**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. **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**