Web pages consists of three parts :

content (HTML), presentation (CSS), and behavior (JavaScript).

Initially web pages was used to show just content, Javascript was created to make web pages alive.

Initially web was a collection of static pages, tied together with hyperlinks, but with the growth of the web, many developers and browsers required more features.

Javascript primarily used to enhance user friendly experience of web pages. These include dynamically updating web pages, user interface enhancements such as menus and dialog boxes, animations, 2D and 3D graphics, interactive maps, video players, and more. This mode of JavaScript usage in the web browser is also referred to as client-side javascript.

JavaScript is a cross-platform, object-oriented scripting language. JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.

JavaScript is prototype-based scripting language with dynamic typing and first-class functions. This mix of features makes it a multi-paradigm language.

javascript is a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles.

* Javascript is High-level, interpreted programming language.
* Dynamic type / static types / weakly typed / strongly typed
* Javascript is prototype-based programming language.
* first-class functions
* javascript support imperative and structural programming style.

**Imperative and structured**

JavaScript supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, do while loops, etc.). One partial exception is scoping: JavaScript originally had only function scoping with var. ECMAScript 2015 added keywords let and const for block scoping, meaning JavaScript now has both function and block scoping. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allows the semicolons that would normally terminate statements to be omitted.

**Dynamic Typing**

Type means provide data type to the variable.

Javascript is untyped language, untyped means no data type declaration.

Untyped means dynamically typed and typed means statically typed.

JavaScript is a dynamically typed language. That means you don't have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. For example, a variable initially bound to a number may be reassigned to a string.

**Note : Complete details of type is in chapter “data-type”**

**Prototypes**

JavaScript uses prototypes where many other object-oriented languages use classes for inheritance. It is possible to simulate many class-based features with prototypes in JavaScript.

**Prototype-based (object-oriented)**

JavaScript is almost entirely object-based. In JavaScript, an object is an associative array, augmented with a prototype. each string key provides the name for an object property, and there are two syntactical ways to specify such a name: dot notation (obj.x = 10) and bracket notation (obj['x'] = 10). A property may be added, rebound, or deleted at run-time. Most properties of an object (and any property that belongs to an object's prototype inheritance chain) can be enumerated using a for...in loop.

JavaScript has a small number of built-in objects, including Function and Date.

**Functional**

A [function](https://en.wikipedia.org/wiki/Subroutine) is [first-class](https://en.wikipedia.org/wiki/First-class_function); a function is considered to be an object. As such, a function may have properties and methods, such as .call() and .bind(). A nested function is a function defined within another function. It is created each time the outer function is invoked. In addition, each nested function forms a [lexical closure](https://en.wikipedia.org/wiki/Closure_(computer_programming)): The [lexical scope](https://en.wikipedia.org/wiki/Scope_(programming)) of the outer function (including any constant, local variable, or argument value) becomes part of the internal state of each inner function object, even after execution of the outer function concludes. JavaScript also supports [anonymous functions](https://en.wikipedia.org/wiki/Anonymous_function).

Other points

**Functions as object constructors**

Prefixing a function call with new will create an instance of a prototype, inheriting properties and methods from the constructor (including properties from the Object prototype). ECMAScript 5 offers the Object.create method, allowing explicit creation of an instance without automatically inheriting from the Object prototype (older environments can assign the prototype to null). The constructor's prototype property determines the object used for the new object's internal prototype. New methods can be added by modifying the prototype of the function used as a constructor. JavaScript's built-in constructors, such as Array or Object, also have prototypes that can be modified. While it is possible to modify the Object prototype, it is generally considered bad practice because most objects in JavaScript will inherit methods and properties from the Object prototype, and they may not expect the prototype to be modified.

**Object composition and inheritance**

Whereas explicit function-based delegation does cover [composition](https://en.wikipedia.org/wiki/Object_composition) in JavaScript, implicit delegation already happens every time the prototype chain is walked in order to, e.g., find a method that might be related to but is not directly owned by an object. Once the method is found it gets called within this object's context. Thus [inheritance](https://en.wikipedia.org/wiki/Inheritance_(computer_science)) in JavaScript is covered by a delegation automatism that is bound to the prototype property of constructor functions.

**Delegative**

JavaScript supports implicit and explicit [delegation](https://en.wikipedia.org/wiki/Delegation_(object-oriented_programming)).

Functions as roles (Traits and Mixins)

JavaScript natively supports various function-based implementations of [Role](https://en.wikipedia.org/wiki/Role-oriented_programming) patterns like [Traits](https://en.wikipedia.org/wiki/Traits_(computer_science)) and [Mixins](https://en.wikipedia.org/wiki/Mixin). Such a function defines additional behavior by at least one method bound to the this keyword within its function body. A Role then has to be delegated explicitly via call or apply to objects that need to feature additional behavior that is not shared via the prototype chain.

**Functions as methods**

Unlike many object-oriented languages, there is no distinction between a function definition and a method definition. Rather, the distinction occurs during function calling; when a function is called as a method of an object, the function's local this keyword is bound to that object for that invocation.