**TypeScript(DAY 3)**

1. **Keyof Type (with Generics):**

**Description:**

The **keyof** type operator, combined with generics, allows for safe property access on objects by ensuring only valid keys are used.

**Example Code:**

function getProperty<T, K extends keyof T>(obj: T, key: K) {

return obj[key];

}

console.log(getProperty(person, "gender"));

console.log(getProperty(bmw, "name"));

1. **Rest Parameters & reduce Function:**

**Description:**

Rest parameters (...args) allow a function to accept any number of arguments as an array. The reduce() method can be used to process that array into a single result (e.g., sum).

**Example Code:**

function sum(...numbers: number[]): number {

let total = 0;

for(let num of numbers) {

total += num;

}

return total;

}

const res = sum(1, 2, 3);

console.log(res);

**return numbers.reduce((total, num) => total + num, 0);**

* numbers: number[] collects all arguments into an array
* Loop or reduce() can be used to process them

1. **Function Overloading**

**Description:**

Function overloading allows you to define multiple signatures for a function, enhancing type safety and flexibility when the function handles different types.

**Example Code:**

class Meow {

speak(s: string): string;

speak(n: number): string;

speak(m: boolean): string;

speak(arg: any): any {

if(typeof(arg) === 'number') {

return "meow number";

}

if(typeof(arg) === 'string') {

return "meow string";

}

if(typeof(arg) === 'boolean') {

return "meow boolean";

}

}

}

let m2 = new Meow();

console.log(m2.speak(10));

console.log(m2.speak("Aaryan"));

console.log(m2.speak('vishal'));

console.log(m2.speak(true));

* Multiple speak(...) signatures are declared
* Only one implementation is provided, handling all types
* This makes your class **type-safe** and **flexible**

1. **Modules:**

**Description:**

* In TypeScript, modules are files that contain code (variables, classes, functions, etc.) and export them so they can be reused in other files. This modular approach improves code organization, maintainability, and reuse.
* Each file in TypeScript is treated as a module if it contains at least one **import or export** statement.

**Export:**

The export keyword is used to expose functions, classes, objects, or variables from a module so they can be imported elsewhere.

**Export default:**

This is used to export a **single value** or entity from a file. A file can have **only one default export**.  
  
**Examples:**

// File: m1/meow.ts

export default function speak() {

return "meow";

}  
------------------------------------------------------------------------  
// File: mex.ts

export class A {

cname: string;

cver: string;

}

export function name1(n: string): string {

return n;

}

In this file, a class A and a function name1 are **named exports**. These are imported using curly braces.

**Import:**

Use the import keyword to bring in functionality from another module.

**Examples**:

// File: main.ts

import { name1, A } from "./mex"; // Named imports

import speak from "./m1/meow"; // Default import

console.log(name1("Naveen")); // Output: Naveen

console.log(speak()); // Output: meow

* name1 and A are imported using named import syntax.
* speak is imported as a default export, which does **not** use curly braces.