**Object and internal representation in JavaScript**

**Object**

Objects, in JavaScript, is its 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.

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.

Loosely speaking, objects in 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.

**Internal representation**

JavaScript engines use various techniques to represent objects efficiently in memory. While the internal details may vary between different JavaScript engines (such as V8, Spider Monkey), the following concepts are commonly employed:

* Properties and Prototypes: Each object in JavaScript has an internal property known as [[Prototype]], which references another object called its prototype. This forms the basis of JavaScript's prototype-based inheritance model, allowing objects to inherit properties and methods from their prototypes.
* Property Descriptors: JavaScript objects also store additional metadata, known as property descriptors, for each property. Property descriptors contain information such as the property's value, whether it is writable, enumerable, or configurable, and any getter/setter functions associated with the property.
* Hidden Classes and Inline Caching: JavaScript engines optimize property access by using techniques such as hidden classes (or shapes) and inline caching. These optimizations improve performance by efficiently mapping property accesses to memory locations and minimizing look-up overhead.