** at least weekly for at least 2 hours (such as via Conferences teleconferencing built into Canvas)
 - Remember, there may be a **revision**
- Read the whole assignment including all links (watch the demo video) **before** you start working!

Assignment 2: Part 3

- A way to get input elements into an array:

- There are other ways

```
// Binding onclick events
```

```
var inputs =  
document.getElementsByTagName("input");
```

- Then you need code that ignores elements in `inputs` that do **not** have a `.type` equal to `"button"`

The “evil” eval function

- Using it indicates you **might not** know what you are doing.
- It could adversely effect code clarity, debug-ability, & performance!
- When you need to use a variable to access a property name, you can use bracket notation.

```
guitar.strings.smallGauge; // returns 9
```

```
var x = "smallGuage";
```

```
guitar["strings"][x]; // also returns 9
```

- <https://www.nczonline.net/blog/2013/06/25/eval-isnt-evil-just-misunderstood/>
-

Jasmine for Behavior Driven Design (BDD)

- Jasmine tutorial:
<http://code.tutsplus.com/tutorials/testing-your-javascript-with-jasmine--net-21229>
- More on BDD:
<http://www.slideshare.net/babakofi/bdd-behavior-driven-development>

Demo of A2

- Make sure all 3 versions of A2 act the **same** like this demo:

<http://edutek.net/foothill/cs/22a/A2/>

To Do

- Proceed through the modules
 - Take "Assignment 2 Team Preferences" survey so you can get assigned to a team