In JavaScript, an object is a composite data type that allows you to group related data and functions into a single entity. Objects are essentially collections of key-value pairs, where each key is a string (or a Symbol in ES6+), and each value can be of any data type – including other objects, functions, and even primitives like strings, numbers, or booleans.

Objects are defined using curly braces **{}** and can contain properties and methods. Properties are key-value pairs, where the key is a property name and the value can be any valid JavaScript data type. Methods are functions stored as values in object properties.

const person = {

name: 'John',

age: 30,

sayHello: function() {

console.log(`Hello, my name is ${this.name}!`);

}

};

**Internal Representation of Objects**

To understand JavaScript objects better, let's look at their internal representation. In JavaScript engines, objects are implemented using various data structures, but a common way is to use a hash table. Each property name is hashed to create an index in the table, allowing for efficient retrieval and storage of properties.

When you access a property of an object, the JavaScript engine hashes the property name to find its location in the hash table, making property lookup quick and efficient.

**Object Properties and Methods**

JavaScript objects can have two main types of members: properties and methods.

1. **Properties**: Properties are values associated with the object. They can be primitive values or other objects. To access a property, you use dot notation or square brackets.

console.log(person.name); // Accessing the 'name' property using dot notation

console.log(person['age']); // Accessing the 'age' property using square brackets

1. **Methods**: Methods are functions associated with the object. They can be called like regular functions.

person.sayHello(); // Calling the 'sayHello' method

### Object Prototypes and Inheritance

One of the powerful features of JavaScript objects is the concept of prototypal inheritance. In JavaScript, objects can inherit properties and methods from other objects through their prototypes. This allows for code reusability and the creation of complex object hierarchies.

// Creating a Person prototype

function Person(name, age) {

this.name = name;

this.age = age;

}

// Adding a method to the prototype

Person.prototype.sayHello = function() {

console.log(`Hello, my name is ${this.name}!`);

};

// Creating an object using the Person prototype

const person = new Person('John', 30);

person.sayHello(); // Outputs: Hello, my name is John!

### Object Literal vs. Constructor Function

In JavaScript, you can create objects in two main ways: using object literals or constructor functions. Object literals are simple and concise, while constructor functions provide more flexibility and the ability to create multiple instances of an object with shared methods.