In JavaScript, almost everything is the object. Objects are the most important data type and form the building blocks for modern JavaScript.

Booleans, Strings, Arrays, Functions, Numbers, Dates, Maths, etc. are objects.

Nearly all objects in JavaScript are instances of Object; a typical object inherits properties (including methods) from Object.prototype, although these properties may be shadowed (a.k.a. overridden). The only objects that don't inherit from Object.prototype are those with null prototype, or descended from other null prototype objects.

Changes to the Object.prototype object are seen by all objects through prototype chaining, unless the properties and methods subject to those changes are overridden further along the prototype chain. This provides a very powerful although potentially dangerous mechanism to override or extend object behavior. To make it more secure, Object.prototype is the only object in the core JavaScript language that has immutable prototype — the prototype of Object.prototype is always null and not changeable.

These objects are quite different from JavaScript's primitive data types in the sense that these primitive data types all store a single value each (depending on their types). Objects are also variables and can contain many values

Objects are more complex and each object may contain any combination of 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.

Object values are written as name : value pairs (name and value separated by a colon).

By convention, the objects are named. i.e. the name of the objects describes the data contained by the object or the action performed by the object.

It is a common practice to declare objects with the 'const' keyword.

Objects written as name value pairs are similar to:

Associative arrays in PHP

Dictionaries in Python

Hash tables in C

Hash maps in Java

Hashes in Ruby and Perl

Objects are mutable

Basic Object Syntax:

```
let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30        // by key "age" store value 30
};
```

These objects have also got various methods, accessors, constructors, prototypes, etc.

Object Prototype:

Object.prototype is on the top of the prototype inheritance chain.

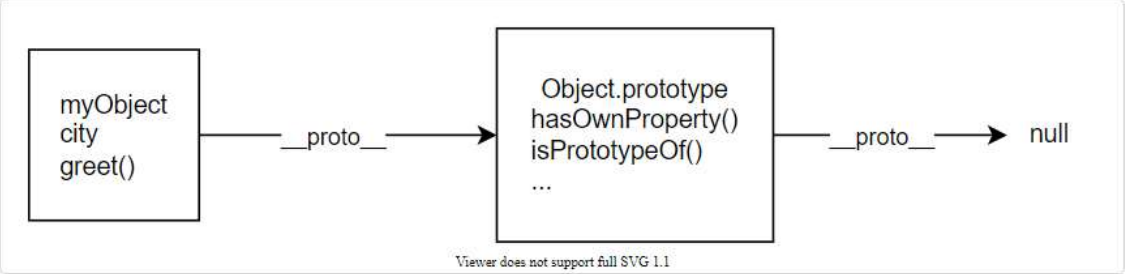
Prototypes are the mechanism by which JavaScript objects inherit features from one another. \

Every object in JavaScript has a built-in property, which is called its prototype. The prototype is itself an object, so the prototype will have its own prototype, making what's called a prototype chain. The chain ends when we reach a prototype that has null for its own prototype.

When one tries to access a property of an object: if the property can't be found in the object itself, the prototype is searched for the property. If the property still can't be found, then the prototype's prototype is searched, and so on until either the property is found, or the end of the chain is reached, in which case undefined is returned.

Object.prototype, and it is the most basic prototype, that all objects have by default. The prototype of Object.prototype is null, so it's at the end of the prototype chain:

Prototype chain for myObject



The prototype of an object is not always Object.prototype.

Object properties:

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

The syntax for accessing the property of an object is:

objectName.property // person.age

or

objectName["property"] // person["age"]

One can add new properties to an existing object by simply giving it a value.

The delete keyword deletes a property from an object:

delete person.age

Object methods:

One can access an object method with the following syntax:

objectName.methodName()

You will typically describe fullName() as a method of the person object, and fullName as a property.

The fullName property will execute (as a function) when it is invoked with ().

This example accesses the fullName() method of a person object:

Object Constructor:

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

References:

1) MDN Webdocs
2) Geeks for geeks
3) W3schools