**DAY – 3(Typescript)**

**Typescript Installation commands:**

* node -v
* npm -v
* npm init -y
* npm install typescript --save-dev
* npx tsc –init

**Commands to run typescript:**

* npx tsc index.ts
* node index.js

**package.json**:

* Defines project metadata, dependencies, scripts, and other configurations.

**package-lock.json**:

* Locks down exact versions of dependencies to ensure consistent installation

**tsconfig.json**:

* Configures the TypeScript compiler, specifying how to process and compile TypeScript files.

**Functions:**

**Definition:** Functions are reusable blocks of code that perform specific tasks and can accept parameters and return results.   
**Examples:** 1.

function sum(num: number | number[]) {

if(Array.isArray(num)) {

let sum: number = 0;

num.forEach(e => sum += e);

return sum;

}

return num;

}

**Examples:** 2.

function getFacts(url: string): Promise<Response> {

return fetch(url).then(res => res.json());

}

**Loops:**

**Definition:** Loops are used to execute a block of code multiple times, with different values or conditions, until a specified condition is met.

**Examples:** 1.

vendors.forEach(e => {

console.log(`ID: ${e.id}, Name: ${e.name}, Product: ${e.productName}, Price: ${e.price}`);

});

**Examples:** 2.

for (let i = 0; i < vendors.length - 1; i++) {

for (let j = 0; j < vendors.length - i - 1; j++) {

if (vendors[j].id > vendors[j + 1].id) {

const temp = vendors[j];

vendors[j] = vendors[j + 1];

vendors[j + 1] = temp;

}

}

}

**Interfaces:**

**Definition:** Interfaces define the structure of an object, specifying the required properties and their types.

**Examples:** 1.

interface User {

name: string;

id: number;

}

**Examples:** 2.

interface User1 {

name: string;

id: number;

role: roles;}

**Classes:**

**Definition:** Classes are blueprints for creating objects. They encapsulate data for the object and provide methods for behavior.

**Examples:** 1.

class UserAccount {

name: string;

id: number;

constructor(name: string, id: number) {

this.name = name;

this.id = id;

}

}

**Examples:** 2.

class UserAcc {

name: string;

id: number;

role: roles;

constructor(name: string, id: number, role: roles) {

this.name = name;

this.id = id;

this.role = role;

}

}  
  
**Arrays:**

**Definition:** Arrays are ordered collections of values that can store multiple elements of any type.

**Examples: 1.**

const vendors: Vendor[] = [

{ id: 3, name: "Naveen", productName: "Watch", price: 600 },

{ id: 1, name: "Vishal", productName: "Airpods", price: 750 },

{ id: 2, name: "Ranjith", productName: "Tv", price: 800 }

];

**Examples: 2.**

const numbers: number[] = [1, 2, 3, 4, 5];

**Conditional Statements:**

**Definition:** Conditional statements allow the program to make decisions based on certain conditions (true or false).

**Examples:** 1.

if (Array.isArray(num)) {

let sum: number = 0;

num.forEach(e => sum += e);

return sum;

}

**Examples: 2.**

if (vendors[j].id > vendors[j + 1].id) {

const temp = vendors[j];

vendors[j] = vendors[j + 1];

vendors[j + 1] = temp;

}

**Arrow Functions:**

**Definition:** Arrow functions are a shorthand syntax for writing functions, typically with shorter code and lexical this.

**Examples:** 1.

num.forEach(e => sum += e);

**Examples:** 2.

vendors.sort((v1, v2) => v1.id > v2.id ? -1 : 1);

**type –** Used to define custom types, allowing union types and type aliases.

**Examples:** 1.

type roles = "Trainer" | "Developer" | "Tester";

**Examples:** 2.

type Vendor = {

    id: number;

    name: string;

    lname?: string;

    productName: string;

    price: number;

  };  
  
**Type Assertions:**

**Definition:** Type assertions are used to tell TypeScript to treat a value as a specific type.

**Examples:** 1.

const user: User = new UserAccount("Aaryan", 1);

**Examples: 2.**

const user1: User1 = new UserAcc("Aaryan", 1, "Trainer");

**Optional Params in TypeScript:**

Optional parameters in TypeScript are parameters that may or may not be provided when creating an object or calling a function. These parameters are denoted by a question mark (?) after the parameter name.  
**Examples:1.**

function greet(vendor: Vendor) {

console.log(`Hello, ${vendor.name} ${vendor.lname ? vendor.lname : ''}`);

}

greet({ id: 1, name: "Acme Inc.", productName: "Gadget", price: 100 });

greet({ id: 2, name: "TechCorp", lname: "Smith", productName: "Widget", price: 150 });

**Examples:2.**

type Vendor = {

id: number;

name: string;

lname?: string; // optional params ?

productName: any; // don't know the type

price: number;

};