1. Identify Files for Conversion:

Start by identifying the JavaScript files you want to convert to TypeScript. In this case, we'll focus on app.js.

2. Install TypeScript:

Open your terminal and navigate to your project directory. Install TypeScript globally using npm or yarn:

Bash

```
npm install -g typescript
```

3. Create tsconfig.json (Optional):

While not strictly necessary for basic conversion, creating a tsconfig.json file allows you to configure the TypeScript compiler. This file can reside in your project root directory. Here's a basic configuration:

JSON

```
{
  "compilerOptions": {
    "target": "es5",  // Set the target ECMAScript version
    "strict": true   // Enable stricter type checking
  }
}
```

4. Rename JavaScript File:

Change the extension of app.js to app.ts. This signifies that the file now contains TypeScript code.

5. Update Existing Code:

Now, open app.ts and start adding type annotations to your existing JavaScript code. Here's an example:

Before (JavaScript):

JavaScript

```
function greet(name) {
  console.log("Hello, " + name);
}
```

greet("Alice");

After (TypeScript):

TypeScript

```
function greet(name: string): void { // Add type annotations
  console.log("Hello, " + name);
}

greet("Alice"); // Type inferred from string literal
```

We added type annotations to the greet function parameters and return type.

TypeScript infers the type of the argument ("Alice") as a string.

6. Handling Existing Code Patterns:

Some JavaScript code patterns might require adjustments for TypeScript:

Variable Declarations: Annotate variables with their expected types:

TypeScript

```
let message: string = "Hello"; // Type annotation for string
```

Implicit any Type: TypeScript avoids implicit any type whenever possible. If a variable can hold different types, consider using a union type (covered later).

7. Resolving Conversion Issues:

As you add type annotations, you might encounter errors from the TypeScript compiler. These errors highlight potential type mismatches. Fix these errors by ensuring types align throughout your code.

Type Errors: These errors point out inconsistencies between assigned values and expected types.

Type Casting: In some cases, you might need type casting to explicitly convert a value to a specific type. However, this is generally discouraged as it reduces type safety.

8. Compiling to JavaScript:

Once you've converted your code and addressed errors, you can compile it to standard JavaScript using the TypeScript compiler:

Bash

tsc app.ts

This will create a new file, typically app.js, containing the compiled JavaScript code.

9. Gradual Conversion:

You can convert your project incrementally, focusing on one file at a time. This allows you to test and debug the conversion process gradually.

Additional TypeScript Features:

As you explore further, consider utilizing more advanced TypeScript features like:

Interfaces: Define object structures and contracts.

Classes: Create reusable blueprints for objects.

Generics: Write reusable components that work with various data types.

Union and Intersection Types: Combine or refine types for complex scenarios.

.