Day 4 MCQ Assignment, Practicing Typescript

1. **Microsoft**

Microsoft created and maintains TypeScript as an open-source programming anguage. It can be used to develop JS applications in both client and server side.

2. **Node**

Commonly used with node.js which runs javascript on server side and other backend technologies as well.

3. **JavaScript**

Type Script is superset of JavaScript that adds static typing. After compilation TypeScript is converted/compiled to JavaScript.

4. **extends**

Inheritance is implemented using the extends keyword similar to JavaScript Inheritance

5. **var x = "string";**

The variable has no type defined after declaring the variable there should be a colon space type.

This a JavaScript syntax, not a TypeScript syntax

6. **var x: number = 999;**

This is TypeScript syntax,

The variable x of number type is stored the value 999.

7 .**ts**

TypeScript files are typically stored in .ts format, for JavaScript it is .js

8. **tsc filename.ts**

tsc is the command to run TypeScript compiler to compile it to .js file, we can also executa a .ts file using node tsc filename.ts

9. **tsc filename.ts -w**

Make the compiler continuously watch for changes in TypeScript files and recompile automatically when changes are detected.

10. **super()**

use the super() keyword within the child class's constructor.

```
class Parent {
  constructor(public name: string) {
    console.log("Parent constructor called");
  }
}

class Child extends Parent {
  constructor(name: string, public age: number) {
    super(name);
    console.log("Child constructor called");
  }
}
const child = new Child("John", 25);
```

1. **Sort**

Practiced on how to use the sort function efficiently.

```
vendors.sort((a, b) => a.id - b.id);
```

2. **Filter**

Go through each item in the products array.

For every item (p), check if its price is less than or equal to maxPrice.

Only the products that pass this test will be included in the result.

filter() is a built-in method that returns a new array of items that match a condition.

```
products.filter(p => p.price <= maxPrice);
```

3. **Finding**

It stops at the first match.

If found, it prints the details

If not found, it says “Product not found.”

```
products.find(p => p.name === productName);
```

4. **Every/Some**

every - checks if all items match the condition.

some - checks if at least one item matches.

```
const allUnder100 = products.every(p => p.price < 100);
```

```
const anyUnder30 = products.some(p => p.price < 30);
```

5. **Mapping**

map() goes through every product.

For each product, it returns just the name.

You get a new array of only names (original array stays the same).

```
// Map to get only names

const productNames = products.map(p => p.name);
```

6. **Reduce**

(sum, p) => sum + p.price \\Adds the current product's price to the total sum.

0 is the starting value of the sum.

It goes through all items and returns one final result:

```
const totalPrice = products.reduce((sum, p) => sum + p.price, 0);
```

7. **Decorators/Modules**

Decorators = add logic to classes/functions automatically.

```
// Enable "experimentalDecorators": true in tsconfig.json

function Logger(constructor: Function) {
  console.log("Class created:", constructor.name);
}

@Logger
class Product {
  constructor(public name: string, public price: number) {}
}

const p1 = new Product("Book", 100);
```

Modules = split code into files and share using export / import

mathUtils.ts (a separate file)

```
export function add(a: number, b: number): number {
  return a + b;
}
export const PI = 3.14;
```

app.ts

```
import { add, PI } from './mathUtils';
let result = add(10, 5);
console.log(` Result: ${result}` );
console.log(` Value of PI: ${PI}` );
```