**Es5**

-ECMAScript phát hành 2009

Các đặc điểm chính: -strict mode

-Json: json.parse() json.stringify()

-Array method: forEach, Filter,reduce,map

-function binding

-string trim()

-property access str[0]

Es6 2015

-thay thế var bằng let, const

-Arrow function

-template string ``

-default parameter: function (a=1){}

-destructuring: const [a,b]=[1,2]

-classes

-module import export

-promise: new Promise((resolve, reject) => {...});

-spread/rest operator: const a={…b}

-loops

**Es7 2016**

-includes

-lũy thừa \*\*

**Es8 2017**

-async await

-obj values=>trả về mảng value của abj, obj entries trả về mảng chứa các cặp key value của obj

-padStart padEnd thêm số lượng ký tự vào trước sau

-Object.getOwnPropertyDescriptors() **Lấy tất cả property descriptors của một object**

-Trailing Commas cho phép dấu phẩy cuối trong khai báo gọi hàm

-Shared Memory và Atomics

**Es9 2018**

-Async Iteration lặp bất đồng bộ

-Rest/Spread Properties cho Object

-Promise.prototype.finally()

-Cải tiến Regular Expression

**Es10 2019**

-Array.prototype.flat() và flatMap()

-Object.fromEntries() **Chuyển đổi danh sách cặp key-value thành object** (ngược lại với Object.entries()):

-String.prototype.trimStart() và trimEnd()

-Optional Catch Binding bỏ qua tham số error trong catch

-Function.prototype.toString() Cải tiến

**Trả về nguyên bản code hàm** (kể cả comments, khoảng trắng):

-JSON.stringify()

-Array.prototype.sort() đảm bảo thứ tự cho các phần tử bằng nhau

Các kiểu dữ liệu

-nguyên thủy: string, number, bigint, boolean, undefined, null,symbil

-không nguyên thủy: object, array,function,date, reg

Stored by value: áp dụng cho kiểu nguyên thủy, tạo bản sao giá trị

Stored by reference: áp dụng cho kiểu không nguyên thủy, tham chiếu đến cùng 1 vùng nhớ

Pass by Value: Hàm nhận bản sao của giá trị, không ảnh hưởng đến biến gốc

Pass by Reference: hàm có thể thay đổi thuộc tính của object gốc

**Iterator** là một cơ chế cho phép bạn lặp qua các phần tử của một tập hợp theo từng bước: Là một object có phương thức next() trả về một object với 2 thuộc tính

**Iterable** là một object có thể được lặp qua bằng cách sử dụng iterator. Array,string, map

**Generator** là một hàm đặc biệt có thể tạm dừng và tiếp tục thực thi, giúp tạo iterator dễ dàng hơn.

Các kiểu khai báo:

Var: funtion scope, được hoisting, undefined nếu dùng trước, có thể gán và khai báo lại

Let: block scope, có hoisting nhưng lỗi nếu truy cập trc, có thể gán lại không khai báo lại

Const: block scope, có hoisting nhưng lỗi nêu truy cập trc, không thể gán lại nhung thay đổi được nội dung bên trong obj array

**Scope (phạm vi)** là nơi mà biến có thể được truy cập.

Các loại scope:

**Global Scope**: biến khai báo bên ngoài mọi function/block, dùng được ở mọi nơi.

**Function Scope**: biến khai báo trong function, chỉ dùng trong function đó.

**Block Scope** (ES6): với let và const, biến chỉ tồn tại trong block {} (vd: if, for).

**Module Scope**: mỗi file module có scope riêng (ES Module).

Trình biên dịch (Compiler): chuyển toàn bộ mã nguồn sang ngôn ngữ máy trước khi chạy. Ví dụ: C, C++.

Trình thông dịch (Interpreter): chạy từng dòng mã khi thực thi. Ví dụ: Python.

JavaScript sử dụng cả hai:

Trình duyệt (V8, SpiderMonkey, v.v.) biên dịch + thông dịch JS thành mã máy (Just-in-time Compilation - JIT).

Arrow funtion là 1 cách khai báo ngắn gọn 1 function const func ()=>{}

Es module dùng export import để chia nhỏ code, mỗi module có 1 scope riêng

Stateful:

Có **trạng thái (state)**, lưu thông tin và có thể thay đổi theo thời gian có thể phụ thuộc vào trạng thái trước đó (counter).

Ví dụ: Component trong React dùng useState.

Stateless:

**Không lưu trạng thái**, đầu vào nào thì đầu ra đó, không phụ thuộc trạng thái bên trong, mỗi lần gọi hàm là 1 giá trị độc lập

Ví dụ : hàm tính tổng

Classes trong JavaScript từ ES6 là cú pháp giúp tạo lập và quản lý các đối tượng theo hướng đối tượng (OOP) dễ dàng hơn gồm các thành phần chính, constructor, phương thức, thuộc tính

Symbol là một kiểu dữ liệu nguyên thủy mới từ ES6. (chưa rõ)

Dùng để tạo key duy nhất cho object, không thể bị trùng:

const id = Symbol('id');

const user = {

[id]: 123

};

**Đồng bộ (Synchronous)**

Code chạy **tuần tự**, câu lệnh sau phải đợi câu lệnh trước hoàn thành.

console.log("1")

console.log("2"); *// Phải đợi dòng trên xong mới chạy*

console.log("3"); *// Phải đợi dòng trên xong mới chạy*

**Bất đồng bộ (Asynchronous)**

Code **không chạy tuần tự**, các tác vụ chạy ngầm và thông báo kết quả sau khi hoàn thành.

**Ví dụ:**

console.log("1");

setTimeout(() => console.log("2"), 1000); *// Chạy sau 1 giây, không block*

console.log("3"); *// Chạy ngay, không đợi setTimeout*

Kết quả: 1 → 3 → 2.

**Đơn luồng (Single-threaded)**

JavaScript là **đơn luồng**, chỉ có **một luồng chính** thực thi code.  
Nhờ **Event Loop**, các tác vụ bất đồng bộ (như setTimeout, fetch) được ủy thác cho **Web APIs** (của trình duyệt hoặc Node.js), sau đó kết quả được đẩy vào **Callback Queue** và trở về Call Stack khi rảnh.

**Đa luồng (Multi-threaded)**

Một số ngôn ngữ (Java, C#, Go) hỗ trợ **nhiều luồng chạy song song**.

**Ưu điểm**: Xử lý CPU-intensive tasks (như video encoding) nhanh hơn.

**Nhược điểm**: Phức tạp (race condition, deadlock).

**Promise là gì?**

Là một **cơ chế xử lý bất đồng bộ**, đại diện cho một tác vụ **chưa hoàn thành** nhưng sẽ hoàn thành trong tương lai (thành công hoặc thất bại).

**3 trạng thái**:

**Pending**: Đang chờ.

**Fulfilled**: Thành công.

**Rejected**: Thất bại.

**Call Stack**

Là **ngăn xếp** lưu các hàm đang thực thi (LIFO - Last In First Out).

**Message Queue (Task Queue)**

Chứa các **callback** từ Web APIs (ví dụ: setTimeout, fetch, DOM events).

**Event Loop**

Là cơ chế **kiểm tra Call Stack rỗng → lấy callback từ Message Queue đẩy vào Call Stack**.

**Quy trình**:

Chạy code đồng bộ trước (Call Stack).

Nếu gặp setTimeout, fetch, Promise.then, giao cho **Web APIs** xử lý.

Khi Web APIs hoàn thành, callback được đẩy vào **Message Queue**.

**Event Loop** đợi Call Stack rỗng → đẩy callback vào Call Stack.

**Biến (Variable)**

Là **tên tham chiếu** đến một giá trị trong bộ nhớ.

**Giá trị (Value)**

Là dữ liệu được lưu trong bộ nhớ (số, chuỗi, object, ...).

**Primitive Types (Giá trị nguyên thủy)**

Là **immutable** (không thay đổi được), được lưu trực tiếp trong **Stack**.

**Ví dụ**: number, string, boolean, null, undefined, symbol, bigint.

**Truyền bằng giá trị (pass by value)**:

**Reference Types (Tham chiếu)**

Là **mutable** (có thể thay đổi), được lưu trong **Heap**, biến chỉ lưu **địa chỉ tham chiếu**.

**Ví dụ**: object, array, function.

**Truyền bằng tham chiếu (pass by reference)**:

**Stack**

Là vùng nhớ **tự động quản lý** (khi hàm chạy xong, bộ nhớ tự giải phóng).

Lưu **biến primitive** và **tham chiếu hàm**.

function foo() {

let x = 10; *// Lưu trong Stack*

return x;

}

**Heap**

Là vùng nhớ **động**, lưu **object, array, function**.

**Phải quản lý thủ công** (trong JS, **Garbage Collector** tự dọn rác).

let obj = { name: "Alice" }; *// Object lưu trong Heap, biến `obj` lưu địa chỉ trong Stack*