Typescript Course

What is Typescript?

- It is a JS superset: This means that it is a language that builds upon JS. It takes the JS language and adds features/advantages to it. Makes your JS more powerful and easier. HOWEVER it can't be executed by JS environments like the browser.
- Typescript is a programming tool which is a compiler run over code to compile Typescript code to JS. Although you didn't write in JS it compiles and translates it to JS.
- The TS features are compiled to JS workarounds, possible errors are thrown so it is not completely doing magic.

What does it do?
It adds TYPES to JS which gives you a chance to identify errors in the code earlier rather than running the script and finding them at runtime in the browser. It also allows for extra error checking where they can be caught and fixed during development.

Advantages of using Typescript

- Makes writing clean code much easier
- Adds the option for types and makes us be much more explicit when writing our code to tell it what to do
- We can use next-gen JavaScript features which can be compiled down to JS down for older browsers. (Use modern JS features which can still ship and work at older generation browsers)
- We can add non JS features like Interfaces or Generics which can help us during development which are not available when using simply JS
- Gives us certain meta-programming features such as decorators (cover later)

CONTENT

To compile:
tsc ___.ts

Using Core Types

Type Inference: TS does its best to try and understand which type you have in a certain var or const.

- **Number**: There is no special type for ints or floats in Typescript like other OO languages.
- **String**: Text. All text values are read the same.
- **Boolean**: True/False. There are no truthy/falsy values in TS/JS
- **Object**: Any JS object is of type object. There are more specific types of objects in TS though e.g requirements
- **Array**: Can store anything in them (numbers, strings, objects, etc). Any JS array is supported and the types can be flexible or strict.
- **Tuple**: An array of a fixed length but also fixed type. This could be handy if you wanted an array with maybe a single string and a single num only.
- **Enum**: Gives you an enumerated list where you have human readable labels you can use in your code.

- **Any**: Any kind of value, no specific type assignment necessary. This makes sure the TS compiler can't check anything (so not always the best use case).

More advanced topics

- **Union Types:** If we have some place in our application where we accept two different values in a function for example then a union type can help. This can tell TS that either calls are ok.
- **Literal Types:** Types where you don't say a certain var or param should hold a number or a string but you are clear on the exact VALUE it should hold.
- **Type Aliases:** Can be cumbersome to have to repeat the union type. You can create a new type which can store these union types by creating type aliases.
- **Function Types:** Types that describe a function, regarding params and return value. Created using arrow function notation. They allow us to describe which type of function we want to use somewhere.
- **Unknown Type**: Used if we are not sure what value of type of value we want to store in it and time of declaration. Should probably be used over any if there is a use case where you are considering both.
- **Never Type:** It is basically a function that throws an error or infinite loop where your code crashes or wants to alert the user to an error.

Compilation

- **Watch mode:** To enter watch mode where your TS file will compile each time it is saved (instead of using the terminal command) enter: **tsc __.ts -w**
- **Compiling a whole project:** Initialise a whole project and tell TS that everything in the directory where you run this command, will be watched for any changes. Creates a config file: **tsc —init.** Once we do this we just have to enter **tsc.** It will compile any TS files in the directory.
- **Exclude files**: e.g In the ts.config file simply enter

```
"exclude": [
        "node_modules"
        ]
}
```

Same goes for include. Only listed files will be compiled.