**Objects and its internal representation in Javascript**

In JavaScript, objects play a crucial role and serve as the building blocks for modern applications. Unlike primitive data types (such as numbers, strings, booleans, null, undefined, and symbols), objects are more complex. Here are some key points about objects and their internal representation:

**1. Reference Data Type:**

- An object is a reference data type. When you assign a value to an object, it receives a reference or pointer to the memory location where the actual object is stored.

- The object itself is not directly stored in the variable; instead, the variable holds a reference to it.

**2. Properties and Methods:**

- An object is an unordered collection of related data, represented as "key: value" pairs.

- Properties (variables) and methods (functions) define the characteristics of an object.

- For example, if you have an object representing a student, it might have properties like name, age, address, and methods like updateAddress and updateName.

**3. Accessing Properties:**

- You can access an object's properties using dot notation:

| const myCar = new Object();  myCar.make = 'Ford';  myCar.model = 'Mustang';  myCar.year = 1969; |
| --- |

- Alternatively, use bracket notation:

| myCar['make'] = 'Ford';  myCar['model'] = 'Mustang';  myCar['year'] = 1969; |
| --- |

**4. Undefined Properties:**

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

- For example, `myCar.color` would be undefined.

**5. Associative Arrays:**

- Objects are sometimes called associative arrays because each property is associated with a string value that serves as the key.

- You can access properties dynamically using square brackets.

Internally, objects are stored as collections of key-value pairs. When you access an object's property or method, JavaScript looks up the property name in this internal collection. Objects are powerful tools for organizing and managing data in JavaScript applications.