**Basic TypeScript Problems**

**1️. Variables & Types**

**Problem:**  
Declare the following variables with appropriate TypeScript types:

* A string variable userName with a value "John Doe".
* A number variable age with a value 25.
* A boolean variable isStudent with a value true.

**Answer:**

let userName: string = "John Doe";

let age: number = 25;

let isStudent: boolean = true;

**2️.Functions & Type Annotations**

**Problem:**  
Write a function add that takes two numbers as arguments and returns their sum.

typescriptCopyEditfunction add(a, b) {  
   return a + b;  
}

**Answer:**

function add(a: number, b: number): number {

return a + b;

}

✅ **Fix this function by adding TypeScript type annotations.**

**3️. Function with Optional Parameters**

**Problem:**  
Modify the function below to make the second parameter **optional**:

typescriptCopyEditfunction greet(name: string, message: string) {  
    return `${message}, ${name}!`;  
}

**Answer:**

function greet(name: string, message: string = "Hello"): string {

return `${message}, ${name}!`;

}

✅ If the message is not provided, default to "Hello".

**4️. Union Types**

**Problem:**  
Define a function displayId that accepts either **a number or a string** as an argument and logs it.

typescriptCopyEditfunction displayId(id) {  
    console.log(`ID: ${id}`);  
}

**Answer**

function displayId(id: number | string): void {

console.log(`ID: ${id}`);

}

✅ Fix this function using **union types**.

**5. Arrays & Tuples**

**Problem:**

* Declare an array numbers that only holds numbers.
* Create a tuple person that contains a string (name) and a number (age).

Example:

typescriptCopyEdit// numbers: [10, 20, 30]// person: ["Alice", 30]

**Answer**

let numbers: number[] = [10, 20, 30];

let person: [string, number] = ["Alice", 30];

**6️. Interfaces**

**Problem:**  
Define an interface Person with the following properties:

* name (string)
* age (number)
* email (string)

Create a variable user that implements this interface.

**Answer**

interface Person {

name: string;

age: number;

email: string;

}

const user: Person = {

name: "John",

age: 30,

email: "john@example.com"

};

**7. Type Aliases**

**Problem:**  
Create a **type alias** Employee that includes:

* id (number)
* name (string)
* role ("admin" | "manager" | "employee")

Define an employee variable of type Employee.

**Answer**

type Employee = {

id: number;

name: string;

role: "admin" | "manager" | "employee";

};

const employee: Employee = {

id: 1,

name: "Alice",

role: "manager"

};

**8️. Enum**

**Problem:**  
Define an enum Color with values: "Red", "Green", and "Blue".  
Create a variable favoriteColor of type Color.

**Answer**

enum Color {

Red = "Red",

Green = "Green",

Blue = "Blue"

}

let favoriteColor: Color = Color.Green;

**9️. Readonly & Partial**

**Problem:**

* Create a readonly object for user information.
* Use Partial<T> to make a type where all properties are optional.

Example:

typescriptCopyEditinterface User {  
   id: number;  
   name: string;  
   email: string;  
}  
const user: Readonly<User> = { id: 1, name: "Alice", email: "alice@example.com" };// user.name = "Bob"; ❌ This should give an error!let newUser: Partial<User> = { name: "Bob" }; // ✅ Allowed

**Answer**

interface User {

id: number;

name: string;

email: string;

}

const user: Readonly<User> = { id: 1, name: "Alice", email: "alice@example.com" };

let newUser: Partial<User> = { name: "Bob" };

**10. Classes in TypeScript**

**Problem:**  
Create a class Car with the following properties:

* brand (string)
* model (string)
* year (number)

Add a method getCarInfo() that returns a string like:  
"Toyota Corolla (2020)"

 Strongly Typed Variables & Functions  
Scenario:  
You're building a simple e-commerce application and need to store product details.

**Answer**

class Car {

constructor(public brand: string, public model: string, public year: number) {}

getCarInfo(): string {

return `${this.brand} ${this.model} (${this.year})`;

}

}

const myCar = new Car("Toyota", "Corolla", 2020);

console.log(myCar.getCarInfo()); // "Toyota Corolla (2020)"

👉 Exercise:

Define a Product type with properties:  
id (number)  
name (string)  
price (number)  
isAvailable (boolean)  
Create a function getDiscountedPrice(product: Product, discount: number): number that calculates the discounted price.

**Answer**

type Product = {

id: number;

name: string;

price: number;

isAvailable: boolean;

};

function getDiscountedPrice(product: Product, discount: number): number {

return product.price - product.price \* (discount / 100);

}

2️⃣ Union & Intersection Types  
Scenario:  
You need to create a function that formats user inputs, which can be a string, a number, or an array of strings.

**Answer**

function formatInput(input: string | number | string[]): string {

if (typeof input === "string") {

return input.toUpperCase();

} else if (typeof input === "number") {

return `$${input}`;

} else {

return input.join(", ");

}

}

 👉 Exercise:

Implement a function formatInput(input: string | number | string[]): string  
If it's a string, return it in uppercase.  
If it's a number, return it as a string with a $ prefix.  
If it's an array of strings, return a comma-separated string.  
3️⃣ Tuples & Enums  
Scenario:  
You're working on a ride-sharing app and need to store driver status and location.

**Answer**

enum DriverStatus {

Available,

OnTrip,

Offline

}

type DriverInfo = [number, string, DriverStatus];

const driver: DriverInfo = [101, "John Doe", DriverStatus.Available];