Objects and its internal representation in Javascript

Introduction:

In JavaScript, objects are a fundamental part of the language. They allow us to represent and manipulate complex data structures. Understanding how objects are internally represented in JavaScript is crucial for writing efficient and effective code. In this we will explore the internal representation of objects in JavaScript and delve into some key concepts related to objects.

1. Objects in JavaScript:

In JavaScript, objects are collections of key-value pairs, where the keys are strings (or symbols) and the values can be of any type, including other objects. Objects can be created using the object literal syntax, constructor functions, or the ES6 class syntax.

2. Internal Representation of Objects:

Internally, objects in JavaScript are implemented as hash tables or dictionaries. This means that when we create an object, JavaScript allocates memory to store its properties and methods. Each property is stored as a key-value pair, where the key is a string and the value can be any JavaScript value.

3. Property Access and Lookup:

To access properties of an object, JavaScript uses a process called property lookup. When we try to access a property, JavaScript first checks if the property exists directly on the object. If not found, it looks up the prototype chain until it finds the property or reaches the end of the chain (Object.prototype). This mechanism allows objects to inherit properties and methods from their prototypes.

4. Adding and Modifying Properties:

In JavaScript, we can add or modify properties of an object at any time. When we add a new property, JavaScript dynamically allocates memory to store the property and updates the internal representation of

the object. Similarly, when we modify an existing property, JavaScript updates the value associated with that property.

5. Object Identity and Equality:

In JavaScript, objects are compared by reference, not by value. This means that two objects with the same properties and values are considered equal only if they refer to the same memory location. Understanding this concept is crucial when working with objects and performing equality checks.

6. Garbage Collection:

JavaScript uses automatic garbage collection to reclaim memory that is no longer in use. When an object is no longer referenced by any part of the program, it becomes eligible for garbage collection. The JavaScript engine periodically identifies and frees up memory occupied by unused objects.

Conclusion:

Objects are a powerful feature of JavaScript, allowing us to represent complex data structures and build sophisticated applications. Understanding the internal representation of objects helps us write efficient code and make informed decisions when working with objects. By grasping the concepts discussed in this post, you'll be better equipped to leverage the full potential of objects in JavaScript. Happy coding!