TypeScript: The Basics

Exercise 1: Basic TypeScript Syntax with Coordinates

Objective: Understand basic TypeScript syntax and type checking.
Task:

1. Create a TypeScript file called `coordinates.ts`.
2. Define a constant `latitude` and `longitude` with sample values representing coordinates.
3. Write a function `getCoordinateString` that takes latitude and longitude as parameters and returns a formatted string.
4. Add type annotations to the function parameters to ensure they are numbers.

Exercise 2: Type Checking with Spatial Functions

Objective: Learn how to use TypeScript's type system to catch errors.
Task:

1. Create a TypeScript file called `distance.ts`.
2. Write a function `calculateDistance` that takes two sets of coordinates and calculates the distance between them using the Haversine formula.
3. Ensure the function parameters are properly typed.
4. Try calling the function with incorrect types and observe TypeScript's error messages.

Exercise 3: Handling Optional Properties in Spatial Data

Objective: Understand how to handle optional properties using TypeScript.
Task:

1. Create a TypeScript file called `optionalProperties.ts`.
2. Define an interface `Coordinate` that includes `latitude` and `longitude` properties, and optionally a `label` property.
3. Write a function `printCoordinate` that takes a `Coordinate` object and prints its properties.
4. Handle the optional `label` property in the function.

Exercise 4: Using TypeScript with Spatial Libraries

Objective: Integrate TypeScript with a spatial library to enhance type safety.

Task:

1. Install the `geolib` library using npm.
2. Create a TypeScript file called `geolibExample.ts`.
3. Use `geolib` to calculate the distance between two points and find the midpoint.
4. Ensure proper type annotations and handle potential errors.

Exercise 5: Implementing Static Type Checking with Spatial Data

Objective: Utilize TypeScript's static type-checking to prevent runtime errors in spatial calculations.

Task:

1. Create a TypeScript file called `staticTypeChecking.ts`.
2. Define an interface `Point` with `x` and `y` properties representing coordinates.
3. Write a function `calculateSlope` that takes two points and calculates the slope of the line connecting them.
4. Use TypeScript to catch potential errors before running the code.

Exercise 6: Using TypeScript with GeoJSON[1]

Objective: Apply TypeScript to work with GeoJSON data structures.

Task:

[1] https://geojson.org/

1. Create a TypeScript file called `geojsonExample.ts`.
2. Define a type for a GeoJSON Point feature.
3. Write a function `printGeoJSONPoint` that takes a GeoJSON Point and prints its coordinates.
4. Ensure proper type annotations and handle potential errors.