Week 06 – Object Oriented JavaScript

**JS OBJECTS:**

\*\*In JavaScript, almost "everything" is an object.

\*\*All JavaScript values, except primitives, are objects.

* Booleans can be objects (if defined with the **new** keyword)
* Numbers can be objects (if defined with the **new** keyword)
* Strings can be objects (if defined with the **new** keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

JS PRIMITIVES:

\*\*A **primitive value** is a value that has no properties or methods.

\*\*A **primitive data type** is data that has a primitive value.

* JavaScript defines 5 types of primitive data types:
* String, number, Boolean, null, undefined

\*\*Primitive values are immutable (they are hardcoded and therefore cannot be changed).

OBJECTS ARE VARIBALES CONAINING VARIABLES:

\*\*Variables 🡪 contain single values

* Objects
* Objects are Variables
* Containing many values 🡪 written in name : value pairs
* Objects are collections of named values 🡪 called properties
* Objects are containers for named values 🡪 called properties & methods
* \*\*Objects written as name value pairs are similar to:

- Associative arrays in PHP

- Dictionaries in Python

- Hash tables in C

- Hash maps in Java

- Hashes in Ruby and Perl

OBJECT PROPERTIES: (See Below)

OBJECT METHODS: (See Below)

**CREATING A JS OBJECT:**

* Define and create a single object, using an object literal
* Define and create a single object, with the keyword new
* Define an object constructor, and then create objects of the constructed type.
* OBJECT LITERAL: \*\*\*
* Easiest way to create an object
* Define & create an object in one statement
* A list of name : value pairs inside curly braces

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

* KEYWORD NEW:
* var person = new Object();  
  person.firstName = "John";  
  person.lastName = "Doe";  
  person.age = 50;  
  person.eyeColor = "blue";
* OBJECT CONSTRUCTOR (Function): \*\*
* Creates an ‘Object Type’ that can be used to create many objects of one type

function person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eye;  
}

\*The function ‘person’ is an object constructor

\*Once you create an object constructor, you can create new objects of the same type

var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

KEYWORD **this:**

* In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code.
* It’s value, when used in a function, is the object that "owns" the function.
* It’s value, when used in an object, is the object itself.

\*\*JS Objects are mutable – addressed by reference, not by value

JS OBJECT PROPERTIES:

* \*Named values
* \*Can be both primitive values, other objects, and functions
* \*The values associated w/ a JS Object
* Adding: 🡪 person.name = “John”;
* Deleting: 🡪 Delete person.name (Other specifications about Delete – see W3)
* Accessing: ObjectName.Property
* Property Attributes:

\*\*All properties have a name & a value

* Value: one of the properties attributes
* Other Attributes: enumerable, configurable, & writable

\*Define how the property can be accessed

\*\*In JS, all attributes can be read, but only value can be changed

* Prototype Properties: JS Objects inherit properties of their prototype

JS OBJECT METHODS:

* \*Actions that can be performed on Objects
* \*An **Object Method** is an Object Property containing a **Function Definition**
* \*Methods are Functions stored as Object Properties
* Syntax: method name: function() { code }
* Accessing: ObjectName.MethodName();

\*JS has built-in methods also

* Adding New Methods:
* Done inside the Constructor Function 🡪 ???W3

JS OBJECT PROTOTYPES:

* \*Every JS Object has a Prototype; the Prototype is also an object
* \*All JS Objects inherit their Properties & Methods from their Prototype
* Object.Prototype: all JS Objects inherit from this; at the top of the Prototype Chain
* Creating a Prototype:
* Standard way is to use an Object Constructor Function

(Basically same as using a Constructor ???)

**JS FUNCTIONS (DEFINITIONS): (Imported from Week 04 – Unobtrusive JS Notes)**

\*\*Defined using the ‘function’ keyword \*\*Can be used as values and in expressions

Function Declaration:

* Not executed immediately, but when invoked
* Not an executable statement 🡪 no ‘ ; ’

function myFunction(a, b) { return a \* b; }

Function Expression:

* Can be stored in a variable & that variable can be used as a function
* Is an executable statement 🡪 ends with ‘ ; ’

var x = function(a, b) {return a \* b};  
var z = x(4, 3);

* Functions stored in variables do **not need a function name** (anonymous)
* They are always invoked (called) using the variable name

Function() Constructor:

* Functions can also be defined using the built-in JS function constructor

var myFunction = new Function("a", "b", "return a \* b");

* Unnecessary to use the function constructor; same as writing:

var myFunction = function (a, b) {return a \* b}; //Usually can avoid ‘new’ in JS

Function Hoisting:

* JS’s default behavior of moving declarations to top of current scope; variable & function
* Functions defined using an expression are not hoisted.

Self-Invoking Functions:

* Function expression can be made self-invoking – executed without being called
* Will execute automatically if followed by ‘()’
* Add ‘( )’ around function to indicate that its an expression

Functions Are Objects:

\*\*typeof returns functions as ‘function’ BUT they’re best described as Objects

* JS Functions have both:
* Properties: ex 🡪 arguments.length
* Methods: ex 🡪 toString()

\*A Function defined as the property of an object 🡪 method to the object

\*A Function designed to create new objects 🡪 object constructor

**JS FUNCTION PARAMETERS:**

\*\*Parameters: names listed in function definition

\*\*Arguments: real values passed to & received by the function

* Rules: JS Functions do not:
* Specify data types for parameters
* Perform type checking on the passed arguments OR check number of arguments received
* Defaults:
* If a function is called with missing arguments 🡪 missing values set to undefined
* Can assign a default value for the parameter in function

function myFunction(x, y) {  
    if (y === undefined) {  
          y = 0;  
    }   
 }

Arguments Object:

* Built-in object that contains an array of the arguments used when the function was called
* Allows you to use a function to find: ex – sum of all input values/arguments

\*\*Arguments:

* Passed by Value
* Changes to arguments are not visible (reflected) outside the function

\*\*Objects:

* Passed by Reference
* Changes to Object Properties are not visible (reflected) outside the function

**JS FUNCTION INVOCATION:**

**Invoking a JS Function:**

* Function is not executed when declared; must be invoked (called)

**Invoking a Function as a Function:**

\*\*In JS there is always a default Global Object 🡪 the HTML page itself

\*\*In a web browser the Page Object is the Browser Window

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2); / window.myFunction(10,2);

* This Function automatically becomes a Window Function;

myFunction() & window.myFunction are the same

The **this** Keyword:

\*\*In JS the thing called ‘this’ is the object that owns the current code

**The Global Object:**

* \*\* When a function is called without an owner object, the value of **this** becomes the global object
* \*\*In a web browser the global object is the browser window

function myFunction() {  
    return this;  
} //This Function returns the Window Object as the value of **this**  
myFunction();   **\*\***Using the Window Object as a variable can crash the program

**Invoking a Function as a Method:**

* \*\*In JS you can define functions as Object Methods

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName(); //myObject owns the fullName Function

\*\*Invoking a Function as an Object Method causes the value of **this** to be the object itself

**Invoking a Function with a Function Constructor:**

* \*\*If a function invocation is preceded with the **new** keyword, it is a constructor invocation

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John", "Doe");  
x.firstName;

\*\***this** doesn’t have a value in a constructor 🡪 the value of **this** will be the new object created when the function is invoked

Object Oriented JavaScript – Video Lecture Notes

**\*\*Everything in JS is an Object**

var x = new Object();

* Don’t need to use new 🡪 using ‘{ }’ makes it obvious

var x = {};

x.Name = “John”;

* Creates a String Object on the Name property of x
* The value of Name is a String Object

x.ID = 123;

* Creates an Integer Object on the ID property of x
* The value of ID is an Integer Object

**\*\*It’s Important to Scope Things**

* Instead of declaring a bunch of global variables, scope them in a namespace

var Main = {}; 🡪 Scopes everything on the Main object (namespace)

Main.X = {};

Main.X.Name = “John”;

Main.X.ID = 123;

**\*\*Changing Object & Variable names to make more sense**

var Main = {};

Main.Marie = {};

Main.Marie.Name = “Marie”;

Main.Marie.age = 20;

**\*\*Creating Objects using Object Literals:**

* Define properties inline
* Known as Literal Object Instantiation or Inline Object Instantiation
* JS object notation - JSON

\*comma delimited list of name value pairs where name & value separated by “:”

Main.Marie =

{

Name: “Marie”;

Age: 20;

}

**\*\*Using a** **Constructor:**

* The piece of the class that defines how you create Objects

Main.Person = function(firstName, lastName, age) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

}

🡪 Creates a person objects that is a function

🡪 Creates a person, & each person will have the same variables

* By creating this constructor:
* You’ve created reusable code, (can reuse the function to create other Person Objects)
* That allows you to invoke the Person Function, which will create the 3 properties specified w/ the 3 values passed in the parameters

Main.Nate = new Main.Person(“Nate”, “Kelling”, 25);

**\*\*Objects can point to the same reference**:

Main.OldestSibling = Main.Nate;

* w/ assignment operator, the item on the right is an already existing object:
* It does not create a new object
* It does not create a copy of the object
* Simply points both objects at the same reference

\*Points the variable OldestSibling to the memory address of Main.Nate

**PROTOTYPE:**

\*\*Methods

\*\*Class Prototype Methods

\*\*Methods you’re attaching to a Constructor or an Object Type

Main.Person = function(firstName, lastName, age) {

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

this.GetFullName = function() {

Return this.firstName + “ “ + this.lastName;

} 🡪 usually don’t want to declare methods in constructor. Instead use prototype attribute

}

**Prototype Attribute:**

\*Makes JS close to being OO w/out having abstract classes polymorphism, interfaces, etc.

* Allows you to declare/add methods to the ‘Person Class’ outside of the constructor

Main.Person.prototype.GetFullName = function()

{

Return this.firstName + “ “ + this.lastName;

}

Main.Person.prototype.SetFirstName = function(name)

{

this.firstName = name;

}