# **Object and its internal representation in JavaScript**

Conceptually, Objects are same in all programming languages i.e they represent real-world things that we want to represent inside our programs with characteristics/properties and methods.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateNam, etc.

Let’s see an example:-

const firstObj = {  
1: “deepak”,  
“age”: 28  
}

firstObj is an object with 2 properties 1 and age and value as deepak and 28 .

JavaScript objects are somewhat different in the way they are created. There is no requirement for a class as such.

**Object Creation👇**

We can create objects in many ways in javascript, let’s look at each one of them.

1. Object literal (Direct way) — Object literals are a comma-separated list of key-value pairs wrapped in curly braces. Object literal property values can be of any data type, including array literals, functions, nested object literals or primitive data type.

var student = {  
id: 1,  
name: “deepak”,  
age: “27”,  
updateAddress: () => {  
// logic to update address  
},  
grade: [‘A’, ‘A+’, ‘A’]  
}

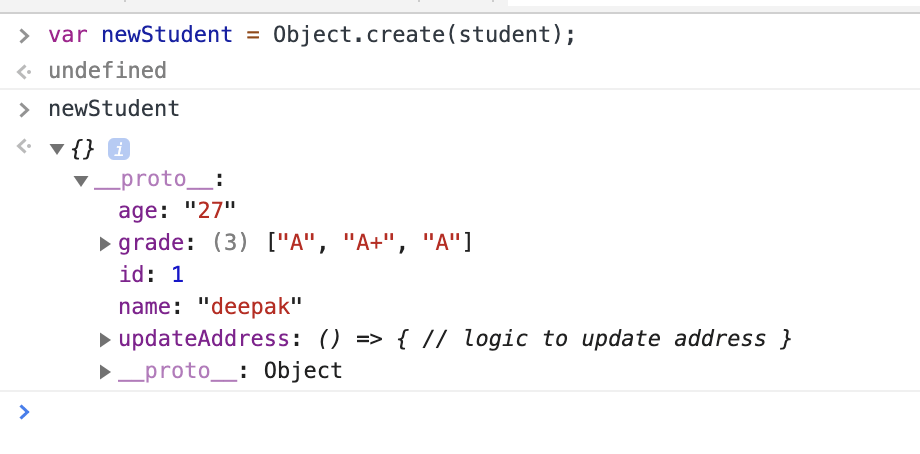
*Note: The student object keys in the above can be accessed via Dot notation i.e student.id, student.name or via a square bracket notation i.e student[‘id’], student[‘name’], etc*

2. Object.create()— the method creates a new object with the specified prototype and properties of the old object.

*Note: Every JavaScript function has a prototype property which .is empty by default. We may attached methods or properties to prototype.*

// syntax — Object.create(prototype[, propertiesObject])  
var newStudent = Object.create(student);   
// this create a new object with old object added in its prototype // chain

Below is the output of the object and prototype key (\_\_proto\_\_).



We can now add new properties and data to newStudentobject using the method we are learning here.

*Note: The newStudentwill have access to the parent studentobject keys and value as it's been added to newStudentprototype chain and this is one way we do inheritance in javascript. That is, newStudentwill store a link to studentobject. This parent object is also looked when a property is read.*

The parent can have a parent and so on. This is repeated until we reach an object that does not have any parent i.e the parent is null.

3. Object Instance— The use of Objectconstructor in conjunction with the “new” keyword allows us to initialize new objects.

Let’s take a look by an example:

const newObj = new Object();  
newObj.name = ‘Deepak’;  
newObj.location = ‘Delhi, India’;

However, the above method using new Object()is not well suited to programs that require the creation of multiple objects of the same kind, as it would involve repeatedly writing the above lines of code for each such object.

To deal with this problem, we can use the next method.

4. Object Constructor — Constructors can be useful when we need a way to create an object “type” that can be used multiple times without having to redefine the object every time and this could be achieved using the Object Constructor function.

Let’s take a look by an example:

function Vehicle(name, model) {   
this.name = name;   
this.model = model;   
}   
  
let car1 = new Vehicle(‘Fiesta’, ‘2019’);   
let car2 = new Vehicle(‘DC avanti’, ‘2018’);

We created two objects with the same property but with different values.

5. Object.assign() —this is another method to create a new object from other objects.

*Note: We will cover enumerable/ownership in the next part, so bear this with me.*

It copies the values of all enumerable own properties from one or more source objects to a target object. It will return the target object. Let’s understand by an examples:-

const target = { a: 1, b: 2 };  
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);  
// expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget);  
// expected output: Object { a: 1, b: 4, c: 5 }

var obj = { a: 1 };  
var copy = Object.assign({}, obj);  
console.log(copy); // { a: 1 }

There are a lot of use cases for Object.assign()like Object cloning, Merging objects, etc.

6. Object.fromEntries() — method transforms a list of key-value pairs into an object. Let’s take a look by an example

const entries = new Map([  
[‘foo’, ‘bar’],  
[‘baz’, 42]  
]);

const obj = Object.fromEntries(entries);

console.log(obj);  
// expected output: Object { foo: “bar”, baz: 42 }

*Note: The best way to create objects is via Literal notation as it takes less space in the source code. It’s clearly recognizable as to what is happening, so using new Object(), you are really just typing more and (in theory, if not optimized out by the JavaScript engine) doing an unnecessary function call. Also, literal notation creates the object and assigns the property in same line of code which is not the case with others.*

Thank You!!