**Class 2 PIAIC**

**Steps to start a project:**

1. **npm install -g typescript** The command "npm install -g typescript" is used to globally install TypeScript on your computer using npm (Node Package Manager)
2. The **npm init -y or npm iniit** command is used to initialize a new Node.js project with a default configuration. It creates a **package.json** file in the current directory, which is a metadata file that contains information about the project and its dependencies.
3. The **tsc --init** command is used to initialize a TypeScript project and generate a **tsconfig.json** file in the current directory. The **tsconfig.json** file contains the configuration settings for the TypeScript compiler (tsc).
4. The **tsc index.ts** command is used to compile a TypeScript file (**index.ts**) into JavaScript using the TypeScript compiler (**tsc**).
5. The **node index** command is used to run a JavaScript file (**index.js**) using the Node.js runtime.
6. The correct command to run a JavaScript file named **index.js** using the Node.js runtime is indeed **node index.js**.
7. **Tsc** compile all the files in the project.
8. **Node.js** provides a built-in package management system called **npm (Node Package Manager)**. npm allows developers to easily manage and install packages, libraries, and dependencies for their Node.js projects.

**Annotations**

In TypeScript, variable annotations are used to explicitly specify the type of a variable. The TypeScript compiler uses these annotations to perform static type checking and provide type information during development and compilation. Here are the common ways to annotate variables in TypeScript:

**Type Annotations:**

* Variables can be explicitly annotated with their types using a **colon (:)** followed by the type. For example:

let name: **string** = "John";

let age: **number** = 25;

let isActive: **boolean** = true;

**Type Inference:**

* TypeScript's type inference system automatically infers the type of a variable based on its assigned value. In many cases, type inference can eliminate the need for explicit type annotations. For example:

let name = "John"; // type inference infers **name** as **string**

let age = 25; // type inference infers **age** as **number**

let isActive = true; // type inference infers **isActive** as **Boolean**

**Union Types:**

* Variables can have multiple possible types by using union types. Union types are specified using the pipe (‘**|’**) symbol. For example:

let value: string | number = "Hello"; // value can be a **string** or a **number**

**Type Aliases:**

* TypeScript allows you to create custom type aliases using the **type** keyword. Type aliases can be used to simplify complex types or create reusable type definitions. For example:

type Person = {

name: string;

age: number;

};

let person: Person = {

name: "John",

age: 25

};

**Datatypes**

1. **number:** Represents numeric values, including integers and floating-point numbers. Example: let age: number = 25;
2. **string:** Represents textual data. Example: let name: string = "John";
3. **boolean:** Represents a logical value, either true or false. Example: let isActive: boolean = true;
4. **array:** Represents an ordered collection of elements of the same type. Example: let numbers: number[] = [1, 2, 3, 4];
5. **tuple:** Represents an array with a fixed number of elements of different types. Example: let person: [string, number] = ["John", 25];
6. **enum**: Represents a set of named constants(set of possibilities). Example:

enum Color {

Red,

Green,

Blue

};

let favoriteColor: Color = Color.Blue;

7. **any**: Represents a dynamic or untyped value. Use with caution as it bypasses type checki ng. Example: **let data: any = "Hello";**

8. **void**: Represents the absence of a value. Typically used as the return type for functions that do not return a value. Example: **function logMessage(): void { console.log("Hello"); }**

9. **null** and **undefined**: Represents the absence of a value or an uninitialized value. Example:

let value1: null = null;

let value2: undefined = undefined;

10. **object**: Represents a non-primitive type (not **number**, **string**, **boolean**, **null**, or **undefined**). Example: **let person: object = { name: "John", age: 25 };**

11. **unknown**: Represents values that are of an unknown type. Similar to **any**, but with type checking enforced. Example: **let data: unknown = "Hello";**

**Template Literals**

Template literals, also known as template strings, are a feature in TypeScript that allows you to create strings with embedded expressions. Template literals are enclosed by backticks (```) instead of single or double quotes.

let name = "John";

let age = 25;

// Using template literals to create a string with embedded expressions

let message = **`**My name is **${name}** and I'm **${age}** years old.**`**;

console.log(message);

// Output: My name is John and I'm 25 years old.

***ASSIGNMET***

***Calculator using operators.***

TypeScript includes various operators that perform different operations on values or variables. Here are some commonly used operators in TypeScript:

1. Arithmetic Operators:
   * +: Addition operator.
   * -: Subtraction operator.
   * \*: Multiplication operator.
   * /: Division operator.
   * %: Modulo (remainder) operator.
   * ++: Increment operator.
   * --: Decrement operator.
2. Assignment Operators:
   * =: Simple assignment operator.
   * +=: Addition assignment operator.
   * -=: Subtraction assignment operator.
   * \*=: Multiplication assignment operator.
   * /=: Division assignment operator.
   * %=: Modulo assignment operator.
3. Comparison Operators:
   * ==: Equality operator (checks for value equality).
   * !=: Inequality operator (checks for value inequality).
   * ===: Strict equality operator (checks for both value and type equality).
   * !==: Strict inequality operator (checks for either value or type inequality).
   * >: Greater than operator.
   * <: Less than operator.
   * >=: Greater than or equal to operator.
   * <=: Less than or equal to operator.
4. Logical Operators:
   * &&: Logical AND operator.
   * ||: Logical OR operator.
   * !: Logical NOT operator.
5. Bitwise Operators:
   * &: Bitwise AND operator.
   * |: Bitwise OR operator.
   * ^: Bitwise XOR (exclusive OR) operator.
   * ~: Bitwise NOT operator.
   * <<: Left shift operator.
   * >>: Right shift operator.
   * >>>: Zero-fill right shift operator.
6. Ternary Operator:
   * condition ? expr1 : expr2: Ternary operator is a shorthand for if-else statements.