TypeScript – це надбудова над JavaScript, яка додає статичну типізацію та нові можливості для розробників. Його основна мета – полегшення створення великих та складних застосунків, підвищення надійності та спрощення підтримки коду.

**Налаштування TypeScript**

**Інсталяція TypeScript**

*npm install -g typescript*

**Створення TypeScript файлу**

Файл має розширення **.ts**, наприклад, **app.ts**.

**Компіляція TypeScript у JavaScript**

*tsc app.ts*

**TypeScript конфігурація (tsconfig.json)**

```
{
  "compilerOptions": {
    "target": "es6", // Визначає версію JavaScript, у яку буде компілюватися TypeScript. "es6" означає ECMAScript 2015
    "module": "commonjs", // Визначає тип модуля, який буде використовуватися під час компіляції. "commonjs" використовується в Node.js
    "strict": true, // Вмикає суворий режим перевірки типів
```

```
    "outDir": "./dist", // Визначає папку, куди компілятор TypeScript збере згенерований JavaScript
    "rootDir": "./src" // Вказує папку з вихідними файлами TypeScript
  }
}
```

**Основи синтаксису**

**Типи даних**

TypeScript додає типізацію змінних

```
let isDone: boolean = false;
let age: number = 25;
let userName: string = "John";
let list: number[] = [1, 2, 3];
let tuple: [string, number] = ["hello", 10];
```

**Необов'язкові параметри та значення за замовчуванням**

Функції в TypeScript можуть мати необов'язкові параметри та параметри зі значеннями за замовчуванням

```
function greet(name: string, age?: number): string {
  return age ? `Hello, ${name}. You are ${age} years old.` : `Hello, ${name}.`;
}

console.log(greet("Alice")); // "Hello, Alice."
console.log(greet("Alice", 30)); // "Hello, Alice. You are 30 years old."
```

```
function multiply(a: number, b: number = 2): number {
  return a * b;
}

console.log(multiply(5)); // 10
console.log(multiply(5, 3)); // 15
```

**ООП у TypeScript**

**Геттери та сеттери**

TypeScript підтримує геттери та сеттери для забезпечення контролю доступу до властивостей класу

```
class Employee {
  private _salary: number;

  constructor(salary: number) {
    this._salary = salary;
  }

  get salary(): number {
    return this._salary;
  }

  set salary(value: number) {
    if (value < 0) {
      throw new Error("Salary cannot be negative.");
    }
    this._salary = value;
```

```
  }
}

let emp = new Employee(5000);
console.log(emp.salary); // 5000
emp.salary = 6000;
console.log(emp.salary); // 6000
```

**Наслідування**

```
class Person {
  name: string;

  constructor(name: string) {
    this.name = name;
  }

  introduce(): void {
    console.log(`Hi, my name is ${this.name}.`);
  }
}

class Employee extends Person {
  position: string;

  constructor(name: string, position: string) {
    super(name);
    this.position = position;
  }
```

```
  work(): void {
    console.log(`${this.name} is working as a ${this.position}.`);
  }
}

let emp = new Employee("Alice", "Developer");
emp.introduce(); // "Hi, my name is Alice."
emp.work(); // "Alice is working as a Developer."
```

**Абстрактні класи**

```
abstract class Shape {
  abstract calculateArea(): number;

  describe(): void {
    console.log("This is a shape.");
  }
}

class Circle extends Shape {
  radius: number;

  constructor(radius: number) {
    super();
    this.radius = radius;
  }
```

```
  calculateArea(): number {
    return Math.PI * this.radius ** 2;
  }
}

let circle = new Circle(5);
circle.describe(); // "This is a shape."
console.log(circle.calculateArea()); // 78.53981633974483
```

**Інтерфейси**

Інтерфейси дозволяють визначити структуру даних або контракт, який повинен виконувати об'єкт або клас. Інтерфейси підтримують наслідування

```
interface Animal {
  name: string;
  age: number;
  makeSound(): void;
}

class Dog implements Animal {
  name: string;
  age: number;

  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }

  makeSound(): void {
```

```
    console.log("Woof!");
  }
}

let dog = new Dog("Buddy", 4);
dog.makeSound(); // "Woof!"
```

Інтерфейси можуть бути розширені

```
interface Person {
  name: string;
  age: number;
}

interface Employee extends Person {
  position: string;
}

let employee: Employee = {
  name: "Alice",
  age: 30,
  position: "Developer"
};
```

**Умовні конструкції та цикли**

TypeScript підтримує ті ж самі умовні конструкції та цикли, що й JavaScript

**Умовні конструкції**

```
let age = 18;

if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

```
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("Start of the week.");
    break;
  case "Friday":
    console.log("Almost weekend.");
    break;
  default:
    console.log("Regular day.");
}
```

**Цикли**

```
// For loop
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

```
// While loop
let count = 0;
while (count < 5) {
  console.log(count);
  count++;
}
```

```
// Do-while loop
let num = 0;
do {
  console.log(num);
  num++;
} while (num < 5);
```

```
// For-of loop
let array = [1, 2, 3, 4, 5];
for (let value of array) {
  console.log(value);
}
```

```
// For-in loop
let obj = { a: 1, b: 2, c: 3 };
for (let key in obj) {
  console.log(`${key}: ${obj[key]}`);
}
```