**TypeScript**

**General**

1. **Static methods or variables** are not accessible inside function as well as we can’t use static method/ variables in class functions with **‘this’ keyword**. Static property is not available in the class instance.
2. **This** keyword points to the **current instance** we are working with.
3. **Static** properties are **detached from instances**. Use class name to utilize static properties.
4. Marking a method **abstract** will cause its implementation to be compulsory in every child and also must **mark the class** **with abstract keyword.** 
   1. E.g., abstract describe(department: string) : void;
   2. No function body no nothing. Now this describe method is compulsory for all the sub classes.
   3. Sub classes does not have to mark with abstract keyword but the parent class should.
5. Index properties (also known as **index signatures**) in TypeScript allow you to define the shape of an object when the names of its properties are not known in advance but the types of their values are known.
   1. Syntax: type TypeName = { [key: KeyType]: ValueType; };
   2. Example:
      1. type MixedDictionary = {

[key: string]: number;

id: number; // Specific properties can coexist with index signatures

};

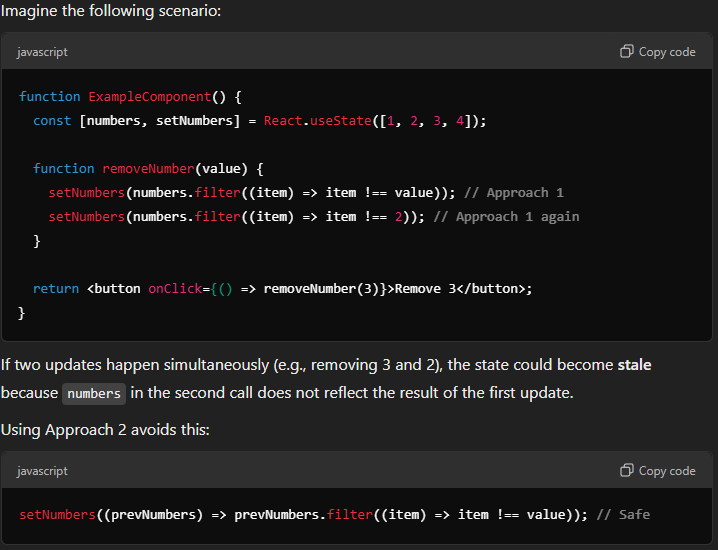
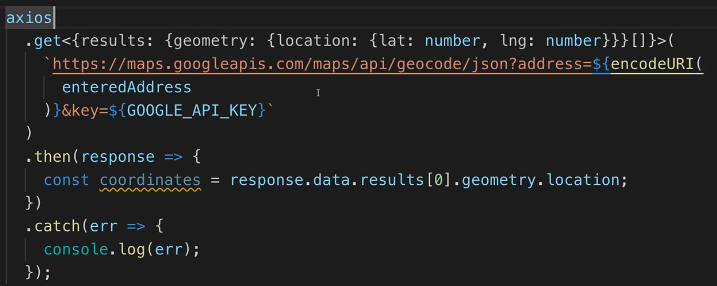
const mixed: MixedDictionary = {

id: 101,

age: 25,

};

* 1. You can add much as properties you want just make sure if **matches the *key* and *value* Type**.

1. The **nullish coalescing operator (??)** is used to provide a default value when the left-hand side is null or undefined.
   1. Syntax: const result = value ?? defaultValue;
2. || treats falsy values (0, false, "") as equivalent to null or undefined. But ‘??’ does not.
3. In TypeScript, you can attach **multiple decorators** to a single class, and they are evaluated **in reverse order** (from bottom to top).
4. Decorators are more famous in angular than react.
5. Decorators are activated when you define a class, no need to instantiate a class to call decorators.
6. When taking props from another component we need to define the type of the prop which can be included when we define the type of the functional component that is React.FC.
   1. Defining types of props will be like React.FC<propTypes>.
7. @type of event for form is **React.FormEvent.**
8. If we are not monitoring user input bases on keystroke then we can use useRef() instead of useState().
   1. Value={state} be replaced with ref={textInputRef}
   2. @type of ref is <React.HTMLInputElement>
   3. And input value will be taken as textInputRef.current!.value
   4. Current!. Because at the time when function will build or called before getting triggered by user the no ref would have be assigned and there will be no current.value
   5. This is the way (!) to assure TS that do not worry the value will be coming.
9. Updating State in React is suggested to be like setState( prevState => prevState.map() ) because in every scenario (when we have state queues or a singular setState call) we will be working with most recent state.
   1. 
10. encodeURI() 🡺 this will convert string into URL compatible string. When working with API’s you often want to send some data along with the URL to get customized response.
    1. Sometime these responses will have character which are not supported by URL.
    2. With encodeURI(<userEnteredString) will be modified into URL compatible format.
11. Axios.get() have generic types so we can define the @type of data we are expecting,
    1. 
    2. Axios supports generics for the request and response to provide type safety.
    3. Get, post, put, delete, patch and request these are some axios methods supporting generic types.
12. Optional @type check in interface. This means I may or may not be getting the data in this exact format but If I do the @type check must be done:
    1. interface ApiResponse {

id: number;

name: string;

status?: "ok" | "error"; // Optional property

}

1. When working with Google maps you have to mention CLI and use Google object in your Typescript file.
   1. But will get error as TS does know Google object will exist or not. With CLI running Google will be available as window object globally but TS does not know that.
   2. Bad Way: Use declare var google: any;
   3. Good Way: install types for Google maps and TS will start supporting Google objects.
2. sd