JavaScript and DOM – Class notes – *Creating objects*

JavaScript allows the creation of custom objects using special functions called “Object functions” as well as object literal. Once objects are created, they can have their own functions as well as properties. Functions that belong to objects are called methods and variables belonging to objects are called properties of the object.

Refer to this site to learn more about objects in JavaScript

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects>

# creating object using object literal

Object literal consist of ***name:value*** pair where the “name” is the variable name of property of an object and value is the value of the property. It is important to note that object literals can saved in variables. The name value pairs are surrounded by {} and each name value pair is separated by comma.

|  |
| --- |
| HTML |
|  |
| JavaScript : *creating an object called car* |
| //create a car object  var car = { year:1979, make:”Toyota” };  //access the car object’s properties  console.log(car.year);  console.log(car.make);  //properties can be accessed using array notation as well. Note that the year property is in quotes when used in the array notation  Car[“year”]  to set the values of the object  car.year = 1980;  or  car[“year”] = 1980 |

Objects created using object literal can have functions (methods) in them as well. When functions are added as part of the object that is being created using object literal, the function is also declared using literal function notation.

|  |
| --- |
| HTML |
|  |
| JavaScript : *creating an object called car* |
| //create a car object  var car = { year:1979, make:”Toyota” , **speed : function() { return 180; }** }; |

# creating object using a constructorfunction

Constructor functions are just like regular functions that can be used to define objects. Using this technique you can create a function that can take parameters and set those parameters as values for the object properties. It is a strong convention to use capitalized name for constructor functions. Contractor functions can be thought of as the blueprint of an object. In order to create an object, we need to use the “new” keyword to create an instance of that object that is defined by the constructor function

|  |
| --- |
| HTML |
|  |
| JavaScript : *to access the div1 div element and change it’s style attribute and set the background color* |
| //define the constructor function  Function Car (yr,mk)  {  this.year = yr;  this.make = make;  this.mySpeed = speed // notice this this references the speed function outside of the object and we are not calling speed function using (). We are simply typing the function name speed. This is called function copying.  }  //create an instance of the Car object and save it in the variable c  var c = new Car (1980, “Toyota”);  //create an instance of the Car object and save it in the variable c  var c2 = new Car (1990, “Honda”);  console.log(c.year);  console.log(c1.year);  function speed()  {  Return 180;  } |
| Explanation:  In the example above, the function Car take two parameters (yr and make). When a new instance is created using the “new” keyword (e.g. when we create c and c2 as car objects) we pass in the value for that specific instance. In the body of the Car object the “this” keyword refers to the instance that will be created from the constructor function. This is the reason c and c2 instances of Car object have different values based on what was passed in when the constructor function was called, even through they both called the same functions but with different parameters. |

# Creating objects using the object object (this is not a typo)

In javaScript we can also create an object by creating an instance of the object called Object (capital O).

|  |
| --- |
| HTML |
|  |
| JavaScript : *to access the link element and also add another link element* |
| var myCar = new Object();  myCar.make = “Toyota”;  myCar.year = 1980;  myCar.mySped = speed;  function speed()  {  Return 180;  } |