Object Literals, Constructors and Object Oriented JavaScript | JavaScript Tutorial In Hindi #27

Before getting into the detail of JavaScript OOP concepts, let us first understand what Object-oriented programming (OOP) is. The basic idea of object-oriented programming is that objects are used to model real world things that we want to represent inside our programs. We can also describe OOP as the OOP provides a simple way to access functionality that would otherwise be hard or impossible to make use of.

What are objects?

Objects can contain related code and data, representing information about the thing we are trying to model, and functionality that we want it to have. Object data is stored neatly inside an object package, making it easy to structure and access.
JavaScript is a powerful programming language that supports Object-Oriented Programming (OOP). In JavaScript, objects created using object literal are **singletons**. This means when we made a change in an object, it affects the object entire the script. Whereas if an object is created using the constructor function, then the change will not affect the object throughout the script. In this tutorial,

```
// constructor function
function MyCar(){};
// literal notation
var myCar = {};
```

Objects created using object literal:-

Since these are **singletons**, a change to an object affects the object's entire script. A change in one instance will affect all the instances of the object. Object literal is a comma-separated list of name-value pairs inside the curly braces. The object literals encapsulate the data. This minimizes the use of global variables, which can cause problems when combining the code.
Following is an example object literal:

```
var myCar = {
    make: 'Ford',
    model: 'Mustang',
    year: 2000
};
```

However, if we have to create multiple instances of a structure and perform operations based on predefined values, then we should use a function constructor.

Objects created using the constructor function:-

The object that is defined with a **function constructor lets** us have multiple instances of that object. If the change is made to one instance, it will not affect other instances. As we know that the constructor is essentially a function that acts as a blueprint for creating objects. A convention for declaring constructors is always to capitalize its function name.

```
function MyCar (make,model,year){
 this.make = make;
 this.model= model;
 this.year = year;
 }
```

Now we have our constructor function called **MyCar**. We can **"construct"** our object instances based on that blueprint. Let us create a MyCar instance object called "car1"

```
// creating objects
let car1 = new MyCar("Ford", "Mustang", 2000);
```

"new" keyword:-

To create a new object instance, we use the **"new"** keyword to create an object based on a constructor. Adding the "new" keyword is a crucial step when creating an object from a constructor. The new keyword creates a new empty object. It binds property or function, which is declared with this keyword to the new object.

Conclusion:-

In this tutorial, we have learned about object literals and constructor functions. The main difference between them is that if we need multiple instances of our object, the object that is defined with a constructor lets us have multiple instances of that object. In comparison, the object literals are basically singletons with variables that are all public.

Website.html code as described/written in the video

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <div class="container">
        <h1 id="heading" class='yourhead rhia is'> Welcome to Code With Harry</h1>
        <div id="myfirst" class="child red good" id="first">child 1

            <ul class="this" id='myul'>
                <li class="childul" id='fui'>this</li>
                <li class="childul">is</li>
                <li class="childul">a</li>
                <li class="childul">list </li>
                <li class="childul" id='lui'>of my dreams</li>
        </div>
        <div class="child">child 2</div>
        <div class="child red">child 3</div>
        <div class="child">child 4</div>
        <form action="none.html" method="post">
            <a href="//codewithharry.com">Go to Code With Harry</a>
            <br>
            <br>
            Search this website: <input type="text" name="Hello" id="">
            <button id="btn">Submit form</button>
            <!-- <input type="button" id='btn' value="submit"> -->
        </form>
    </div>
    <br>
    <div class="no">this is a dummy div1</div>
    <div class="no">this is a dummy div2</div>
    <div class="no">this is a dummy div3</div>
</body>
<!-- <script src="js/tut12.js"></script> -->
<!-- <script src="js/tut14.js"></script> -->
```

```
<!-- <script src="js/tut15.js"></script> -->
<!-- <script src="js/tut16.js"></script> -->
<!-- <script src="js/tut17.js"></script> -->
<!-- <script src="js/tut18.js"></script> -->
<!-- <script src="js/tut20.js"></script> -->
<!-- <script src="js/tut21.js"></script> -->
<script src="js/tut23.js"></script>
</html>
```

JavaScript code as described/written in the video

Copy

```
console.log("This is tut 27");

// Object Literal for creating objects
let car = {
  name: "Maruti 800",
  topSpeed: 89,
  run: function() {
    console.log("car is running");
  }
};
// we have already seen constructors like these:
// new Date();

// Creating a constructor for cars
function GeneralCar(givenName, givenSpeed) {
  this.name = givenName;
  this.topSpeed = givenSpeed;
  this.run = function() {
    console.log(`${this.name} Is Running`);
  };
  this.analyze = function() {
    console.log(
      `This car is slower by ${200 - this.topSpeed} Km/H than Mercedes`
    );
  };
}
car1 = new GeneralCar("Nissan", 180);
```

```
car2 = new GeneralCar("Marutu Alto", 80);
car3 = new GeneralCar("Mercedes", 200);
console.log(car1, car2, car3);
```

Previous

Next

CodeWithHarry