**Objects And Its Internal Representation In JavaScript**

In JavaScript, almost "everything" is an object.

* 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

All JavaScript values, except primitives, are objects.

## **JavaScript 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 quite different from JavaScript’s primitive data-types, It is more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “**key: value**” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

Example,

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

*“A JavaScript object is a collection of Key value pairs.”*

## **Creating a JavaScript Object**

There are different ways to create new objects:

1. Create a single object, using an object literal.
2. Create a single object, with the keyword new.
3. Define an object constructor, and then create objects of the constructed type.
4. Create an object using Object.create().

## *Using an Object Literal*

This is the easiest way to create a JavaScript Object. Using an object literal, we both define and create an object in one statement. An object literal is a list of key:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties

Example,

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

## *Using the JavaScript Keyword new*

The following example create a new JavaScript object using new Object(),and then adds 4 properties:

Example,

const person = new Object();  
  person.firstName: "John";  
  person.lastName: "Doe";  
  person.age: 50;  
  person.eyeColor: "blue";

## JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value. If person is an object, the following statement will not create a copy of person:

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Example,

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

const x = person;

x.age =10; //Will change both x.age and person.age

Any changes to x will also change person, because x and person are the same object.

1. *Using the Object.create method*

Objects can also be created using the Object.create() method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

Example,

var person = {  
  firstName: "John",  
  lastName: "Doe",  
  displayName: function(){

console.log(this.firstName + " " + this.lastName);

}  
};

var class1 = Object.create(person);

class1.displayName(); //Output:John Doe

## **JavaScript Properties**

Properties are the values associated with a JavaScript object. A JavaScript object is a collection of unordered properties. Properties can usually be changed, added, and deleted, but some are read only.

## **Accessing JavaScript Properties**

The syntax for accessing the property of an object is:

Example,

*objectName.property*// person.age

or

*objectName*["*property*"]   // person["age"]

or

*objectName*[*expression*]   // x = "age"; person[x]

**Adding New Properties**

We can add new properties to an existing object by simply giving it a value.  Assume that the person object already exists - we can then give it new properties:

Example,

Person.nationality = "English";

## **Deleting Properties**

The delete keyword deletes a property from an object:

Example,

**delete** person.age;

Or

**delete** person[“age”];

The delete keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again. The delete operator is designed to be used on object properties. It has no effect on variables or functions.  The delete operator should not be used on predefined JavaScript object properties. It can crash the application.

## **Nested Objects**

Values in an object can be another object:

Example,

myObj = {  
  name:"John",  
  age:30,  
  cars: {  
    car1:"Ford",  
    car2:"BMW",  
    car3:"Fiat"  
  }  
};

we can access nested objects using the dot notation or the bracket notation:

Example,

myObj.cars.car2;

or

myObj.cars["car2"];

or

myObj["cars"]["car2"];

or

**let** p1 = "cars";

**let** p2 = "car2";

myObj[p1][p2];

## **How to Display JavaScript Objects?**

Some common solutions to display JavaScript objects are:

1. Displaying the Object Properties by name
2. Displaying the Object Properties in a Loop
3. Displaying the Object using Object.values()
4. Displaying the Object using JSON.stringify()

## *Displaying Object Properties*

The properties of an object can be displayed as a string:

Example,

person.name + "," + person.age + "," + person.city;

## *Displaying the Object in a Loop*

The properties of an object can be collected in a loop:

Example,

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
let txt = "";  
for (let x in person) {  
txt += person[x] + " ";  
};  
  
console.log(txt);

## *Using Object.values()*

Any JavaScript object can be converted to an array using Object.values()

Example,

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
const myArray = Object.values(person);

myArray is now a JavaScript array, ready to be displayed:

Example,

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
const myArray = Object.values(person);  
console.log(myArray);

## *Stringify Dates*

JSON.stringify converts dates into strings:

Example,

const person = {  
  name: "John",  
  today: new Date()  
};  
  
let myString = JSON.stringify(person);  
console.log(myString);

## *Stringify Functions*

JSON.stringify will not stringify functions:

Example,

const person = {  
  name: "John",  
  age: function() {return 30;}   
};  
  
let myString = JSON.stringify(person);  
console.log(myString);

This can be "fixed" if you convert the functions into strings before stringifying.

Example,

const person = {  
  name: "John",  
  age: function() {return 30;}   
};

person.age = person.age.toString();  
  
let myString = JSON.stringify(person);  
console.log(myString);

## *Stringify Arrays*

It is also possible to stringify JavaScript arrays:

Example,

const arr = ["John", "Peter", "Sally", "Jane"];  
  
let myString = JSON.stringify(arr);  
console.log(myString);