**Objects and its internal representation in JavaScript**

* Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).
* Objects are 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.
* 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.

**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.

**The Basics of JavaScript Objects:**

* At its core, JavaScript is an object-oriented language, and nearly everything in JavaScript is an object or behaves like one.
* Objects in JavaScript are collections of key-value pairs, where each key is a string (or a symbol) and each value can be any data type, including other objects.
* This simplicity and flexibility make objects an integral part of the language's syntax.

**Internal Representation of Objects:**

* Understanding how JavaScript internally represents objects is crucial for developers aiming to write efficient and optimized code.
* Unlike some other languages, JavaScript utilizes a variety of internal mechanisms for object representation.

**Properties and Methods:**

* Objects in JavaScript consist of properties and methods.
* Properties are essentially variables attached to objects, while methods are functions that belong to objects.
* The internal representation involves storing these properties and methods, making them accessible through dot notation or square bracket notation.

**Prototypes and Inheritance:**

* JavaScript employs a prototype-based model for inheritance.
* Each object in JavaScript has an internal link to another object called its prototype.
* This linkage creates a chain of prototypes, allowing objects to inherit properties and methods from their prototypes. This mechanism significantly contributes to code reuse and the creation of more maintainable and scalable applications.

**Hidden Classes and Shapes:**

* JavaScript engines use optimizations like hidden classes and shapes to speed up property access.
* Hidden classes are used to represent the structure of an object, and shapes are used to represent the transitions between hidden classes as properties are added or removed.
* These optimizations enhance the performance of property access and modification.

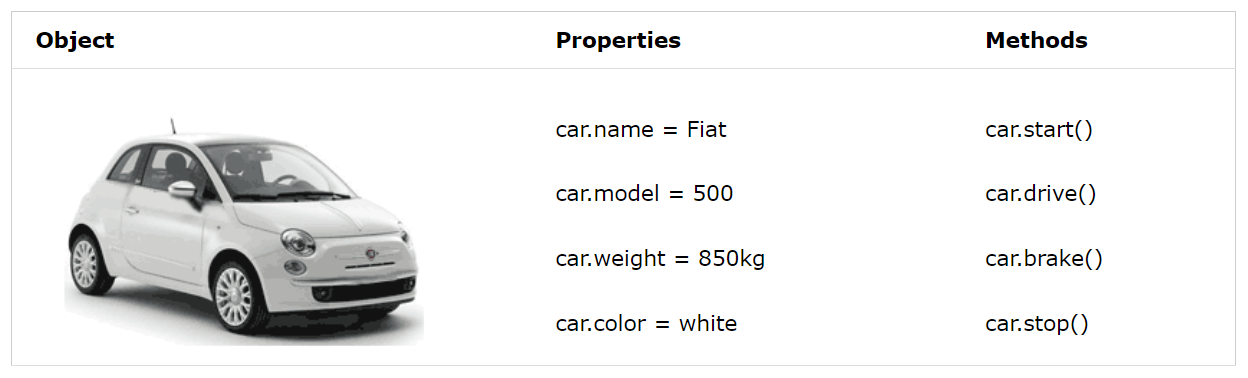
**Object Descriptors:**

* Every property of an object in JavaScript has an associated object descriptor that contains information about the property, such as whether it is writable, enumerable, or configurable.
* This metadata is crucial for determining how properties can be interacted with and modified.

**Practical Implications for Developers:**

* Understanding the internal representation of objects in JavaScript has practical implications for developers.
* Writing efficient code involves considering factors like property access patterns, prototype chains, and the impact of object creation on performance. Developers can leverage this knowledge to optimize their code, improve application performance, and create more maintainable and scalable projects.

## **Real Life Objects, Properties, and Methods:**



**Examples:**1. Basic object.

// Creating a simple object

let person = {

name: 'John',

age: 30,

sayHello: function() {

console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);

}

};

// Accessing properties and invoking a method

console.log(person.name); // Output: John

person.sayHello(); // Output: Hello, my name is John and I'm 30 years old.

2. Prototype and Inheritance.

// Creating a prototype object

let animal = {

makeSound: function() {

console.log('Some generic sound');

}

};

// Creating a new object using the prototype

let dog = Object.create(animal);

// Extending the new object

dog.breed = 'Labrador';

dog.bark = function() {

console.log('Woof, woof!');

};

// Using inherited properties and methods

console.log(dog.breed); // Output: Labrador

dog.makeSound(); // Output: Some generic sound

dog.bark(); // Output: Woof, woof!

**Conclusion:**

* Objects lie at the heart of JavaScript, providing a versatile and dynamic foundation for building complex applications.
* Delving into the internal representation of objects unveils the intricacies of how JavaScript handles data structures, inheritance, and optimization.
* Armed with this knowledge, developers can unlock the full potential of objects in JavaScript, writing code that is not only functional but also performs optimally in various scenarios.