Exercise 1: Install TypeScript and Compile a File

1. **Install TypeScript globally:**

```
npm install -g typescript
```

2. **Initialize a TypeScript project to create a `tsconfig.json` file:**

```
tsc --init
```

3. **Open the `tsconfig.json` file and review its contents. Try to understand what each configuration option does.**

4. **Create a new TypeScript file named `index.ts` and define a simple function `greet` that takes a string parameter `name` and returns a greeting message. Use `console.log` to print the greeting message for "World".**

```
function greet(name: string): string {
    return `Hello, ${name}!`;
}

console.log(greet('World'));
```

5. **Compile the TypeScript file to JavaScript:**

```
tsc index.ts
```

6. **Check if a JavaScript file (`index.js`) was created. Open this file and try to understand how TypeScript was transpiled into JavaScript.**

7. **Run the compiled JavaScript file using Node.js:**

```
node index.js
```

8. **Create a new TypeScript file named `hello.ts` and define a function `greet` that takes a string parameter `name` and returns a greeting message. Use `console.log` to print the greeting message for "World".**

9. **Compile the TypeScript file to JavaScript using `tsc`.**

10. **Run the compiled JavaScript file using `node`.**

Exercise 2: Basic Types and Type Annotations

1. **Create a TypeScript file named `basicTypes.ts` and define variables with the following types:**

   - `isDone` of type `boolean`
   - `age` of type `number`
   - `firstName` of type `string`
   - `numbers` of type `array of numbers`
   - `tuple` of type `string and number`

2. **Define a function named `square` that takes a number parameter and returns its square.**

3. **Call the function and use `console.log` to print the result.**

Exercise 3: Creating and Using Interfaces

1. **Create an interface named `Person` with the following properties:**

   - `firstName` of type `string`

- `lastName` of type `string`
- `age` of type `number`
- `greet` method that receives nothing and returns a string

2. **Create a variable of type `Person` and implement the `greet` method to return a greeting message.**

3. **Use `console.log` to print the greeting message.**

Exercise 4: Classes and Inheritance

1. **Create a class named `Animal` with a constructor that initializes a `name` property and a method `move` that takes a distance parameter (default value 0) and logs a message `*animalName* moved *distance* amount of meters` .**

2. **Create a subclass named `Dog` that extends `Animal` and has an additional method `bark` that logs `Woof woof!`. Override the `move` method to log a different message `dog named *animalName* moved *distance* amount of meters` and then call the `move` method from the parent class.**

3. **Create an instance of `Dog`, call the `bark` method, and then call the `move` method.**

Exercise 5: Working with Arrays and Tuples

1. **Create an array of numbers and write a function named `sumArray` that takes the array and returns the sum of its elements.**

2. **Create a tuple that holds a string and an array of numbers. Write a function named `logTuple` that logs each element of the tuple.**

Exercise 6: Advanced Object with Methods

1. **Create an object `ComplexObject` with the following properties:**

   - `num1` of type `number`
   - `num2` of type `number`
   - `str1` of type `string`
   - `str2` of type `string`
   - `numArray` of type `array of numbers`

2. **Add the following methods to the object:**

   - `logValues` that takes two parameters (`num` of type `number` and `strArray` of type `array of strings`) and logs them.
   - `addNumbers` that returns the sum of `num1` and `num2`.
   - `getStrings` that returns the concatenation of `str1` and `str2`.
   - `getArrayLength` that returns the length of `numArray`.

USE GOOGLE

If you are stuck and unsure about the syntax, use google and find it.