Typescript handson:

Why we need typescript?

1. In Javascript, type safety is not there. It supports dynamic typing. Example:

*var* i=20;

console.log(i);

*var* i="Raja";

console.log(i);

console.log("WOrking finee.");

But in typescript, it is static typing. Means, once declared then its type is static. Does not change.

In typescript: two.ts

*var* i=20;

*var* i="Raja";

we are not allowed to do like above code. The data type of the variable i cannot be dynamically changed.

1. Typescript is compiled.

This helps catch type-related errors at compile-time, which can prevent many bugs and make the code more robust and reliable.

TypeScript is a programming language and a superset of JavaScript that adds optional static typing to the language. It was created by Microsoft and has gained significant popularity among developers and companies for several reasons:

1. **Type Safety**: TypeScript introduces static typing, allowing developers to specify the types of variables, function parameters, and return values. This helps catch type-related errors at compile-time, which can prevent many bugs and make the code more robust and reliable.
2. **Code Maintainability**: With static typing, TypeScript provides better code readability and maintainability. The type annotations serve as documentation for the codebase, making it easier for developers to understand the intent of the code. Additionally, as projects grow larger and more complex, TypeScript helps in managing the codebase by enforcing clear contracts between functions and components.
3. **Intellisense and Tooling Support**: TypeScript provides excellent tooling support, including intelligent code completion, code navigation, and refactoring tools. This allows developers to be more productive and write code more efficiently, thanks to the code editor's ability to offer suggestions and catch errors in real-time.
4. **Improved Collaboration**: With TypeScript, teams can collaborate more effectively. Since the types are explicitly defined, team members can better understand each other's code, reducing misinterpretations and potential misunderstandings.
5. **Gradual Adoption**: TypeScript is designed to be compatible with existing JavaScript code. This means you can start using TypeScript in a project by incrementally adding type annotations to existing JavaScript code. This gradual adoption path makes it easier for developers and teams to transition to TypeScript without requiring a complete rewrite of their codebase.
6. **Future JavaScript Features**: TypeScript often incorporates upcoming JavaScript features and syntax, even before they are officially supported by browsers. This allows developers to use modern JavaScript features while transpiling the code to older versions of JavaScript that are compatible with all major browsers.
7. **Popular in the Ecosystem**: TypeScript has gained widespread adoption and is widely used in various large-scale projects and libraries. This means there is a rich ecosystem of tools, libraries, and resources available for developers using TypeScript.
8. **Strong Community Support**: TypeScript has a vibrant and active community that contributes to its development, maintains various packages, and provides support through forums and other online platforms.

Overall, TypeScript offers numerous advantages for JavaScript development, making it a powerful tool for building scalable, maintainable, and robust applications, especially in large and complex projects or teams. However, it's essential to consider your project's specific needs and the familiarity of your team with TypeScript before deciding to adopt it.

TypeScript supports the following data types:

1. \*\*Boolean\*\*: Represents a true or false value.

```typescript

let isDone: boolean = false;

```

2. \*\*Number\*\*: Represents both integer and floating-point numbers.

```typescript

let age: number = 30;

let pi: number = 3.14;

```

3. \*\*String\*\*: Represents a sequence of characters (text).

```typescript

let name: string = "John Doe";

```

4. \*\*Array\*\*: Represents a collection of elements of the same type. TypeScript supports arrays of specific types or a union of types.

```typescript

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

let fruits: Array<string> = ["apple", "banana", "orange"];

```

5. \*\*Tuple\*\*: Represents an array with a fixed number of elements, where each element can have a different type. The types of elements are known at specific indices.

```typescript

let employee: [string, number] = ["John Doe", 30];

```

6. \*\*Enum\*\*: Represents a set of named constants (enumeration).

```typescript

enum Color {

Red,

Green,

Blue,

}

let favoriteColor: Color = Color.Blue;

```

7. \*\*Any\*\*: Represents a value for which the type is not known or doesn't matter. It allows you to opt-out of type checking for specific variables.

```typescript

let dynamicValue: any = "This could be anything!";

```

8. \*\*Void\*\*: Represents the absence of a value. It is commonly used as the return type for functions that don't return a value.

```typescript

function logMessage(message: string): void {

console.log(message);

}

```

9. \*\*Null and Undefined\*\*: Represents null and undefined values, respectively.

```typescript

let someValue: null = null;

let anotherValue: undefined = undefined;

```

10. \*\*Never\*\*: Represents the type of values that never occur. It's often used to denote functions that throw exceptions or enter infinite loops.

```typescript

function throwError(message: string): never {

throw new Error(message);

}

```

11. \*\*Object\*\*: Represents non-primitive types (anything that is not number, string, boolean, null, or undefined). It is a type that describes any JavaScript object.

```typescript

let person: object = { name: "John Doe", age: 30 };

```

12. \*\*Union Types\*\*: Represents a type that can be one of multiple types. It is denoted by using the pipe (|) symbol.

```typescript

let numberOrString: number | string = 42;

numberOrString = "hello";

```

13. \*\*Intersection Types\*\*: Represents a type that combines multiple types. It is denoted by using the ampersand (&) symbol.

```typescript

type Point2D = { x: number; y: number };

type Point3D = { z: number };

let point: Point2D & Point3D = { x: 1, y: 2, z: 3 };

```

These are the core data types in TypeScript. Additionally, you can create your own custom types using interfaces, type aliases, and classes to represent complex data structures and models. TypeScript's strong type system enhances code reliability, maintainability, and readability, making it a popular choice for many developers and projects.

In typescript, we can declare a variable of multiple data types / types.