# **Learning TypeScript**

## ****1. Installation & Basic Commands****

* **Installing TypeScript globally**

TypeScript can be installed globally using npm to use the tsc compiler from any directory.

* + Command: npm i typescript -g
* **Initializing TypeScript configuration**

A tsconfig.json file is created in the project to store compiler options and settings.

* + Command: tsc --init
* **Compiling a TypeScript file**

Compile your app.ts file into app.js using:

* + Command: tsc app.ts
* **Enable watch mode**

Watches TypeScript files for changes and automatically recompiles them.

* + Command: tsc --watch

## ****2. Type Inference****

* **Definition:** TypeScript automatically infers the type of a variable or function based on the assigned value, even without explicit type annotation.
* **Examples:**
  + let a = 10; → inferred as **number**.
  + let str = "harsh"; → inferred as **string**.

## ****3. Type Annotation****

* **Definition:** Explicitly mentioning the type of a variable, parameter, or function return type.
* **Examples:**
  + let a: number = 10;
  + let str: string = "harsh";

## ****4. Using Type Annotation in Functions****

* **Example (addition function):**

typescript

CopyEdit

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

return a + b;

}

* + Here:
    - a: number → parameter type annotation
    - b: number → parameter type annotation
    - : number after parentheses → return type annotation
* **Void Return Type Example:**

typescript

CopyEdit

function greet(name: string): void {

console.log("Hello, " + name);

}

## ****5. Type Aliases****

* **Definition:** Creating a custom name for a type to improve readability and reusability.
* **Example:**

typescript

CopyEdit

type StringOrNumber = string | number;

let id: StringOrNumber = 123;

id = "ABC123"; // also valid

* **Function with type alias example:**

typescript

CopyEdit

type User = { name: string; age: number; };

function printUser(user: User): void {

console.log("Name: " + user.name + ", Age: " + user.age);

}

## ****6. Interface****

* **Definition:** Defines the structure of an object, enforcing specific properties with their types.
* **Example:**

typescript

CopyEdit

interface User {

name: string;

age: number;

}

function printUser(obj: User) {

console.log("name is " + obj.name + " and age is " + obj.age);

}

printUser({ name: "harsh", age: 22 });

* + **Wrong usage:** printUser({ name: "harsh", age: "22" }); // age should be number.

### ****Extending Interfaces****

* **Create a USER interface:**

typescript

CopyEdit

interface USER {

name: string;

email: string;

password: string;

age: number;

}

* **Extend USER with ADMIN:**

typescript

CopyEdit

interface ADMIN extends USER {

isAdmin: boolean;

}

* **Example objects:**

typescript

CopyEdit

const user: USER = {

name: "alpha",

age: 21,

email: "alpha@a.com",

password: "password"

};

const admin: ADMIN = {

name: "admin1",

age: 22,

email: "harsh@h.com",

password: "admin",

isAdmin: true

};

### ****Interface Merging****

* **Definition:** When two interfaces have the same name, they merge into a single interface with combined properties.
* **Example:**

typescript

CopyEdit

interface abab {

name: string;

}

interface abab {

email: string;

}

// Now abab type has both name and email properties.

## ****7. Class and Objects****

* **Definition:** A class is a blueprint for creating objects with properties and methods. Objects are instances of a class.
* **Example Device class:**

typescript

CopyEdit

class Device {

name = "dell";

price = 50000;

category = "laptop";

}

let d1 = new Device();

let d2 = new Device();

* + Both d1 and d2 have the same default properties.

### ****Using**** this ****in OOP****

* **Definition:** this refers to the current object of the class within its methods.
* **Example:**

typescript

CopyEdit

class Example {

name = "alpha";

displayName() {

console.log(this.name);

}

}

const obj = new Example();

obj.displayName(); // prints "alpha"

## ****8. Generics****

* **Definition:** Create reusable components, functions, or classes that work with different data types while ensuring type safety.
* **Example (generic function):**

typescript

CopyEdit

function ab<T>(val: T) {

console.log(val);

}

ab<string>("harsh");

ab<boolean>(true);

### ****Generics in OOP****

* **Example generic class:**

typescript

CopyEdit

class Box<T> {

content: T;

constructor(value: T) {

this.content = value;

}

display(): void {

console.log(this.content);

}

}

let b1 = new Box<string>("Hello");

let b2 = new Box<number>(100);

b1.display(); // Hello

b2.display(); // 100

## ****9. Type Assertion****

* **Definition:** Tells the compiler about the specific type of a variable when it cannot infer it automatically. Does not change type at runtime.
* **Example:**

typescript

CopyEdit

let a: any = "test";

(a as string).indexOf("e");

## ****10. Type Casting****

* **Definition:** Converts one data type to another using built-in functions.
* **Example:**

typescript

CopyEdit

let num = Number(12);

let num1 = Number("127");

let str = String("testing type casting");

console.log(num, str);

## ****11. Non-null Assertion Operator****

* **Definition:** The ! operator is used to tell TypeScript a variable is **not null or undefined** at that point.
* **Example:**

typescript

CopyEdit

let d: null | undefined | string = "test1";

d!.charAt(3);

* **Note:** Use with caution as it overrides strict null checks, potentially causing runtime errors.