TypeScript and JavaScript are closely related languages, with TypeScript being a superset of JavaScript. Here are five key differences between TypeScript and JavaScript, with examples:

1. **Static Typing vs. Dynamic Typing**:
   - **JavaScript**: JavaScript is dynamically typed, which means variables can change their type during runtime and there is no static type checking.

   Example:
   ```javascript
   let x = 42; // x is a number
   x = 'hello'; // x is now a string
   ```

   - **TypeScript**: TypeScript introduces static typing, allowing you to specify types for variables, function parameters, and return values. This helps catch type-related errors at compile time.

   Example:
   ```typescript
   let x: number = 42; // x is explicitly a number
   // The following line will cause a compile-time error because 'hello' is a string, not a number
   // x = 'hello';
   ```

2. **Type Inference**:
   - **JavaScript**: JavaScript does not infer types; variables are declared without specifying their types.

   Example:
   ```javascript
   let name = 'Alice'; // name is a string, but JavaScript doesn't enforce this
   ```

   - **TypeScript**: TypeScript supports type inference, automatically inferring the type of a variable based on its initialization value.

   Example:
   ```typescript
   let name = 'Alice'; // TypeScript infers the type as string
   ```

3. **Interfaces**:
   - **JavaScript**: JavaScript does not support interfaces natively.

Example:
```javascript
const person = {
    name: 'Alice',
    age: 30
};
```

- **TypeScript**: TypeScript introduces interfaces, which allow you to define the shape of objects and specify the types of properties they should contain.

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

const person: Person = {
    name: 'Alice',
    age: 30
};
```

4. **Classes and Access Modifiers**:
   - **JavaScript**: JavaScript supports classes (added in ES6) but does not have access modifiers like `private` or `protected`.

Example:
```javascript
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        console.log(`${this.name} makes a noise`);
    }
}
```

- **TypeScript**: TypeScript enhances classes with access modifiers such as `private`, `protected`, and `public` to control the visibility of class members.

Example:
```typescript
class Animal {
    private name: string;

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

    speak() {
        console.log(`${this.name} makes a noise`);
    }
}
```

5. **Tooling and IDE Support**:
   - **JavaScript**: JavaScript has limited IDE support for type checking and code navigation because of its dynamic typing.

   Example:
   ```javascript
   let x = 'hello';
   console.log(x.length); // This will work, but there might not be IDE suggestions for methods on a string
   ```

   - **TypeScript**: TypeScript offers enhanced tooling and IDE support due to static typing. You can benefit from features such as autocompletion, type checking, and code navigation.

   Example:
   ```typescript
   let x: string = 'hello';
   console.log(x.length); // TypeScript provides IDE suggestions for methods on a string
   ```

These differences illustrate how TypeScript extends JavaScript with static typing, interfaces, and other features to enhance code quality, maintainability, and tooling support.