**Objects in JavaScript: A Brief Overview**

In JavaScript, an object is a collection of key-value pairs where each key is a string or symbol, and the associated value can be any data type, including other objects. Objects are used to model real-world entities and enable developers to structure, organize, and interact with data in a meaningful way.

**Creating Objects**

You can create objects in JavaScript using either object literals or constructor functions. Here's a simple example using an object literal:

// **Object literal**

const person = {

name: "John Doe",

age: 30,

profession: "Developer",

};

Objects can also be created using constructor functions, providing a blueprint for creating multiple instances:

// Constructor function

function Person(name, age, profession) {

this.name = name;

this.age = age;

this.profession = profession;

}

// Create an instance

const person = new Person("John Doe", 30, "Developer");

**// Create an instance**

const person = new Person("John Doe", 30, "Developer");

**Internal Representation of Objects**

Understanding how JavaScript internally represents objects is essential for optimizing code performance and grasping some of the language's quirks.

**Objects and properties**

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();

myCar.make = 'Ford';

myCar.model = 'Mustang';

myCar.year = 1969;

Unassigned properties of an object are undefined (and not null).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see property accessors). Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:

myCar['make'] = 'Ford';

myCar['model'] = 'Mustang';

myCar['year'] = 1969;

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime).

Examples are as follows:

// four variables are created and assigned in a single go,

// separated by commas

var myObj = new Object(),

str = 'myString',

rand = Math.random(),

obj = new Object();

myObj.type = 'Dot syntax';

myObj['date created'] = 'String with space';

myObj[str] = 'String value';

myObj[rand] = 'Random Number';

myObj[obj] = 'Object';

myObj[''] = 'Even an empty string';console.log(myObj);

You can also access properties by using a string value that is stored in a variable:

var propertyName = 'make';

myCar[propertyName] = 'Ford';propertyName = 'model';

myCar[propertyName] = 'Mustang';

You can use the bracket notation with for...in to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:

function showProps(obj, objName) {

var result = ``;

for (var i in obj) {

// obj.hasOwnProperty() is used to filter out properties from the object's prototype chain

if (obj.hasOwnProperty(i)) {

result += `${objName}.${i} = ${obj[i]}\n`;

}

}

return result;

}

So, the function call showProps(myCar, "myCar") would return the following:

myCar.make = Ford

myCar.model = Mustang

myCar.year = 1969

**Creating Objects In JavaScript :**

Create JavaScript Object with Object Literal One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below

let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};

**Create JavaScript Object with Constructor**

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

function Vehicle(name, maker) {

this.name = name;

this.maker = maker;

}

let car1 = new Vehicle(’Fiesta’, 'Ford’);

let car2 = new Vehicle(’Santa Fe’, 'Hyundai’)

console.log(car1.name); //Output: Fiesta

console.log(car2.name); //Output: Santa Fe

**Using the JavaScript Keyword new**

The following example also creates a new JavaScript object with four properties:

Example

var person = new Object();

person.firstName = “John”;

person.lastName = “Doe”;

person.age = 50;

person.eyeColor = “blue”;

**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.

// Animal properties and method encapsulation

var Animal = {

type: 'Invertebrates', // Default value of properties

displayType: function() { // Method which will display type of Animal

console.log(this.type);

}

};

// Create new animal type called animal1

var animal1 = Object.create(Animal);

animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes

var fish = Object.create(Animal);

fish.type = 'Fishes';

fish.displayType();

// Output:Fishes

**Hidden Classes**

Modern JavaScript engines use a concept called hidden classes to optimize object property access. When an object is created, the engine associates it with a hidden class that dictates how properties are stored and accessed. This optimization enhances the execution speed of property lookups.

**Memory Management**

JavaScript manages memory automatically through mechanisms like garbage collection. Objects that are no longer reachable are marked for garbage collection, freeing up memory. Understanding these memory management principles is crucial for writing efficient and performant JavaScript code.

**Conclusion**

Objects are the backbone of JavaScript, providing a powerful and flexible way to represent and manipulate data. Understanding their internal representation, including property descriptors, prototype chains, hidden classes, and memory management, empowers developers to write efficient and optimized code. As you continue your journey with JavaScript, delve deeper into the intricacies of objects to harness their full potential in building robust and scalable applications.