# TYPESCRIPT CHEATSHEET

## INTRODUCTION TO TYPESCRIPT

*A statically typed superset of JavaScript developed by Microsoft. It compiles to plain JavaScript and is designed to develop large applications.*

### Basic Types

- ***Basic Types:*** Includes `number`, `string`, `boolean`, `null`, `undefined`, `symbol`, and `bigint`.
- ***Arrays:*** Defined by `type[]` or `Array<type>`.
- ***Tuple:*** Fixed-size array where each element has a specified type: `[string, number]`.
- ***Enum:*** A way of giving more friendly names to sets of numeric values.
- ***Any:*** A type that can represent any JavaScript value with no constraints.
- **Void*:*** Used for functions that do not return a value.

### Advanced Types

- ***Union Type:*** Allows a variable to be one of several types: `type1 | type2`.
- ***Intersection Type:*** Combines multiple types into one: `type1 & type2`.
- ***Literal Types:*** Restricts a variable to exact values.
- ***Custom Types:*** Defined using `type` or `interface`.

# TYPESCRIPT VS JAVASCRIPT

## 1. Language Type

***JavaScript:*** Dynamic typing (types are associated with values not variables).
***TypeScript:*** Superset of JavaScript that adds static typing (types are associated with variables).

## 2. Type System

***JavaScript:*** Weakly typed, allows for more flexibility but can lead to more bugs at runtime.
***TypeScript****:* Strongly typed with static type checking, leading to fewer runtime errors and better predictability.

## 3. Compilation

***JavaScript****:* Interpreted directly by browsers.

***TypeScript:*** Compiled into JavaScript before execution, allowing errors to be caught beforehand.

## 4. Tooling Support

***JavaScript:*** Linters and other tools for error checking.

***TypeScript:*** Advanced autocompletion, navigation features, and type checking enhance IDE support.

### 5. Community and Ecosystem

***JavaScript:*** Larger ecosystem with extensive libraries and frameworks.

***TypeScript:*** Growing rapidly, with most modern frameworks (e.g., Angular, Vue 3, and React with create-react-app) offering out-of-the-box TS support.

### 6. Use Cases

***JavaScript:*** Suitable for small to medium projects and quick scripting.

***TypeScript:*** Preferred for large-scale applications due to better maintainability and scalability.

## IMPORTANCE OF TYPESCRIPT

- ***Early Bug Detection:*** By catching errors during development rather than at runtime.
- ***Enhanced Editor Support:*** Provides autocompletion, type inference, and more, which simplifies development.
- ***Improved Code Quality*****:** Enforces typing and makes the code less prone to runtime errors.
- ***Easier Refactoring:*** Safely refactor large codebases with less risk of breaking functionality.
- ***Better Collaboration:*** Makes code easier to understand and maintain when working in teams.
- ***Scalability:*** More suitable for large projects that need maintainability and scalability.

## INTERFACES

***Interfaces:*** Describes the shape of an object, providing strong typing.

***Optional Properties:*** Properties that may or may not be present on an object.

***Readonly Properties:*** Properties that can only be assigned at the time of creation.

***Function Types:*** Interfaces can also define function types.

## CLASSES AND INHERITANCE

***Classes:*** Similar to ES6 classes but with enhanced features.

***Access Modifiers:*** `public`, `private`, `protected` to control accessibility.

***Properties:*** Support for traditional class-based features like properties.

***Inheritance:*** Using `extends` to inherit classes.

## GENERICS

***Generics:*** Provide a way to create reusable components.

***Generic Types:*** Can be applied to functions, interfaces, and classes.

## ADVANCED TECHNIQUES

***Decorators:*** Provide a way to add annotations and a meta-programming syntax for class declarations and members.

***Namespaces and Modules:*** Support for code organization via modules and namespaces.

***Type Assertions:*** Similar to type casting, used to override TypeScript's inferred types.

***Type Guards:*** Techniques to influence the type checking, e.g., `typeof`, `instanceof` guards.

## TOOLING AND COMPILATION

***Compilation Context:*** Configured via `tsconfig.json`, which specifies the root files and compiler options.

***Using TypeScript with JSX:*** TypeScript supports JSX and can be configured to work with frameworks like React.

***Advanced Compiler Options:*** Options like `noImplicitAny`, `strictNullChecks` to enforce strict type-checking.

## DEBUGGING AND LINTING

***Source Maps:*** Allow you to debug your TypeScript files in the browser.

***Linting:*** Tools like TSLint or ESLint (with TypeScript plugin) to maintain code quality.

## INTEGRATION WITH BUILD TOOLS

***Webpack:*** Configuring TypeScript with webpack using `ts-loader`.

***Babel:*** Using Babel to compile TypeScript in projects that already use Babel for JavaScript.

## PRACTICAL APPLICATIONS

***Using with Frameworks:*** Angular, React, and Vue have rich support for TypeScript.

***Node.js:*** Using TypeScript in Node.js applications.

## TYPE ANNOTATIONS

*Type annotations in TypeScript are a way for you to explicitly specify the type of a variable, function parameter, return value, or property within your code. This explicit typing helps catch potential bugs during development and before code execution.*

### Variable Declaration

*When you declare a variable, you can specify its type immediately after its name.*

```
let username: string = "Babu";
let age: number = 30;
```

### Function Parameters

*Type annotations can be applied to the parameters of a function to indicate the expected type of each argument.*

```
function userData(name: string, age: number): string {
  return `Hello, ${name}. You are ${age} years old.`;
}
```

### Function Return Types

*You can also explicitly state what type a function should return.*

```
function add(x: number, y: number): number {
  return x + y;
}
```

### Object Properties

*When defining objects, type annotations can be applied to each property.*

```
let user: { name: string; age: number } = {
  name: "Hari",
  age: 25
};
```

### Array and Tuples

*Specify the types of elements in arrays and tuples.*

```
let scores: number[] = [75, 85, 95];
let tuple: [string, number] = ["hello", 10];
```

## TYPE INFERENCE

*Type inference refers to TypeScript's capability to deduce the types of certain elements based on the initial values or how they are used. This feature reduces the need to explicitly define a type in many situations*

### Variable Initialization

*TypeScript infers the type of a variable from the value assigned to it during initialization.*

```
let isCompleted = false; // boolean inferred
```

### Function Return Inference

*TypeScript automatically infers the return type of a function based on the type of the returned value.*

```
function getLength(item: string) {
  return item.length; // inferred return type is number
}
```

### Array Inference

*When an array is created, TypeScript infers the type based on the elements in the array.*

```
let numbers = [1, 2, 3]; // inferred as number[]
```

*TypeScript's type inference helps keep the code concise and readable while maintaining the safety and robustness provided by the type system. Using type annotations in*

*conjunction with type inference can make your TypeScript code clearer, more reliable, and easier to refactor.*

*Together, type annotations and type inference form a powerful part of TypeScript's type system, helping developers to write safer and more predictable code while still enjoying the flexibility of JavaScript.*