Introduktion till TypeScript

- **Programmeringsspråk** som är en **superset av JavaScript**.
- **Lanserades 2012** av **Microsoft**.
- **Öppen källkod**.
- **Transpileras till JavaScript**.

Fördelar med TypeScript

- **Minskar risken för fel** genom att använda typer.
- **Transpileras** (omvandlas) till JavaScript.
- **Fångar många fel redan vid transpilation**.
- **Stöd för modern utveckling** genom bättre verktyg och Intellisense.

TypeScript Playground

- Man kan testa TypeScript direkt i webbläsaren med **TypeScript Playground**: https://www.typescriptlang.org/play/

Skapa en första TypeScript-fil

- Använd **filändelsen** `.ts` istället för `.js`.

Sätta upp ett TypeScript-projekt

Skapa en ny projekt-mapp:

```
npm install typescript --save-dev  # Installera TypeScript
npx tsc --init  # Skapa en tsconfig.json
```

Konfigurera `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "es6",  // Kompilera till ECMAScript 6
    "module": "commonjs",  // Modulhantering för Node.js
    "outDir": "./dist",  // Output-mapp
    "rootDir": "./src",  // Källkodsmapp
    "strict": true,  // Aktivera strikt typkontroll - använd denna!
    "esModuleInterop": true,  // Bättre stöd för ES-moduler
    "forceConsistentCasingInFileNames": true,  // Känslighet för filnamn
    "moduleResolution": "node",  // Modulhantering enligt Node.js
    "resolveJsonModule": true,  // Tillåt import av JSON
    "sourceMap": true  // Skapa källkartor för debugging
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "**/*.spec.ts"]
}
```

Första fil

Lägg till en `index.ts` -fil med TypeScript-kod.

```
//tex:
const hello:string = "Hello World";
console.log(hello);
```

Kompilera koden

```
npx tsc  # Transpilera TypeScript till JavaScript
```

- filen `index.js` skapas med JavaScript-kod
- kan köras med kommandot:

```
node index.js
```

Använda `ts-node`

För att köra TypeScript i **Node.js**:

```
npm install ts-node --save-dev  # Installera ts-node
npx ts-node app.ts  # Köra TypeScript-koden direkt utan transpilering
```

Om du använder **Nodemon**, så fungerar det automatiskt med `ts-node`.

TypeScript Type checking

- Tack vare denna visas fel i koden när du utvecklar.
- Felet visas med en röd linje under koden
- Ett meddelande med info visas vid mouse hover

Deklaration av variabler i TypeScript

- Använd **modern deklaration** med `const` och `let`.
- TypeScript har stöd för stark typning

```
const isStudent: boolean = true;
const startYear: number = 2021;
const firstname: string = "Håkan";
```

- **Valfritt:** TypeScript kan automatiskt inferera dessa typer

```
let someValue = "Detta är en sträng"; // Automatisk typning som string
let strLength: number = someValue.length;
```

Type Assertion

Om du vet typen bättre än TypeScript kan du **ange den manuellt**.

```
let someValue: unknown = "Detta är en sträng";

let strLength: number = (someValue as string).length;
```

Arrayer och Listor

TypeScript stöder två syntaxer för att deklarera arrayer, denna är vanligast:

```
const names: string[] = ["Hans", "Greta", "Gun"];
const years: number[] = [2004, 2007, 2018];
```

- **Valfritt:** TypeScript kan automatiskt inferera dessa typer.

Union Types

Variabler kan ha **flera möjliga typer**.

```
let something: number | string | boolean;

something = 1; // OK
something = "1"; // OK
something = true; // OK
something = {}; // Fel: Objekt ej tillåtet
```

Tuples

Tuples används för **typsäkrade fält**.

```
let employee: [number, string] = [1, "Magda"];
console.log(employee[0])// 1
```

Enums

Enums skapar en lista av **konstanta värden**.

```
enum Color { Blue, Yellow, Red };
const color: Color = Color.Blue;
```

Funktioner i TypeScript

TypeScript stöder **typsatta parametrar och returvärden**.

```
const isEmailValid = (email: string): boolean => email.includes("@");
```

- NOT: Parametrar måste vara typsatta när man kör strict-mode och det ska ni använda.
- Returvärden måste oftast inte vara typsatta

Funktioner utan returvärde (`void`)

```
const logMessage = (message: string): void => {
  console.log(message);
};
```

Valfria parametrar

Parametrar kan göras **valfria** med `?`.

```
const greet = (name, greeting?) => {
  console.log(greeting ? `${greeting}, ${name}!` : `Hello, ${name}!`);
};
```

Typning av objekt

Objekt kan definieras med **type eller interface**.

```
type Person = {
  firstname: string;
  lastname: string;
  birthYear?: number; // Valfri egenskap
};
```

```
interface Person = {
  firstname: string;
  lastname: string;
  birthYear?: number; // Valfri egenskap
};
```

Skapa en instans av Person

```
const person: Person = {
  firstname: "Jonatan",
  lastname: "Hallenberg"
};
```

NOT: variabler i objekt måste vara typsatta när man kör strict-mode

Skillnad mellan `type` och `interface`

- `interface` **kan utökas**, medan `type` **kan användas för unions**.
- Båda är lika funktionella – **välj en och håll dig till den**.
- Jag kommer att köra på `type`