Chapter 1: Giới thiệu TypeScript và Các Kiểu Dữ Liệu Cơ Bản

Mô tả tổng quát

Chào mừng bạn đến với hành trình khám phá **TypeScript**! Chapter này là bước khởi đầu giúp bạn hiểu TypeScript là gì, tại sao nó trở thành công cụ mạnh mẽ trong phát triển web hiện đại, và cách sử dụng các kiểu dữ liệu cơ bản. Nếu bạn đã quen với JavaScript, TypeScript sẽ là một bước tiến lớn giúp code của bạn an toàn, dễ bảo trì, và dễ mở rộng hơn. Chúng ta sẽ:

- Tìm hiểu TypeScript và so sánh với JavaScript.
- Cài đặt môi trường TypeScript và cấu hình dự án cơ bản.
- Làm quen với các kiểu dữ liệu cơ bản và cách khai báo biến.
- Thực hành qua các ví dụ và bài tập.

Mục tiêu: Sau chapter này, bạn sẽ viết được các đoạn code TypeScript cơ bản, hiểu lợi ích của kiểu tĩnh, và sẵn sàng cho các khái niệm nâng cao hơn.

Tiêu đề Chapter

TypeScript là gì? Tại sao nên dùng? Các kiểu dữ liệu cơ bản.

Lý thuyết chính

1. TypeScript là gì? So sánh với JavaScript

Định nghĩa

TypeScript là một ngôn ngữ lập trình mã nguồn mở, được phát triển bởi Microsoft. Nó là một **superset của JavaScript**, nghĩa là mọi code JavaScript hợp lệ đều chạy được trong

TypeScript. Tuy nhiên, TypeScript bổ sung **kiểu tĩnh (static typing)** và các tính năng như interfaces, enums, generics, giúp code dễ quản lý hơn, đặc biệt trong các dự án lớn.

- **Cách hoạt động**: Code TypeScript (`.ts` hoặc `.tsx` cho React) được **biên dịch (transpiled)** thành JavaScript (`.js`) bằng trình biên dịch `tsc` . File JavaScript này có thể chạy trên trình duyệt hoặc Node.js.
- **Tích hợp dần dần**: Bạn có thể thêm TypeScript vào dự án JavaScript hiện có mà không cần viết lại toàn bộ.

So sánh JavaScript và TypeScript

Tiêu chí	JavaScript	TypeScript
Kiểu dữ liệu	Động (dynamically typed), kiểu được xác định tại runtime.	Tĩnh tùy chọn (optional static typing), kiểm tra kiểu tại compile-time.
Kiểm tra lỗi	Lỗi kiểu chỉ phát hiện khi chạy (runtime).	Phát hiện lỗi kiểu ngay khi biên dịch (compile-time).
Ưu điểm	Linh hoạt, dễ học, cộng đồng lớn.	An toàn hơn, dễ bảo trì, hỗ trợ công cụ phát triển mạnh mẽ (autocompletion).
Nhược điểm	Dễ gặp lỗi kiểu trong dự án lớn (truyền sai kiểu, truy cập thuộc tính sai).	Cần thời gian học thêm cú pháp kiểu và cấu hình.

Ví dụ minh họa lỗi trong JavaScript:

```
function greet(name) {
    return "Hello, " + name.toUpperCase();
}
greet(123); // Runtime error: name.toUpperCase is not a function
```

TypeScript phát hiện lỗi ngay khi biên dịch:

```
function greet(name: string) {
    return "Hello, " + name.toUpperCase();
}
greet(123); // Lỗi biên dịch: Argument of type 'number' is not assignable to
greet("Alice"); // OK
```

2. Ưu điểm của TypeScript

- **Kiểm tra kiểu tĩnh**: Phát hiện lỗi trước khi chạy, như truyền sai kiểu dữ liệu.
- **Code dễ đọc, dễ bảo trì**: Kiểu dữ liệu rõ ràng giúp đội nhóm hiểu code nhanh hơn.
- **Công cụ phát triển mạnh mẽ**: Tích hợp tốt với VS Code, cung cấp:
  - Tự động hoàn thành (autocompletion).
  - Gợi ý kiểu (type hinting).
  - Báo lỗi trực tiếp khi viết code.
  - Tái cấu trúc an toàn (refactoring).
- **Hỗ trợ ECMAScript mới nhất**: TypeScript luôn cập nhật các tính năng JavaScript mới.
- **Phù hợp dự án lớn**: Giảm lỗi, dễ mở rộng trong các dự án phức tạp như ứng dụng React/Next.js.

3. Cài đặt TypeScript và cấu hình `tsconfig.json`

Yêu cầu

- Cài đặt **Node.js** và **npm** (hoặc **yarn**).
- Trình soạn thảo hỗ trợ TypeScript (khuyến nghị: **VS Code**).

Cài đặt TypeScript

Chạy lệnh sau để cài TypeScript toàn cục:

```
npm install -g typescript
tsc -v  // Kiểm tra phiên bản (ví dụ: 5.5.x tính đến 2025)
```

Biên dịch file TypeScript đầu tiên

Tạo file `app.ts`:

```
let message: string = "Chào mừng bạn đến với TypeScript!";
console.log(message);
```

Biên dịch: `tsc app.ts` (tạo ra `app.js`).
Chạy: `node app.js`.

Cấu hình `tsconfig.json`

Tạo file cấu hình bằng lệnh: `tsc --init`. File `tsconfig.json` chứa các tùy chọn quan trọng trong `compilerOptions`:

- `target`: Phiên bản JavaScript đầu ra (ví dụ: `"ES2020"` cho các trình duyệt hiện đại hoặc `"ES5"` cho tương thích ngược).
- `module`: Hệ thống module (ví dụ: `"ESNext"` cho ES Modules, `"CommonJS"` cho Node.js).
- `outDir`: Thư mục chứa file `.js` đầu ra (ví dụ: `"./dist"`).
- `rootDir`: Thư mục chứa file `.ts` nguồn (ví dụ: `"./src"`).
- `strict`: Bật tất cả kiểm tra kiểu nghiêm ngặt (khuyến khích `true`):
  - `noImplicitAny`: Báo lỗi nếu biến không có kiểu rõ ràng.
  - `strictNullChecks`: Ép kiểm tra `null` / `undefined` chặt chẽ.
- `esModuleInterop`: Cải thiện tương thích giữa ES Modules và CommonJS.
- `sourceMap`: Tạo file `.map` để debug dễ dàng.

Ví dụ `tsconfig.json` cơ bản:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "ESNext",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "sourceMap": true
  }
}
```

Chạy `tsc` để biên dịch toàn bộ dự án hoặc `tsc -w` để theo dõi thay đổi tự động.

4. Các kiểu dữ liệu cơ bản (Primitive Types)

TypeScript cung cấp các kiểu cơ bản để định nghĩa giá trị của biến, tham số, hoặc giá trị trả về của hàm.

`string`

Chuỗi ký tự (bao gồm text, ký tự Unicode).

```
let fullName: string = "Nguyễn Văn A";
```

`number`

Số (nguyên hoặc thực, không phân biệt `int` / `float`).

```
let age: number = 30.5;
```

`boolean`

Giá trị logic (`true` hoặc `false`).

```
let isActive: boolean = true;
```

`null` và `undefined`

- `null` : Biểu thị sự vắng mặt cố ý của giá trị.
- `undefined` : Biến chưa được gán giá trị.

Lưu ý: Với `strictNullChecks` , bạn phải sử dụng **union types** để cho phép `null` hoặc `undefined` :

```
let userAddress: string | null = null; // Union type: string hoặc null
let lastLoginDate: Date | undefined;   // Date hoặc undefined
```

`any`

Kiểu "thoát hiểm", cho phép gán bất kỳ giá trị nào và vô hiệu hóa kiểm tra kiểu. **Hạn chế sử dụng** vì mất đi lợi ích của TypeScript.

```
let anything: any = "Chuỗi";
anything = 100;
anything.doSomething(); // Không lỗi biên dịch, nhưng có thể lỗi runtime
```

`unknown`

An toàn hơn `any` . Nhận mọi giá trị, nhưng phải **kiểm tra kiểu (type narrowing)** trước khi sử dụng.

```ts
let valueFromApi: unknown = "Dữ liệu";
if (typeof valueFromApi === "string") {
    console.log(valueFromApi.toUpperCase()); // OK
} else {
    console.log("Không phải chuỗi");
}
```

`void`

Dành cho hàm không trả về giá trị.

```ts
function logMessage(message: string): void {
    console.log(message);
}
```

`never`

Dành cho hàm không bao giờ hoàn thành bình thường (throw lỗi, vòng lặp vô tận).

```ts
function throwError(message: string): never {
    throw new Error(message);
}
```

5. Union Types và Type Narrowing

Union Types

Cho phép một biến có thể thuộc nhiều kiểu khác nhau, sử dụng dấu `|`.

```typescript
let result: string | number;
result = "Success"; // OK
result = 200;       // OK
// result = true;   // Lỗi
```

Type Narrowing

Kiểm tra kiểu để thu hẹp phạm vi kiểu của biến, đảm bảo sử dụng an toàn.

```typescript
function processInput(input: string | number): void {
    if (typeof input === "string") {
        console.log(`Chuỗi: ${input.toUpperCase()}`);
    } else {
        console.log(`Số: ${input * 2}`);
    }
}
processInput("Hello"); // Chuỗi: HELLO
processInput(10);      // Số: 20
```

6. Khai báo biến và suy luận kiểu

- **Khai báo tường minh**: Chỉ định kiểu rõ ràng.

```typescript
let studentName: string = "Bùi Văn C";
const studentAge: number = 22;
```

- **Suy luận kiểu (Type Inference)**: TypeScript tự động suy ra kiểu nếu gán giá trị ngay.

```typescript
let teacherName = "Trần Thị D"; // TypeScript suy luận là string
```

```
// teacherName = 100; // Lỗi: Type 'number' is not assignable to type 'strin
```

Thực tiễn tốt nhất:

- Dùng `const` khi giá trị không thay đổi.
- Dùng `let` khi cần thay đổi giá trị.
- Tránh `var` vì phạm vi (scope) dễ gây lỗi.
- Khai báo kiểu tường minh cho tham số hàm và giá stricter trả về để code rõ ràng hơn.

7. Code ví dụ chính

Dưới đây là đoạn code tổng hợp, minh họa các khái niệm đã học, kèm chú thích chi tiết:

```
// 1. Khai báo biến với kiểu cơ bản và type inference
let frameworkName: string = "TypeScript"; // Kiểu tường minh
let version = 5.5; // TypeScript suy luận là number
let isLearning: boolean = true;
let projectDescription: string | undefined; // Cho phép undefined

console.log(`Chào mừng đến với ${frameworkName} phiên bản ${version}!`);
console.log(`Bạn đang học? ${isLearning}`);

// 2. Union type với null
let authorName: string | null = "Một tác giả";
console.log(`Tác giả: ${authorName}`);
authorName = null; // OK vì đã khai báo union type
console.log(`Tác giả sau khi gán null: ${authorName}`);

// 3. Kiểu any - Hạn chế sử dụng
let flexibleVar: any = "Có thể là bất cứ thứ gì";
console.log(`flexibleVar (string): ${flexibleVar}`);
flexibleVar = 2025;
console.log(`flexibleVar (number): ${flexibleVar}`);
flexibleVar = { topic: "TypeScript Basics" };
console.log(`flexibleVar (object): ${flexibleVar.topic}`);

// 4. Kiểu unknown - An toàn hơn any
```

```typescript
let userInput: unknown = "Dữ liệu người dùng";
if (typeof userInput === "string") {
    console.log(`Độ dài chuỗi userInput: ${userInput.length}`); // OK
} else {
    console.log("userInput không phải là chuỗi.");
}
userInput = 12345;
if (typeof userInput === "number") {
    console.log(`userInput nhân 2: ${userInput * 2}`); // OK
}

// 5. Kiểu void
function displayCurrentTime(): void {
    const now = new Date();
    console.log(`Thời gian hiện tại: ${now.toLocaleTimeString()}`);
}
displayCurrentTime();

// 6. Kiểu never
function criticalError(message: string): never {
    console.error("Lỗi nghiêm trọng!");
    throw new Error(message);
}

// 7. Phát hiện lỗi kiểu
function calculateSum(a: number, b: number): number {
    return a + b;
}
let num1: number = 10;
let num2: number = 5;
console.log(`Tổng ${num1} và ${num2} là: ${calculateSum(num1, num2)}`);
// console.log(calculateSum(num1, "5")); // Lỗi biên dịch
```

8. Danh sách bài tập

Bài tập 1: Trắc nghiệm - Mục đích chính của TypeScript

Câu hỏi: Mục đích chính của TypeScript là gì?

- A. Viết code chạy nhanh hơn JavaScript.

- B. Thêm kiểu tĩnh vào JavaScript, giúp phát hiện lỗi sớm và cải thiện bảo trì code.
- C. Thay thế hoàn toàn JavaScript trong phát triển web.
- D. Giảm kích thước file JavaScript sau biên dịch.

Đáp án: B

Bài tập 2: Trắc nghiệm - Sự khác biệt giữa `any` và `unknown`

Câu hỏi: Phát biểu nào mô tả đúng nhất sự khác biệt giữa `any` và `unknown`?

- A. Cả `any` và `unknown` đều không cho phép thao tác nếu chưa kiểm tra kiểu.
- B. `any` cho phép mọi thao tác mà không cần kiểm tra kiểu, `unknown` yêu cầu kiểm tra kiểu trước.
- C. `unknown` là bí danh của `any`.
- D. `unknown` chỉ nhận `null` hoặc `undefined`.

Đáp án: B

Bài tập 3: Code - Khai báo thông tin sản phẩm

Yêu cầu:

- Khai báo biến `productName: string` = "Laptop XYZ".
- Khai báo biến `productPrice: number` = 25000000.
- Khai báo biến `isAvailable: boolean` = true.
- In các giá trị ra console.

Giải pháp mẫu:

```
let productName: string = "Laptop XYZ";
let productPrice: number = 25000000;
let isAvailable: boolean = true;

console.log(`Tên sản phẩm: ${productName}`);
```

```
console.log(`Giá: ${productPrice}`);
console.log(`Còn hàng: ${isAvailable}`);
```

Bài tập 4: Code - Viết hàm chào hỏi

Yêu cầu: Viết hàm `sayHello` nhận tham số `userName: string`, trả về `void`, in ra "Xin chào, [userName]!".

Giải pháp mẫu:

```
function sayHello(userName: string): void {
    console.log(`Xin chào, ${userName}!`);
}
sayHello("Mai");
```

Bài tập 5: Code - Sửa lỗi kiểu

Đoạn code gốc:

```
function calculateTotalPrice(quantity, price) {
    return quantity * price;
}
let itemCount = "5";
let itemPrice = 10000;
let total = calculateTotalPrice(itemCount, itemPrice);
console.log("Tổng tiền:", total);
```

Giải pháp sửa lỗi:

```
function calculateTotalPrice(quantity: number, price: number): number {
    return quantity * price;
}
```

```
let itemCount: number = 5; // Sửa từ string thành number
let itemPrice: number = 10000;
let total: number = calculateTotalPrice(itemCount, itemPrice);
console.log("Tổng tiền:", total);
```

Bài tập 6: Code - Xử lý Union Types

Yêu cầu: Viết hàm `displayValue` nhận tham số `value: string | number`. Nếu là `string`, in ra chuỗi in hoa. Nếu là `number`, in ra số nhân đôi.

Giải pháp mẫu:

```
function displayValue(value: string | number): void {
    if (typeof value === "string") {
        console.log(`Chuỗi in hoa: ${value.toUpperCase()}`);
    } else {
        console.log(`Số nhân đôi: ${value * 2}`);
    }
}
displayValue("Hello"); // Chuỗi in hoa: HELLO
displayValue(10);      // Số nhân đôi: 20
```