JavaScript vs TypeScript

JavaScript হচ্ছে ব্রাউজার ও সার্ভারের জন্য রানটাইম ভাষা; **TypeScript** হচ্ছে JavaScript-এর উপরে দাঁড়ানো একটি *superset* — অর্থাৎ সব ES/JS চলে, আর পরে টাইপসিস্টেম ও আরও অনেক ডেভেলপার-ফ্রেন্ডলি ফিচার যোগ করে। নিচে ধাপে ধাপে বিস্তারিত তুলনা দিলাম।

1) মৌলিক ধারণা

- **JavaScript (JS)**
    - ডায়নামিকালি টাইপড ভাষা — ভ্যারিয়েবলকে টাইপ ডিক্লেয়ার করা লাগে না।
    - ব্রাউজার ও Node.js-এ সরাসরি চলবে।
    - ইন্টারপ্রেটেড (আসলেই engines — V8, SpiderMonkey ইত্যাদি compile করে JIT)।
- **TypeScript (TS)**
    - Microsoft তৈরি করা ভাষা। TS হলো *statically typed superset of JavaScript*.
    - TS কোডকে প্রথমে **transpile** করে JavaScript-এ (tsc বা Babel ব্যবহার করে)।
    - টাইপ চেকিং compile-time-এ করে — runtime-এ JS চলে।

2) টাইপিং (Type system)

- **JS:** ডায়নামিক টাইপ — এক ভ্যারিয়েবল কখনো সংখ্যা, কখনো স্ট্রিং হতে পারে। ভুল টাইপ runtime-এ ধরা পড়ে (বা না পড়েও পারে)।
- **TS:** স্ট্যাটিক টাইপিং — তুমি টাইপ লিখতে পারো (বা inference থাকবে)। টাইপ মিসম্যাচগুলো কম্পাইল টাইমে ধরা পড়ে।
    - Primitive types: string, number, boolean
    - Complex: array, tuple, enum, object
    - Advanced: union, intersection, generics, mapped types, conditional types

- **উদাহরণ (TS):**

```
function add(a: number, b: number): number {
  return a + b;
}
// ভুল: add("1", 2) — compile-time এ error দেবে
```

3) সিনট্যাক্স ও ভাষাগত ফিচার

- **JS**: ES6+ থেকে অনেক আধুনিক ফিচার এসেছে — classes, modules, arrow functions, async/await ইত্যাদি।
- **TS**: উপরন্তু interface, enum, access modifiers (public/private/protected), readonly, abstract classes, advanced generics ইত্যাদি দেয়।
- TS-এ তুমি JS এর সব কোড লিখে দিতে পারো — পরে টাইপ যোগ করলে ধরা পড়বে।

4) টুলিং ও ডেভেলপার এক্সপিরিয়েন্স

- **TypeScript সুবিধা**:
  - Editor autocompletion, intelligent refactor, go-to-definition শক্তিশালী।
  - বড় প্রজেক্টে রিফ্যাক্টর করা সহজ।
  - API কনট্র্যাক্ট স্পষ্ট থাকে — teammates-এর জন্য সুবিধা।
- **JavaScript**:
  - সরাসরি রান করা যায়, কম কমপ্লেক্সিটি।
  - ছোট স্ক্রিপ্ট বা প্রোটোটাইপ তৈরিতে দ্রুত।

5) কম্পাইল/বিল্ড প্রক্রিয়া

- **JS:** সরাসরি রান (বা Babel/webpack দিয়ে transpile করে পুরোনো ব্রাউজার সাপোর্ট)।

- **TS:** tsc বা Babel → .ts / .tsx ফাইলকে .js-এ রূপান্তর করে। tsconfig.json দিয়ে কনফিগার করা লাগে (target, module, strict ইত্যাদি)।
- TS ব্যবহার করলে build step যুক্ত হয় — কিন্তু modern toolchains (esbuild, swc, Vite) খুব দ্রুত।

---

6) Runtime performance

- TS নিজে রান করে না — ফলত **runtime performance পুরোপুরি JavaScript-এর মতোই**। টাইপিং compile-time-এ কাজ করে; runtime-এ টাইপ ইন্টারফেস মুছে যায় (except যদি তুমি explicit runtime checks যোগ করো)।

---

7) Ecosystem এবং Interop

- **Library ব্যবহার**: অধিকাংশ লাইব্রেরি JS-এ; TS প্রজেক্টে @types/* থেকে type definitions পাওয়া যায় (DefinitelyTyped)। অনেক লাইব্রেরি নিজেই type definitions দেয়।
- **Migration**: ধাপে ধাপে .js থেকে .ts করা যায় — mixed codebase সম্ভব (allowJs), এবং ধীরে ধীরে টাইপ যোগ করা যায়।
- **Declaration files**: .d.ts ফাইল দিয়ে JS লাইব্রেরির জন্য টাইপ বর্ণনা করা হয়।

---

8) কোনো সমস্যা বা কনসিডারেশন

- **TypeScript-এ কিছু কষ্ট**:
  - নতুনদের জন্য টাইপ কনসেপ্ট initially কঠিন (generics, mapped, conditional)।
  - অতিরিক্ত কনফিগারেশন/বিল্ড স্টেপ।
  - টাইপ সিস্টেম সবসময় perfect নয় — কখনো any বা টাইপ assertions (as) দিয়ে হার্ড-ইস্কেপ করতে হয়।
- **JavaScript-এর সমস্যা**:
  - বড় প্রজেক্টে টাইপ-সংক্রান্ত runtime bug বেশি হতে পারে।
  - রিফ্যাক্টরিং ঝুঁকিপূর্ণ হতে পারে।

---

9) উদাহরণ — একই কাজ JS vs TS

- **JavaScript**

```
// users.js
function getFullName(user) {
  return user.firstName + " " + user.lastName;
}
// যদি user null আসলে runtime error!
```

TypeScript

```
interface User {
  firstName: string;
  lastName: string;
  age?: number;
}

function getFullName(user: User): string {
  return `${user.firstName} ${user.lastName}`;
}
// compile-time এ নিশ্চিত — user-এ অবশ্যই firstName ও lastName আছে
```

10) কখন কোনটা ব্যবহার করবেন?

- **TypeScript ভাল যখন**
  - বড় কোডবেস, বহু ডেভেলপার, দীর্ঘ-মেয়াদি maintenance।
  - API contracts, লাইব্রেরি ডেভেলপমেন্ট, ডিফেন্ড টাইপ সেফটি দরকার।

- **JavaScript ভাল যখন**
    - ছোট প্রোজেক্ট, প্রোটোটাইপ, quick scripting।
    - শিক্ষামূলক কারণেই সরলতা প্রয়োজন হলে।

---

11) শুরু করার গাইড (সংক্ষিপ্ত)

- ছোট প্রোজেক: শুরু JS দিয়ে, পরে প্রয়োজন হলে TS-এ মাইগ্রেট।
- নতুন প্রোজেক যেখানে stability দরকার: শুরুই TS দিয়ে — tsconfig.json-এ strict: true রাখলে ভালো।
- উন্নত টুলিং: VSCode + ESLint + Prettier + TypeScript integration ব্যবহার করুন।

---

12) সংক্ষিপ্ত তুলনা টেবিল

দিক	**JavaScript**	**TypeScript**
টাইপিং	Dynamic	Static (compile-time)
রানটাইম	সরাসরি ব্রাউজার/Node	Transpile → JS → রান
বাগ ধরার সময়	Runtime	Compile-time
শেখার পাথ	হালকা দ্রুত	একটু বেশি লার্নিং কুরি
বড় প্রজেক্ট	ঝুঁকিপূর্ণ	ভালো maintainability
Tooling	প্রচুর	আরও উন্নত IDE support

একজন সফটওয়্যার ইঞ্জিনিয়ার হিসেবে JavaScript (JS) এবং TypeScript (TS)-এর মধ্যে পার্থক্য গভীরভাবে বোঝা খুবই গুরুত্বপূর্ণ। সংক্ষেপে, **TypeScript হলো JavaScript-এর একটি স্ট্যাটিক টাইপড সুপারসেট (Statically Typed Superset)**, যা বড় স্কেলের অ্যাপ্লিকেশন ডেভেলপমেন্ট সহজ করার জন্য ডিজাইন করা হয়েছে।

আসুন, এই দুটি ল্যাঙ্গুয়েজের মূল পার্থক্যগুলো বিস্তারিত বিশ্লেষণ করি।

প্রধান ধারণা: স্ট্যাটিক বনাম ডায়নামিক টাইপিং

এটাই হলো JS এবং TS-এর মধ্যে সবচেয়ে মৌলিক এবং গুরুত্বপূর্ণ পার্থক্য।

- **JavaScript (ডায়নামিক টাইপিং):** JS একটি **ডায়নামিকালি টাইপড (Dynamically Typed)** ল্যাঙ্গুয়েজ। এর মানে হলো, ভেরিয়েবলের টাইপ (যেমন number, string, boolean) রানটাইমে (runtime) নির্ধারিত হয়, কোড লেখার সময় নয়।

JavaScript

```
let age = 30; // 'age' এখন একটি number

age = "ত্রিশ"; // কোনো সমস্যা নেই, 'age' এখন একটি string

age = { years: 30 }; // এটাও বৈধ, 'age' এখন একটি object
```

  - **সমস্যা:** এই নমনীয়তা ছোট স্ক্রিপ্টে সুবিধাজনক হলেও, বড় অ্যাপ্লিকেশনে এটি বাগ (bug)-এর প্রধান উৎস। আপনি ভুলবশত একটি ফাংশনে string-এর জায়গায় number পাস করতে পারেন, এবং এই এররটি রানটাইমে (যখন ইউজার অ্যাপটি ব্যবহার করছে) ধরা পড়বে, কম্পাইল-টাইমে নয়।

- **TypeScript (স্ট্যাটিক টাইপিং):** TS একটি **স্ট্যাটিকালি টাইপড (Statically Typed)** ল্যাঙ্গুয়েজ। আপনি কোড লেখার সময়ই ভেরিয়েবল, ফাংশন প্যারামিটার এবং রিটার্ন ভ্যালুর টাইপ ডিফাইন করে দেন।

TypeScript

```
let age: number = 30; // 'age'-কে number হিসেবে টাইপ ডিফাইন করা হলো

age = "ত্রিশ"; // !!! কম্পাইল-টাইম এরর !!!

// Error: Type 'string' is not assignable to type 'number'.
```

  - **সুবিধা:** TS কম্পাইলার কোড রান করার আগেই টাইপ-জনিত ভুলগুলো ধরে ফেলে। এটি **টাইপ সেফটি (Type Safety)** নিশ্চিত করে, যা কোডকে আরও নির্ভরযোগ্য (reliable) এবং মেইনটেইন (maintainable) করা সহজ করে তোলে।

১. কম্পাইলেশন এবং ট্রান্সপাইলেশন (Compilation & Transpilation)

এটি একটি গুরুত্বপূর্ণ কনসেপ্ট যা অনেক জুনিয়র ডেভেলপার ভুল বোঝে।

- **JavaScript:** ব্রাউজার বা Node.js এনভায়রনমেন্ট সরাসরি JS কোড বুঝতে এবং রান করতে পারে।
- **TypeScript:** ব্রাউজার TypeScript কোড সরাসরি **বোঝে না**।

TS কোডকে প্রথমে **ট্রান্সপাইল (Transpile)** করতে হয়। TS কম্পাইলার (tsc) আপনার লেখা TS কোড (.ts ফাইল) ইনপুট হিসেবে নেয় এবং একে সাধারণ, ক্লিন JavaScript কোডে (.js ফাইল) রূপান্তর করে। এই .js ফাইলটিই ব্রাউজারে বা সার্ভারে রান হয়।

গুরুত্বপূর্ণ ধারণা: টাইপ ইরেজার (Type Erasure) ট্রান্সপাইলেশন প্রসেসের সময়, TS তার সমস্ত টাইপ-সম্পর্কিত তথ্য (যেমন :number, interface) মুছে ফেলে। কারণ রানটাইমে জাভাস্ক্রিপ্টের টাইপের কোনো প্রয়োজন নেই। টাইপের কাজ শুধু ডেভেলপমেন্ট এবং কম্পাইল-টাইমে এরর চেক করা।

উদাহরণ:

TypeScript কোড (main.ts):

TypeScript

```
function greet(name: string): string {
  return "Hello, " + name;
}
let user: string = "Alice";
console.log(greet(user));
```

ট্রান্সপাইলড JavaScript কোড (main.js):

JavaScript

```
function greet(name) {
  return "Hello, " + name;
}
let user = "Alice";
console.log(greet(user));
```

দেখুন, আউটপুট JS কোডে কোনো :string টাইপ নেই।

২. TypeScript-এর এক্সক্লুসিভ ফিচার (যা JS-এ নেই)

TS শুধু টাইপ চেকিং-ই করে না, এটি এমন অনেক ফিচারও দেয় যা মডার্ন OOP (Object-Oriented Programming) এবং বড় প্রজেক্টের জন্য অপরিহার্য।

- **ইন্টারফেস (Interfaces):** অবজেক্টের 'শেপ' বা গঠন ডিফাইন করার জন্য ইন্টারফেস ব্যবহার করা হয়। এটি একটি ক্লিয়ার 'কন্ট্র্যাক্ট' (contract) তৈরি করে।

TypeScript

```
interface User {
  id: number;
  name: string;
  isActive: boolean;
  login(): void; // একটি মেথড
}

function promoteUser(user: User) {
  // ...
}
```

এখানে promoteUser ফাংশনটি নিশ্চিত করে যে আপনি এমন একটি অবজেক্ট পাস করছেন যাতে অবশ্যই id, name, isActive এবং login প্রপার্টি আছে।

- **জেনেরিক্স (Generics):** এটি রি-ইউজেবল (re-usable) কম্পোনেন্ট বা ফাংশন তৈরি করতে সাহায্য করে যা বিভিন্ন টাইপের সাথে কাজ করতে পারে, কিন্তু তারপরও টাইপ-সেফ থাকে।

TypeScript

```
// T হলো একটি জেনেরিক টাইপ প্লেসহোল্ডার
function getFirstElement<T>(arr: T[]): T {
  return arr[0];
```

```
}

let firstNum = getFirstElement([1, 2, 3]); // firstNum-এর টাইপ হবে 'number'
let firstStr = getFirstElement(["a", "b", "c"]); // firstStr-এর টাইপ হবে 'string'
```

- **এনামস (Enums):** নির্দিষ্ট কিছু ভ্যালুর সেট ডিফাইন করার জন্য ব্যবহৃত হয়।

TypeScript

```
enum UserRole {
  ADMIN,
  EDITOR,
  GUEST
}

if (user.role === UserRole.ADMIN) {
  // ...
}
```

- **অ্যাক্সেস মডিফায়ার (Access Modifiers):** ক্লাসের প্রপার্টি বা মেথডের ভিজিবিলিটি কন্ট্রোল করে (OOP-এর প্রধান কনসেপ্ট)।
  - public: সব জায়গা থেকে অ্যাক্সেসযোগ্য (ডিফল্ট)।
  - private: শুধু ক্লাসের ভেতর থেকে অ্যাক্সেসযোগ্য।
  - protected: ক্লাস এবং এর চাইল্ড ক্লাস থেকে অ্যাক্সেসযোগ্য।

TypeScript

```
class Employee {
  private salary: number;
  public name: string;

  constructor(name: string, salary: number) {
```

```
  this.name = name;
  this.salary = salary;
 }
}
const emp = new Employee("Bob", 50000);
console.log(emp.name); // বৈধ
// console.log(emp.salary); // !!! এরর: 'salary' is private.
```

(উল্লেখ্য, JS-এ এখন # ব্যবহার করে প্রাইভেট ফিল্ড তৈরি করা যায়, কিন্তু TS-এর সিস্টেমটি বেশি ফ্লেক্সিবল)।

৩. টুলিং এবং ডেভেলপার এক্সপেরিয়েন্স (Tooling & DX)

একজন ইঞ্জিনিয়ারের জন্য এটি অন্যতম বড় সুবিধা। যেহেতু TS আপনার কোডের প্রতিটি ভেরিয়েবলের টাইপ জানে, তাই কোড এডিটর (যেমন VS Code) আপনাকে অসাধারণ সাপোর্ট দিতে পারে:

- **অটো-কমপ্লিশন (Auto-completion):** আপনি একটি অবজেক্টের পর . টাইপ করলেই সেই অবজেক্টের সব প্রপার্টি এবং মেথড দেখতে পাবেন।
- **এরর হাইলাইটিং:** কোড রান করার আগেই টাইপ-জনিত ভুলগুলো লাল দাগ দিয়ে দেখিয়ে দেয়।
- **সহজ রিফ্যাক্টরিং (Refactoring):** কোনো ফাংশনের নাম বা প্যারামিটার পরিবর্তন করলে, এডিটর প্রজেক্টের সব জায়গায় তা স্বয়ংক্রিয়ভাবে আপডেট করে দিতে পারে। JS-এ এটি প্রায় অসম্ভব।

কখন কোনটা ব্যবহার করবেন?

- **JavaScript ব্যবহার করুন:**
    - ছোট প্রজেক্ট বা স্ক্রিপ্ট।
    - দ্রুত প্রোটোটাইপিং।
    - DOM ম্যানিপুলেশনের জন্য ভ্যানিলা (vanilla) কোড।
    - যখন আপনার টিমের TS-এ দক্ষতা নেই এবং শেখার সময়ও কম।
- **TypeScript ব্যবহার করুন:**

- মাঝারি থেকে বড় স্কেলের (enterprise-level) অ্যাপ্লিকেশন।
- যখন একাধিক ডেভেলপার একই কোডবেজে কাজ করে।
- যখন কোডের **মেইনটেনেবিলিটি (Maintainability)** এবং **স্টেবিলিটি (Stability)** খুব জরুরি।
- ফ্রেমওয়ার্ক যেমন **Angular** (যা TS-এ নির্মিত) বা **NestJS** ব্যবহার করলে।
- React বা Vue-এর বড় প্রজেক্টেও TS এখন স্ট্যান্ডার্ড প্র্যাকটিস।