```
//Typescript


// Open - source programming language by Microsoft. built on top of javascript.

// Superset of JavaScript,

// statically typed - bcz js is dynamically typed (determine the dataype during run time)

// we can catch the error during compile time


//.ts => transpiler  ==> .js


//let n=10;

// n="10";

// Math.floor(n); // need unit test to find the error


//medium to large prject


// It is a superset of JavaScript, which means that all valid JavaScript code is also valid TypeScript
code.

// However, TypeScript adds optional static typing to JavaScript,

// providing developers with the ability to define and

// enforce types for variables, function parameters, and return values.


// Key Features:


// Static Typing: Allows developers to specify variable types for early error detection.

// ECMAScript Compatibility: Aligns with JavaScript standards.

// OOP Support: Supports classes, interfaces, and other object - oriented features.

// Tooling: Comes with a compiler(tsc) and great IDE support(e.g., Visual Studio Code).

// Code Readability: Enhances code self - documentation through explicit type definitions.



// Advantages:
```

```
// Early error detection.
// Object - oriented features for building scalable applications.
// Improved code readability and maintainability.



// const btn: HTMLElement | null = document.getElementById('Button1');

// if (btn) {
//     btn.addEventListener('click', () => {
//         alert("Click");
//     });
// }




{
    let name1: String = "Darshan";
    console.log(name1);
}

//! configuration
// tsc --init

//enable this json keys - target
//"module": "commonjs",
// "rootDir": "./",   -- src folder
// "outDir": "./dist",            -- outDir
// "removeComments": true,
// "noEmitOnError": true,       -- if any errors during compile time but ts still generates the js file
```

```
//run
//tsc

//! How to debug?
//enable -  "sourceMap": true
//

let a1: number = 10;
if (a1 > 8) {
   a1++;
}
console.log(a1);


// create -- node js lauch.js - inside add key  - "preLaunchTask": "tsc: build - tsconfig.json",
// then if u press f5 it will generate index.js.map


//! Intro
let n: number | string = 10;
n: "10";
console.log(n);
function test(msg: string) {
   console.log(msg);
}

test("Hello");

//Arrays
let numbers: number[] = [1, 2, 3, 45];
numbers.forEach(n => n.toString);
```

```
console.log(numbers);

//tuples
// fixed datatypes and values
let user: [number, string] = [10, "Darshan"];
user.push(1);
console.log(user[0]);

//enum

// constants -- small by default -1
// medium -
//large -

// const small=0;
// const medium=1;
// const large=2;

const enum Size { small, medium, large };
let mySize: Size = Size.medium;
console.log(mySize);



//exactly number of aruments
// we can add ? -
function calulates(a: number, b: number): number {
  //   let x;
  if (a < 10) {
    return a + b;
  }
  return a;
```

```
}
console.log(calulates(10, 20));

//default value
function calulateSum(a: number, b = 20): number {
   if (a < 10) {
      return a + b;
   }
   return a;

}
console.log(calulateSum(10));




//object
//type - we can use it in multiple areas
type empS = {
   name: String,
   age: number,
   email?: string,
   display: () => void;

}
let emp: empS = {
   name: "Darshan",
   age: 10,
   display: function () {
      console.log(this.name);
   }
```

```
}
console.log(emp);
emp.display();


//union

function dis(n: number | string): number {
    if (typeof n == "number") {
        return n * 2;
    } else {
        return parseInt(n) * 10;
    }
}
console.log(dis(10));
console.log(dis("10"));


//intersection
type Draggable = {
    drag: () => void
}

type Resizable = {
    resize?: () => void
}


type UIWid = Draggable & Resizable;

let textBox: UIWid = {
  drag: () => {
    console.log("Hi");
  },
  resize: () => {
    console.log("Hello");
  }
}
console.log(textBox);



//! interface


interface Person {
  name: string,
  age: number
}
interface person1 extends Person {
  city: string
}
let myInfo: person1 = {
  city: "H",
  name: "D",
  age: 10
}

console.log(myInfo.city);



//! literal type
type quantity = 50 | 100;
let quantity1: quantity = 50;
```

```
console.log(quantity1);


//! How to handle null or undefiined values?
function greet(name: string | null) {
    if (name) {
        console.log(name.toLowerCase());
    } else {
        console.log("Hola");
    }
}
greet(null);


//optional property access operator




//optional element access operator
//we have an array
// customers?.[0]






import * as readline from 'readline';


const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});


rl.question('Enter your Name: ', (name) => {
  rl.question("Enter your birth year", (year) => {
    console.log("Name: " + name + "Age is " + printAge(parseInt(year)));
    rl.close();
  })
});

function printAge(year: number): number {
  let n: Date = new Date();
  return n.getFullYear() - year;
}



// React

// import React from 'react';
// const App: React.FC = () => {
//    return (
//        <div>
//        <h1>Hello World < /h1>
//            < /div>
//    )}

// export default App;



//! UseState

// import React from 'react';
// const App: React.FC = () => {
// const [state, setSate]=useState<String>("");
//    return (
//        <div>
//        <h1>{state}< /h1>
//          < /div>
//    )}

// export default App;


//! props
// import React from 'react';
// import Child from './child';

// interface arrInterface{
// id:number;
// name:string;
// isCheck:boolean;
// }

// const App: React.FC = () => {
// const [state, setState]=useState<String>("");
// const [arr, setArr] = useState<arrInterface []>([])
// const inputRef = useRef<HTMLInputElement>(null)

// let handleAdd=(e:React.FormEvent):void=>{
// e.preventDefault();
// }

//    return (
//        <div>
//        <h1>Test< /h1>
// <input ref={inputRef} type="text" value={state} onChange={(e)=>setValue(e.target.value)}/>
// <Child state={ state } setState = { setState } handleAdd={handleAdd}/>
//          < /div>
//    )}
// export default App;


//* Child comp

// interface props {
//    state: String,
//    setState: React.Dispatch<React.SetStateAction<string>>
// handleAdd: (e:React.FormEvent) => void;
// }

// const Child: React.FC <props> = ({ state, setState, handleAdd }: props) => {
//    return (
//        <div>
//            { state }
//        </div>
//    )


// }
// export default Child;
```